

The CLM Book - Optimized Component Lifecycle Management with Sonatype CLM

Contents

1	How to Use This Book	1
2	Downloads	3
3	Sonatype CLM - Requirements	4
3.1	CLM Server	5
3.2	CLM Web Application	6
3.3	REST API Versioning	7
3.4	Command Line Scanner Requirements	7
3.5	Sonatype CLM for Eclipse Requirements	7
3.6	Sonatype CLM for Hudson / Jenkins Requirements	8
3.7	Sonatype CLM for Maven Requirements	9
3.8	Sonatype CLM for Nexus Pro Requirements	9

3.9	Sonatype CLM for SonarQube Requirements	9
4	Component Lifecycle Management	10
4.1	Increasing Component Usage and Open Source Components	10
4.2	Security Vulnerability and License Compliance Risks	11
4.3	Complicating Factors for CLM	11
4.4	Stages of CLM Adoption and Performance	12
4.5	The Four Requirements for True Component Lifecycle Management	13
4.6	Sonatype and Sonatype CLM	14
4.6.1	Who is Sonatype?	14
4.6.2	What is Sonatype CLM?	14
4.6.3	How does Sonatype CLM work?	15
4.6.4	Which component ecosystems does Sonatype CLM support?	15
4.7	Conclusion	16
5	Sonatype CLM - Server Setup	17
5.1	CLM Server Installation and Configuration	17
5.1.1	Starting CLM Server	18
5.1.2	License Installation	19
5.1.3	CLM Server Directories	21

5.1.4	Running the CLM Server as a Service	21
5.2	Configuration	24
5.2.1	Initial Configuration of CLM Server	25
5.2.2	Running the CLM Server Behind a HTTP Proxy Server	25
5.2.3	Setting the Base URL	26
5.2.4	Appending a User Agent String	26
5.2.5	File Configuration	27
5.2.6	Email Configuration	27
5.2.7	Logging Configuration	28
5.2.8	HTTP Configuration	28
5.2.9	HTTPS/SSL	28
5.2.10	Anonymous Access	29
5.3	Backing Up the CLM Server	29
5.4	Upgrading the CLM Server	30
5.4.1	Upgrade Paths	31
5.4.1.1	Upgrading from Sonatype CLM 1.9x or Later	31
5.4.1.2	Upgrading from Sonatype CLM 1.8x	31
5.4.1.3	Upgrading from Sonatype CLM 1.7x and 1.6x::	32

5.4.1.4	Upgrading from Sonatype CLM 1.5x or Earlier	33
6	Sonatype CLM - Security Administration	34
6.1	User Management	35
6.1.1	Logging in to Sonatype CLM	35
6.1.2	Viewing Notifications	36
6.1.3	Changing the Admin Password	36
6.1.4	Creating a User	37
6.1.5	Editing and Deleting User Information	38
6.2	LDAP Integration	39
6.2.1	Configuring the LDAP Server Connection	40
6.2.2	LDAP Configuration Parameters	42
6.2.3	Mapping LDAP Users to Sonatype CLM	43
6.2.4	LDAP User Parameters	45
6.2.5	Mapping LDAP Groups to Sonatype CLM	46
6.2.6	LDAP Group Parameters	48
6.2.6.1	Static Groups	48
6.2.6.2	Dynamic Groups	49
6.2.7	Verifying LDAP Configuration	50

6.2.7.1	Test Connection	50
6.2.7.2	Check User and Group Mapping	51
6.2.7.3	Check Login	51
6.3	Role Management	52
6.3.1	Viewing Role and Permission Descriptions	52
6.3.2	Assigning Users to Roles	53
6.3.3	Creating Custom Roles	54
6.3.4	Excluding Groups from Search Results	55
7	Sonatype CLM - Policy Management	57
7.1	Organization and Application Management	58
7.1.1	Organizational Structure	59
7.1.2	Creating an Organization	59
7.1.3	Creating an Application	60
7.1.4	Organization, Application, and Inheritance	62
7.1.5	Avoiding Policy Micromanagement	63
7.2	Policy Development	63
7.2.1	Advanced Anatomy of a Policy	64
7.2.2	Risk and Organizational Intent	67

7.3	Policy Creation	67
7.3.1	Step 1: Understand the Policy Intent	69
7.3.2	Step 2: Decide on a Descriptive Policy Name	70
7.3.3	Step 3: Choose an Appropriate Threat Level	71
7.3.4	Step 4: Choose the Application Matching Parameters	73
7.3.5	Step 5: Create Constraints with Conditions	73
7.3.6	Step 6: Set Policy Actions And Notifications	78
7.3.6.1	Actions	78
7.3.6.2	Notifications	79
7.3.6.3	Stages	80
7.4	Policy Elements	82
7.5	Labels	82
7.5.1	Creating, Editing and Deleting a Label	83
7.5.2	Creating a Condition Based on a Label	85
7.6	License Threat Groups	86
7.6.1	Creating, Editing, and Deleting a License Threat Group	87
7.6.2	Creating a Condition Based on a License Threat Group	90
7.6.3	Creating a Condition Based on an Unassigned License Threat Group	90

7.6.4	Tags	91
7.6.5	Creating, Editing, and Deleting Tags	93
7.6.6	Applying a Tag	95
7.6.7	Matching Policies to Specific Applications	96
7.6.8	Viewing Tag-based Policies	97
7.7	Evaluating via the CLM Server	98
7.8	Reviewing Evaluation Results	101
7.9	Importing Policies	108
7.9.1	Sonatype Sample Policy Set	108
7.9.2	Importing a Policy to an Organization	109
7.9.3	Importing a Policy to an Application	110
7.10	Policy Monitoring	111
7.10.1	Setup Policy Monitoring for an Application	113
7.10.2	Configuring Notification Times	116
8	Sonatype CLM - Dashboard	117
8.1	Accessing the Dashboard	118
8.2	Viewing CLM Data in the Dashboard	118
8.2.1	Filters	119

8.2.2	Visual Overview	122
8.3	Highest Risk Violations	125
8.3.1	Newest	126
8.3.2	By Component	127
8.3.3	By Application	129
8.4	Viewing Component Details	131
9	Sonatype CLM - Report	134
9.1	Accessing an Application Composition Report	136
9.2	Reviewing a Report	138
9.2.1	Summary Tab	139
9.2.2	Policy Tab	141
9.2.3	Security Issues Tab	143
9.2.4	License Analysis Tab	144
9.3	Printing and Reevaluating the Report	145
9.4	The Component Information Panel (CIP)	145
9.5	Resolving Security Issues	152
9.5.1	Security Issues	152
9.5.2	The Component Information Panel (CIP)	154

9.5.3	Editing Vulnerability Status	154
9.5.4	Matching to Violations	156
9.6	License Analysis Tab	157
9.6.1	License Threat Group	157
9.6.2	License Analysis	158
9.6.3	The Component Information Panel (CIP)	159
9.6.4	Editing License Status and Information	160
9.7	Component Identification	162
9.7.1	Matching Components	163
9.7.2	Managing Proprietary Components	164
9.7.3	Claiming a Component	167
9.8	Label Overview	169
9.8.1	Where do labels begin?	169
9.8.2	Assigning a Label	171
9.9	Waivers	172
9.9.1	A Use Case for Waivers	172
9.9.2	Adding a Waiver	174
9.9.3	Viewing and Removing a Waiver	175

9.10 Policy Reevaluation	176
9.11 Sonatype CLM PDF Report	178
9.11.1 Creating the PDF	178
9.11.2 Reviewing the PDF	178
10 Sonatype CLM and Repository Management	190
11 Sonatype CLM for Nexus Pro	192
11.1 Repository Health Check (RHC) vs. Sonatype CLM	193
11.2 Connecting Nexus to CLM Server	194
11.3 Accessing CLM Component Information	196
11.4 The Component Information Panel (CIP)	200
11.5 Component Details (CLM)	203
11.6 Sonatype CLM for Nexus Staging	204
11.6.1 Staging Profile Configuration	205
11.6.2 Policy Actions for Staging	206
11.7 Policy Actions for Release Repositories	208
12 Sonatype CLM and Continuous Integration	209
13 Sonatype CLM for Bamboo	211

13.1	Install Sonatype CLM for Bamboo	212
13.2	Configure Sonatype CLM for Bamboo	213
13.3	Adding the Sonatype CLM Analysis Task	215
13.4	Reviewing CLM Policy Results	218
14	Sonatype CLM for Hudson and Jenkins	219
14.1	Installation	220
14.2	Global Configuration	221
14.3	Job Configuration	223
14.4	Inspecting Results	225
15	Sonatype CLM and IDEs	227
16	Sonatype CLM for Eclipse	229
16.1	Installing Sonatype CLM for Eclipse	229
16.2	Configuring Sonatype CLM for Eclipse	231
16.3	Using the Component Info View	235
16.4	Filtering the Component List	240
16.5	Searching for Component Usages	241
16.6	Inspecting Component Details	241

16.7 Migrating to Different Component Versions	242
17 Sonatype CLM for SonarQube	246
17.1 Installation	247
17.2 Configuration	248
17.3 Proxy Configuration	249
17.4 Select the CLM Application	250
17.5 Add and Configure the Sonatype CLM Widget	251
17.6 Accessing the Application Composition Report	252
18 Sonatype CLM for CLI	253
18.1 Downloading Sonatype CLM for CLI	254
18.2 Locating Your Application Identifier	254
18.3 Evaluating an Application	255
18.3.1 Additional Options	256
18.4 Example Evaluation	256
18.5 Using Sonatype CLM for CLI with a CI Server	258
19 Sonatype CLM for Maven	259
19.1 Evaluating Project Components with Sonatype CLM Server	260

19.1.1	Authentication	262
19.1.2	Simplifying Command Line Invocations	263
19.1.3	Skipping Executions	263
19.2	Creating a Component Index	264
19.2.1	Excluding Module Information Files in Continuous Integration Tools	265
19.3	Creating a Component Info Archive for Nexus Pro CLM Edition	266
19.4	Using Sonatype CLM for Maven with Other IDEs	267
19.4.1	Maven Plugin Setup	267
19.4.2	IntelliJ IDEA	268
19.4.3	NetBeans IDE	270
20	Sonatype CLM REST APIs	273
20.1	Component Search REST APIs (v1)	274
20.2	Component Information API (v1)	277
20.3	Application REST APIs (v1)	281
20.4	Violation REST API (v1)	290
20.5	Supported Component Identifiers	296
20.6	Component Search REST APIs (v2)	297
20.7	Component Information API (v2)	300

20.8 Component Evaluation REST APIs (v2)	304
20.9 Application REST APIs (v2)	310
20.10 Violation REST API (v2)	319
20.11 Reports REST API (v2)	326
A Copyright	329

List of Figures

5.1	Installing a Product License on Sonatype CLM Server	19
5.2	Sonatype CLM Server End User License Agreement Window	20
5.3	Installed Product License on Sonatype CLM Server	20
5.4	Application Without Organization v.1.8 UI	31
5.5	Application Without Organization v.1.7 and Earlier UI	33
6.1	Login	35
6.2	Create User	38
6.3	Edit User	39
6.4	Sample LDAP Server Configuration	41
6.5	User Mapping	45
6.6	Group Mapping	48
6.7	Dynamic Group Options	50

6.8	Testing LDAP Server	50
6.9	Checking User Mapping	51
6.10	Checking User Login	51
6.11	Role and Permission Descriptions	52
6.12	Assigning Users to Roles	54
6.13	Assigning Groups Manual Search	56
7.1	Using New Organization button	60
7.2	Using Global Create Button	60
7.3	Using New Application button	61
7.4	Using Global Create Button	61
7.5	Editing a Policy and its Attributes	65
7.6	Using New Policy Button	68
7.7	Using Global Create Button	69
7.8	Naming the Policy	71
7.9	Editing the Policy Threat Level	72
7.10	Example Constraint	74
7.11	Adding Constraints	75
7.12	Policy Actions Example	79

7.13 Policy Notifications Example	80
7.14 Using New Label Button	84
7.15 Using Global Create Button	84
7.16 Label Example	85
7.17 Creating a Label Condition	86
7.18 Using New License Threat Group Button	88
7.19 Using Global Create Button	88
7.20 Creating a License Threat Group	89
7.21 Creating a Condition Evaluating a License Threat Group	90
7.22 Creating a Condition Evaluating an unassigned License Threat Group	91
7.23 Example of Applied Tags	92
7.24 Using New Tag Button	93
7.25 Using Global Create Button	93
7.26 Creating a Tag	94
7.27 Example of Tags with Description	95
7.28 Evaluate an Application	99
7.29 Violations Report after Scan	100
7.30 Reporting Area	102

7.31 Application Area	103
7.32 Summary Tab of an Application Composition Report	104
7.33 Policy Tab of an Application Composition Report	105
7.34 Security Issues Tab of an Application Composition Report	105
7.35 License Analysis Tab of an Application Composition Report	106
7.36 Component Information Panel CIP for a Specific Component	106
7.37 Policy Section for a Specific Component Displayed on the Component Information Panel	107
7.38 Organization View with Import Button	109
7.39 Import Policy Dialog	110
7.40 Example of a Policy Monitoring Email	112
7.41 Access Application Management Area	113
7.42 Selecting a Sonatype CLM Stage to Monitor	114
7.43 Adding Email Recipient	114
7.44 Policy Monitoring Notification Example	115
7.45 Sample Email Notification	115
8.1 Dashboard Default View	117
8.2 Accessing the Dashboard	119
8.3 Dashboard Filter Example	119

8.4	Filtering the Dashboard	120
8.5	Dashboard Visuals	122
8.6	Counts	123
8.7	Matches	123
8.8	Policy Violation Trends	124
8.9	Newest Risk	126
8.10	Highest Risk - By Component	128
8.11	Highest Risk - By Application	130
8.12	Component Detail Page	132
9.1	Summary Tab of the Application Composition Report	135
9.2	Reporting Area	137
9.3	Application Area	138
9.4	The Four Tabs	138
9.5	Security Issues Summary	140
9.6	License Analysis Summary	141
9.7	Policy Tab	142
9.8	Security Issues Tab	143
9.9	License Analysis Tab	144

9.10 Application Composition Report Buttons For Printing and Reevaluation	145
9.11 Component Information Panel CIP Example	146
9.12 CIP, Policy Section	148
9.13 CIP, Similar Section	148
9.14 CIP, Occurrences Section	149
9.15 CIP, Licenses Section	150
9.16 CIP, Edit Vulnerabilities Section	150
9.17 CIP, Labels Section	151
9.18 CIP, Claim Component	151
9.19 CIP, Audit	152
9.20 Security Issues Tab	153
9.21 Component Information Panel (CIP)	154
9.22 Security Information Modal	155
9.23 Editing Vulnerabilities	156
9.24 Example of Component with Security Issue, but No Policy Violation	156
9.25 License Analysis Tab	157
9.26 The Default License Threat Groups	158
9.27 Component Information Panel (CIP)	159

9.28	Editing License Using the Select Option	161
9.29	Unknown Component	162
9.30	Filter and Matching Options	163
9.31	Proprietary Component	164
9.32	Proprietary Packages Configuration via the Sonatype CLM Server	166
9.33	Claim a Component	167
9.34	Claimed Component Indicator	168
9.35	Update or Revoke Claimed Component Indicator	168
9.36	Labels at the CLM Server Level	169
9.37	Assigning a Label	171
9.38	Waiver Visualization on Policy Tab	173
9.39	Waiver Button	174
9.40	Options to Apply Waiver to the Application or the Entire Organization	175
9.41	View and Remove Waivers	176
9.42	Application Composition Report Buttons For Printing and Reevaluation	177
9.43	Summary Section of a Application Composition Report in PDF Format	180
9.44	Policy Violations Section of a Application Composition Report in PDF Format	182
9.45	Security Issues Section of a Application Composition Report in PDF Format	184

9.46 License Analysis Section of a Application Composition Report in PDF Format	186
9.47 Components Section of a Application Composition Report in PDF Format	188
10.1 The Central Role of A Repository Manager in Your Infrastructure	191
11.1 CLM configuration tab in Nexus	195
11.2 CLM configuration tab after Test Connection	196
11.3 Typical Search Results in Nexus Pro	197
11.4 Nexus Search Showing All Versions	198
11.5 Accessing the Component Info Tab	199
11.6 Component Information Panel	200
11.7 Component Information Panel Example	201
11.8 CIP Text	201
11.9 CIP Graph	202
11.10View Details Button	203
11.11View Details	204
11.12Staging Profile with a CLM Application Configured	206
11.13Staging and Release Configuration for a Policy in the CLM Server	207
11.14Staging Repository Activity with a CLM Evaluation Failure and Details	208

14.1 Jenkins Global Configuration Menu	220
14.2 Global Configuration of Sonatype CLM for CI in Jenkins	221
14.3 Sonatype CLM Build Scan Configuration for a Build Step	223
14.4 Job Overview Page with a Link to the Application Composition Report	226
14.5 Left Menu with Link to the Application Composition Report	226
16.1 Eclipse Dialog to Install New Software with Sonatype CLM for Eclipse	230
16.2 Activating the Component Info View of Sonatype CLM for Eclipse	231
16.3 Warning after initial installation	232
16.4 Sonatype CLM for Eclipse Configuration Dialog	233
16.5 Example Component Info View	235
16.6 Details for a Component in the Component Info View	237
16.7 Properties of a Component for a Version Range	239
16.8 Filter Dialog for the Component Info View	240
16.9 Example Component Details Display	242
16.10 Migrating to a Newer Component Version	243
16.11 Applying a Dependency Version Upgrade	244
16.12 Selecting Dependency Version or Property Upgrade	244
16.13 Applying a Property Upgrade	245

17.1 SonarQube Overview	247
17.2 SonarQube Plugin Directory	248
17.3 SonarQube Settings Menu	249
17.4 SonarQube CLM Server Settings	249
17.5 SonarQube Sonatype CLM Configuration Menu	250
17.6 SonarQube Sonatype CLM Application Selection	251
17.7 SonarQube Configure Widgets Menu	251
17.8 SonarQube Search for CLM Widget	251
17.9 SonarQube Configure Sonatype CLM Widget options	252
17.10 SonarQube Sonatype CLM Widget Example	252
18.1 Application Overview and Application Identifier	254
18.2 Violations Report After an Evaluation	257
19.1 Creating a Maven Run Configuration for a CLM Evaluation in IntelliJ	269
19.2 Maven Projects View with the CLM Evaluation Run Configuration in IntelliJ	269
19.3 CLM for Maven Output in the Run Console in IntelliJ	270
19.4 Project View with the pom.xml in NetBeans	271
19.5 Maven Goal Setup for a CLM Evaluation in NetBeans	271
19.6 CLM for Maven Output in the Output Window in NetBeans	272

List of Tables

7.1 Threat Levels	71
-------------------------	----

Preface

A book covering the concepts and practices of component lifecycle management in general and specifically usage of the Sonatype CLM suite of tools.

Last updated and published on 2015-08-05.

Chapter 1

How to Use This Book

The Sonatype CLM Book you are reading right now is the first source of knowledge for using the different components of the Sonatype CLM suite of tools. It is separated into a number of different sections.

The navigation bar on the left-hand side of the documentation allows you to access all the different sections directly.

The top right-hand side of the documentation features a search input box that accesses all sections and documents to provide you with the relevant document for your search term.

A user interface label or button is highlighted. E.g. use the *Applications* button to access the list of applications in the CLM server.

A code segment or command line input in a paragraph would be looking like this `mvn clean install`.

Larger code segments are encapsulated in a block:

```
$tar xfvz sonatype-clm-server-1.12.0-bundle.tar.gz
x sonatype-clm-server-1.12.0.jar
x demo.sh
x config.yml
x eula.html
x demo.bat
x README.txt
```

The documentation uses specific blocks for notes, tips and important images:

Note

You can download Java from the Oracle website.

Tip

The CLM server should be set up to run as a service.



Important

CLM server requires Java 7.

In addition there are blocks for warnings and cautioning alerts:



Warning

Importing a policy will erase existing policies.



Caution

Upgrading components might require you to refactor your codebase.

Application screenshots and other images are typically included as numbered figures and referenced from the flowing text.

Chapter 2

Downloads

The latest release of the CLM Server and all applicable tools can be downloaded from the [Sonatype support website](#) and is available as *.tar.gz* or *.zip* archive. The contents of the two files are identical and you can choose to download either one. Successful download should result in files named `sonatype-clm-server-xyz-bundle.tar.gz` or `sonatype-clm-server-xyz-bundle.zip`, where `xyz` is the version of the latest release e.g. `1.11.0-01` or `1.10.2`.

Chapter 3

Sonatype CLM - Requirements

Sonatype CLM consists of a variety of components, which include:

- CLM Server
- Command line scanner
- Sonatype CLM for Eclipse
- Sonatype CLM for Hudson
- Sonatype CLM for Jenkins
- Sonatype CLM for Maven
- Sonatype CLM for Nexus Pro
- Sonatype CLM for SonarQube

For each of these components, there are a variety of requirements. These requirements are included in the respective installation sections. However, they have also been gathered here for quick reference.

3.1 CLM Server

Sonatype CLM Server is typically deployed on a dedicated server. More specific hardware requirements are ultimately a function of the deployment architecture, the primary usage patterns and the scale of deployment.

With these influencing factors in mind, we recommend a modern processor speed with at least 8 CPU cores and 8GB of physical RAM for initial setup. A minimum of 6GB of process space should be available to the CLM server. Additional RAM can improve the performance due to decreased disk caching.

As an example a Sonatype CLM Server deployment at Sonatype is using a Dual Intel Xeon E5620 with 2.4Ghz, 12M Cache, 5.86 GT/s QPI, Turbo, HT.

Storage requirements range with the number of applications projected to use the CLM server. For less than 1000 applications 200GB is recommended. Between 1000 and 2000 applications we suggest to use 500GB. Above 2000 applications we suggest to provide 1TB of storage.

Tip

Monitoring disk-space usage will help you gauge the storage needs in your actual deployment and react to growing demands in time.

Sonatype CLM Server is an I/O intensive application and disk speed will affect the performance of the CLM server considerably. We therefore recommend to use local drives or SAN usage. Usage of network mapped storage via NFS or similar is not recommended. It is important to consider the I/O load when running CLM server in a virtual machine, especially when other virtual machines on the same host are running other I/O intensive applications e.g. the Nexus repository manager.

Development, test or evaluation deployments can be scaled smaller than the above recommendations and will continue to function, while a minor performance degradation can be observed.

CPU and RAM

We recommend any reasonably modern 2-4 core CPU with 4GB of RAM. While individual needs can vary, this should generally be sufficient for normal uses of CLM.

Disk

By modern standards, 500GB to 1 TB of free disk space should provide more than adequate resources.

Operating System

Generally, any machine that can run a supported Sun/Oracle Java version should work. Refer to the Oracle documentation for specifics: Oracle JDK 1.7 to 1.8 and JRE 1.7 to 1.8 Certified System Configurations. The most widely used operating system for CLM is Linux and therefore customers should consider it the best tested platform.

Ports

The following ports are used for communication and should be addressed according to your Firewall configuration.

8070

Used by all CLM clients to access the CLM Server. This port is configurable.

8071

Used by the local host or other IT monitoring tools for monitoring and operating functions. This port is optional and configurable. If omitted port 8081 will be used.

443

Used by the CLM server to securely access Sonatype's Hosted Data Services (HDS). This port is not configurable.

Java

Oracle Java 1.7 to 1.8 64 bit

3.2 CLM Web Application

Browser

In all cases JavaScript must be enabled for the chosen browser.

Internet Explorer

- IE 9
- IE 10
- IE 11

Firefox

- ESR (extended support release)
- "Stable"

Chrome

"Stable"

Safari (On OSX)

- 5.1.9 corresponding to OS X 10.6
 - 6.0.4 corresponding to OS X 10.7 and 10.8
 - 7.1 corresponding to OS 10.9
-

3.3 REST API Versioning

Like many REST APIs, Sonatype CLM REST APIs are versioned. As a best practice we recommend using the latest version of the REST APIs. This ensures your system will take advantage of the latest features and improvements.

However, we also realize that users of previous API versions need to maintain this compatibility even when there is an update. For this reason, we do provide support for some older versions of the APIs functioning and accessible.

Supported API Versions

- CLM 1.12 and earlier - **Only** Supports REST API v1
- CLM 1.13 and later - Supports **Both** REST API v1 and v2

Identifying the version of the API is simple. Below we have provided an example using the REST API for retrieval of an organization ID.

```
http://localhost:8070/api/v2/organizations
```

As you can see, the `v2` located just after `api` indicates the version of the API. If you find that the API version you are using is not documented, and would like information on upgrading to the latest version you can [contact our support team](#) for assistance.

3.4 Command Line Scanner Requirements

The Command Line Scanner requires Java be installed on the machine running the application, supports version 1.6 to 1.8.

3.5 Sonatype CLM for Eclipse Requirements

General Requirements , Eclipse

3.6 (Helios) to 4.4 (Luna)

m2e

(currently supports 1.0 - 1.5)

Java

Oracle Java 7 update 6 or newer (7u6+)

Browser Requirements:

In most cases the Eclipse plugin will render browser content using its own browser widget based on JavaFX.

Where JavaFX is not available the plugin will use the system browser. To properly render content the minimum requirement is Internet Explorer (IE) 9, or another modern browser such as Chrome / Firefox.

Note

Sonatype CLM functionality is not supported for those running Sonatype CLM for Eclipse with Internet Explorer 8.

**Warning**

These requirements assume that an installation of the currently released version of the Sonatype CLM for Eclipse plugin and the compatible Sonatype CLM Server. Before picking a version please [verify compatibility](#).

3.6 Sonatype CLM for Hudson / Jenkins Requirements

Hudson

- 2.2.1
- 3.0.0
- 3.1.2

Jenkins

- 1.447.2
-

- 1.532.1 (LTS)
- 1.550

Java

1.6 to 1.8

3.7 Sonatype CLM for Maven Requirements

Maven

2.1 to 3.2.3

Java

1.6 to 1.8

3.8 Sonatype CLM for Nexus Pro Requirements

Sonatype CLM for Nexus is a part of the Nexus CLM Edition, and does not require installation of an specialized components. However, the CLM capabilities do require a user have at least **Internet Explorer 9** or any other **modern browser**.

For information on Nexus requirements please see the [support article detailing Sonatype Nexus System Requirement](#) and the [prerequisites section of the Nexus Book](#).

3.9 Sonatype CLM for SonarQube Requirements

SonarQube

3.7.4 (LTS) 4.3.2 4.4

Java

1.6 to 1.8

Chapter 4

Component Lifecycle Management

Component lifecycle management can be defined as the **practice of**

- **analyzing,**
- **controlling,** and
- **monitoring**

the components used in your software development lifecycle.

It has emerged as a new category of software development products, information services, and practices that help manage agile and collaborative, component-based development efforts. With modern component lifecycle management, you are able to ensure the integrity of the modern software supply chain and amplify the benefits of modern development, all while reducing risk.

4.1 Increasing Component Usage and Open Source Components

Modern software development practices have shifted dramatically from large efforts of writing new code, to the precise usage of components to assemble applications. This approach limits the amount of code authorship to the business-specific aspects of your software.

As such, a large number of open source components in the form of libraries, reusable widgets, whole applications, application servers, and others are now available. These components are of a high quality, and support use cases that could not be implemented as a side-effect of your business application development. For example, creating a new web application framework and business workflow system just to create a website with a publishing workflow would be extremely inefficient.

Because of these changes and move to efficiency, open source has become an integral part of modern applications. So much so, that a typical enterprise application is now comprised of tens, if not hundreds, of components, which accounts for 80% and more of the application.

4.2 Security Vulnerability and License Compliance Risks

With the huge benefits derived from using open source, as well as commercial components, comes the complexity of understanding all the implications they have to your software delivery. This includes security vulnerabilities, license compliance problems, and quality issues, all of which need to be managed through the whole life cycle. This starts at the inception of the application all the way through development, quality assurance, production deployments, and even on through decommissioning of the application.

Given the number of components, their rapid change rate, and the ease of adding new dependencies, it quickly becomes clear that the management and full understanding of all components associated with an application is a daunting task, one that can not be carried out manually. Luckily, this is simplified with the assistance of tools such as Sonatype CLM.

4.3 Complicating Factors for CLM

As the owner of a specific application you are responsible for any license, security and quality issues arising. For the common approach of a component based software product, one that reuses a lot of externally available components, this means you are not only responsible for your source code, but also for any issues originating from the components you used.

Needless to say, tracking all these components and potential issues is complicated due to a number of factors, including:

Complexity

A typical application relies on a number of components, that in turn rely on others, which builds up a complex dependency graph that can be hard to grasp and manage without proper tooling.

Diversity

The number of components available from open source projects is staggering and continuously rising, together with the choice of commercial components, you are faced with a hugely diverse set of choices. All of which will impact your CLM efforts.

Adaptability

Component authors strive to provide a rapid rate of innovation, which brings you a larger number of benefits. Unfortunately, this rate of change means that you need to keep up in order to stay current, and take advantage of new features as well as fixes. Not to mention, many times new feature mean new issues.

4.4 Stages of CLM Adoption and Performance

When endeavoring to initially implement, and subsequently establish, CLM as an ongoing process, a number of stages and actions are commonly required:

Integrate Sonatype CLM Sample Policy

What can tend to be the most difficult part to new users of Sonatype CLM is something that is developed outside the application, policies. Policies are a set of rules that you expect a component to meet as it relates to a particular application. These rules should include the level of risk you are willing to accept. Given this, policies starts as a statement of what you do and do not desire to be included in your applications. This is something that is dynamic though. Meaning, that over time your policies will change and evolve to adapt to your business. So, instead of trying to determine that all upfront, make your first stage on of seeking out the sample policies we've provided to get you started.

Improve Component Selection

With policies created (or hopefully the sample policies implemented), it can be enticing to begin calling a full stop on development when something negative is found. While that is an approach, it's not the recommended path. Instead, start by only implementing the developer set of CLM tools. This will allow you to expose your development teams to the information that Sonatype CLM has. When they encounter components that would violate a policy, it will be apparent. They will also be able to easily select alternatives, by quickly finding the best version. Development teams want to do the best job they can, and this stage puts them first and foremost in improving your applications, they way it should be.

Establish Component Inventory and Governance

The component selection phase allowed the development team to make better choices for components they use. Now that they are familiar with the type of information the Sonatype CLM provides,

it is time to start tracking the inventory and approval of components used in applications that make up the enterprise.

Sonatype CLM provides tools to integrate into the build release management systems to validate and ensure the components in use are Sonatype CLM policy approved components. Governance sets the expectations of what components will be approved and allows for starting the dialog, with the Development teams, to provide business justification for why a risky component should be allowed.

Monitor Component Usage

At this point, you will also need to make sure security and licensing policies have been established, and are continually reviewed and updated. This works most effectively if carried out during your ongoing development efforts, as well as for any components already in production. Ultimately, this will allow you to both preemptively address any issues, or react to any that are newly discovered. Remember to evaluate your applications often, and at major milestones during development (e.g. during builds and when staging a release). In this final stage, you should begin to consider putting gates, which Sonatype CLM provides, making sure a balance between ongoing fluid development and releasing software with unwanted components is achieved.

4.5 The Four Requirements for True Component Lifecycle Management

To truly manage components in a way that reduces risk and encourages fast-paced development, you must have a solution capable of being . . .

Precise

identifies components, exactly, even when altered.

Actionable

provides detailed security, quality, and licensing information to be able to make conclusions and take action.

Complete

analyzes the entire component including dependencies.

Update Aware

notifies users of new versions with detailed update reasoning

Integrated

blends seamlessly with existing software development lifecycle tools

4.6 Sonatype and Sonatype CLM

With process of CLM outlined, you clearly begin to see the need for a set of tools that can integrate into this new way of thinking. This is where Sonatype and Sonatype CLM come into play. Of course that presents several new questions:

- Who is Sonatype?
- What is Sonatype CLM?
- How does Sonatype CLM Work?
- Which component ecosystems does Sonatype CLM support?

4.6.1 Who is Sonatype?

Maven, Nexus, and The Central Repository are perhaps the most familiar names associated with Sonatype, and Sonatype CLM is the newest name to join these ranks.

If you're interested in the Sonatype story, head over to our web page, and read all [about Sonatype](#) and the best place to [get started](#).

4.6.2 What is Sonatype CLM?

Sonatype CLM is the suite of tools and products dedicated to optimizing your component lifecycle management efforts. We do this by offering products which allow different stake holders and participants to collaborate on their CLM efforts, with suitable, easy-to-use tools that add value across the enterprise.

In essence we are working to do away with the scan and scold mentality of managing components. Instead we want to provide your teams with the information they need, early in the development process. We believe that being able to make informed decisions when selecting components is essential to success.

Sonatype CLM supports a number of different tasks and activities, including, but not limited to:

- Development in an IDE
-

- Build via Continuous Integration Servers
- Release Management
- Production Monitoring

This integration can vary from embedded GUI interfaces all the way to custom plugins and command line functionality. The real thing to remember is that no matter what system you are using, Sonatype CLM can find a way to evaluate your application and deliver the necessary information to everyone involved.

4.6.3 How does Sonatype CLM work?

Sonatype CLM makes use of Sonatype's HDS (Hosted Data Services) which logs security, license, and architecture information for supported open source component ecosystems.

This information is, in some cases exclusive to Sonatype (i.e. the Central Repository), and in all cases, constantly being refined using the most up-to-date sources as possible.

The type of information available for components can vary, but generally revolves around these three:

- Security Vulnerabilities
- License Analysis
- Architecture (e.g. Age, Popularity and Usage)

4.6.4 Which component ecosystems does Sonatype CLM support?

Sonatype CLM isn't just about Java components, though that's where we started. As you will see, we also provide detailed information for other component systems, such as NuGet (associated with Microsoft .Net Development), as well. A brief description of each is provided below.

The Central Repository and Java

Managed by Sonatype, the Central Repository (or "Central") is the largest repository of components for the Java ecosystem. It is the default repository in Apache Maven, which is the most widely used build system for Java development. It is also available as a repository in most other build systems as a component source.

Of course, Central isn't the only repository we collect information on, access to additional repositories is available as well.

NuGet and the .NET Framework

Originally founded by Microsoft, but now independently managed by the [Outcurve foundation](#), NuGet provides open source components for .NET development.

In cooperation with Microsoft and NuGet, Sonatype's HDS regularly reviews NuGet packages to match any known issues related to license, security, and architecture.

4.7 Conclusion

You have likely heard, "We work with customers just like you." In many ways this is often true. While the internal workings of our businesses are different, the problems we are looking to solve (or resolve) are quite similar. Given this, we too have worked with customers just like you.

This is a good thing though. The information presented here, as well as in all of our materials, and even the design of Sonatype CLM all is heavily influenced by what our customers request. In this fashion, we have set our primary goal to providing you with solutions that let you do what you do best, develop great software. Through that, we are here to support and guide your efforts with the best information possible.

So, whether your journey toward component lifecycle management has just started, or in some cases may be well on its way, we're here to help. If at any point you feel we missed something, or have questions you can't find answers to, don't hesitate contacting us. Good luck, and we look forward to hearing about your success.

The Sonatype CLM Team

Chapter 5

Sonatype CLM - Server Setup

The Sonatype CLM Server is the central component of your Sonatype CLM implementation. It allows you to create and store policies, manage access, set configuration items, and review the effect of your policies through a variety of reports.

In addition, it will be the centralized location for ensuring seamless integration to systems such as Eclipse, Hudson / Jenkins, and/or Nexus.

The following chapter will provide everything you need to install and deploy the Sonatype CLM server, as well as make sure you can setup and define access for your team.

5.1 CLM Server Installation and Configuration

Before installation, please check check the [CLM Server requirements](#).

After a successful [download](#) of the CLM server bundle archive, you should create an installation directory in the desired location and move the archive into the directory.

```
cd /opt
mkdir sonatype-clm-server
mv ~/Downloads/sonatype-clm-server.* sonatype-clm-server/
cd sonatype-clm-server
```

Moving into the directory and extracting the archive with either one of the commands:

```
unzip sonatype-clm-server*.zip
tar xfvz sonatype-clm-server*.tar.gz
```

should result in a directory with the following files:

```
README.txt
config.yml
demo.bat
demo.sh
eula.html
sonatype-clm-server-1.10.2-bundle.tar.gz
sonatype-clm-server-1.10.2.jar
```

5.1.1 Starting CLM Server

Once the CLM server is installed, it can be started with:

```
cd /opt/sonatype-clm-server
java -jar sonatype-clm-server-*.jar server config.yml
```

This command will start the server with the Sonatype CLM Server application using the configuration from the config.yml file and logging any output straight to the console. After a complete start your console should display a message similar to:

```
... [main] org.eclipse.jetty.server.AbstractConnector - Started ↔
InstrumentedBlockingChannelConnector@0.0.0.0:8070
... [main] org.eclipse.jetty.server.AbstractConnector - Started ↔
SocketConnector@0.0.0.0:8071
```

The command to start the server can be modified by adding java configurations parameters such as `-Xmx1024m -XX:MaxPermSize=128m` to improve performance and adapt to the server hardware.

At this stage you can access the web application at port 8070 of your server via any web browser. Initial startup will display a screen for the [Section 5.1.2](#).

5.1.2 License Installation

Sonatype CLM server requires a license to be installed. The required license file will be supplied to you by the Sonatype support team in the form of a `.lic` file.

Open a web browser and navigate to the CLM server web application at port 8070 to install the license. Opening the URL, e.g. for a localhost deployment at <http://localhost:8070>, displays the Product License Configuration of the CLM server shown in Figure 5.1.

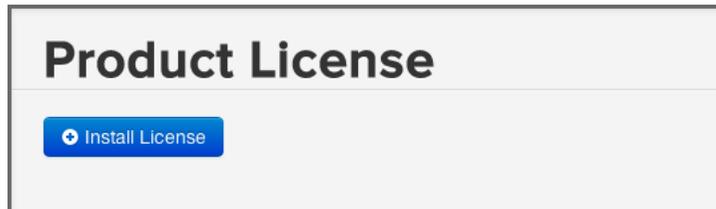


Figure 5.1: Installing a Product License on Sonatype CLM Server

Press the *Install License* button and select the `.lic` file in the file selector. As a next step you are required to accept the end user license agreement shown in Figure 5.2 by pressing the *I Accept* button.

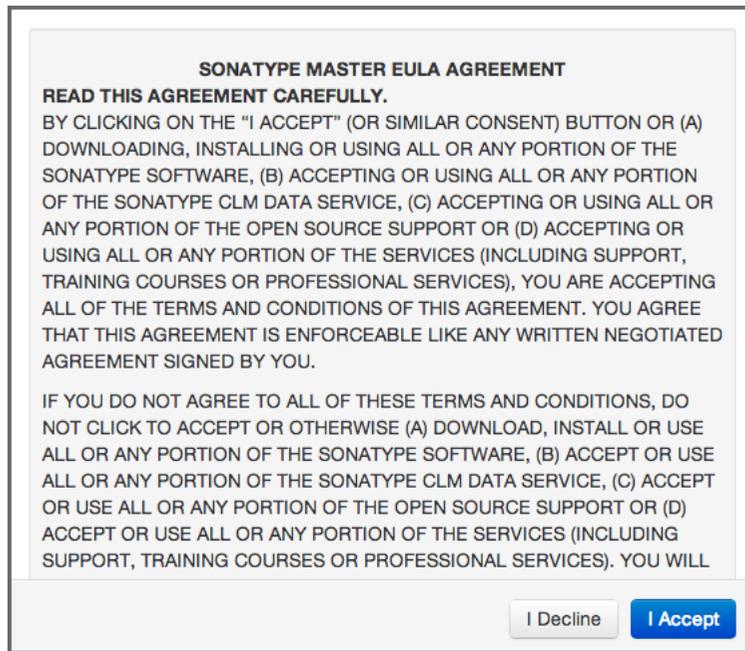


Figure 5.2: Sonatype CLM Server End User License Agreement Window

After a success message you will be redirected to the Product License page, which will now display the expiry date of the license as visible in Figure 5.3.



Figure 5.3: Installed Product License on Sonatype CLM Server

5.1.3 CLM Server Directories

When the Sonatype CLM Server first starts, it creates a directory for the storage of all its data and configuration. This directory is configured in `config.yml` and defaults to `./sonatype-work/clm-server`. This path is relative to the location from which the invoking `java` command is used.

Using the default startup command from the installation directory, causes `sonatype-work/clm-server` to be created within it.

If you would like to separate the installation and data directories you can set the `sonatypeWork` to a different location.

Additionally a `log` directory is created within the installation directory and the `currentLogFileName` parameter in `config.yml` can be used to change the location. Further information on logging configuration can be found in Section [5.2.7](#).

5.1.4 Running the CLM Server as a Service

For production usage, we strongly recommend to set up the CLM server as a service or daemon. This will ensure that any operating system reboots will include starting up the CLM server.

A dedicated user for running a service is a well known best practice. This user should have reduced access rights as compared to the root user. Configuration of this user will depend on the operating system and security system used.

Once the user is configured, you need to ensure that full access rights to the CLM server installation directory are granted. An example command to achieve this for a service user with the username `clmserver` is:

```
chown -Rv clmserver /opt/sonatype-clm-server
```

If you have configured the `sonatypeWork` parameter in `config.yml` to point to a different directory, you have to adjust the access rights for it as well.

The principal command for starting the CLM server can be used in a simple startup script as displayed in [Startup Script](#). The `javaopts` variable should be adjusted to suit the hardware used.

Startup Script

```
#!/bin/sh
cd /opt/sonatype-clm-server
javaopts="-Xmx1024m -XX:MaxPermSize=128m"
java $javaopts -jar sonatype-clm-server-*.jar server config.yml
```

A running server can be stopped with a simple shutdown script in [Shutdown script](#).

Shutdown script

```
#!/bin/sh
pid=`ps aux | grep sonatype-clm-server | grep -v grep | awk '{print ←
    $2}' `
kill $pid
```

Typically these approaches are combined to a service script similar to the script listed in [Simplistic Service Script for Unix Systems](#). Saving this script as e.g. `sonatype-clm-server` allows you to start the server with its log running to the current shell with

```
./sonatype-clm-server console
```

Starting as a background process can be initiated with:

```
./sonatype-clm-server start
```

and a running background server can be stopped with:

```
./sonatype-clm-server stop
```

This example script can be improved to be more robust against repeat invocations, long running stops and potentially work better across different Unix flavors, but shows the principal functionality. A similar script can be used for Windows.

Simplistic Service Script for Unix Systems

```
#!/bin/sh

# The following comment lines are used by the init setup script like the
# chkconfig command for RedHat based distributions. Change as
# appropriate for your installation.
```

```
### BEGIN INIT INFO
# Provides:          clm-server
# Required-Start:    $local_fs $remote_fs $network $time $named
# Required-Stop:     $local_fs $remote_fs $network $time $named
# Default-Start:     3 5
# Default-Stop:      0 1 2 6
# Short-Description: Sonatype clm-server service
# Description:       Start the Sonatype clm-server service
### END INIT INFO

SONATYPE_CLM_SERVER_HOME=/opt/tools/sonatype-clm-server
VERSION=1.12.0
JAVA_OPTIONS="-Xmx1024m -XX:MaxPermSize=128m"
# The user ID which should be used to run the CLM server
# # IMPORTANT - Make sure that the user has the required privileges to ←
# write into the CLM work directory.
RUN_AS_USER=clm

do_start()
{
    cd $SONATYPE_CLM_SERVER_HOME
    su -m $RUN_AS_USER -c "java -jar $JAVA_OPTIONS sonatype-clm-server- ←
    $VERSION.jar server config.yml > /dev/null 2>&1 &"
    echo "Started Sonatype CLM Server"
}

do_console()
{
    cd $SONATYPE_CLM_SERVER_HOME
    java -jar $JAVA_OPTIONS sonatype-clm-server-$VERSION.jar server config ←
    .yml
}

do_stop()
{
    pid=`ps aux | grep sonatype-clm-server | grep -v grep | awk '{print $2 ←
    }'`
    kill $pid
    echo "Killed Sonatype CLM Server - PID $pid"
}

do_usage()
{
    echo "Usage: clm [console|start|stop]"
}

case $1 in
console) do_console
;;
```

```
start) do_start
;;
stop) do_stop
;;
*) do_usage
;;
esac
```

Setting up this script as a startup script will vary between operating systems and distributions depending on the init system used. Generally the script would be copied to a dedicated startup directory and assigned with run-levels and other characteristics for the start up. As an example on a Debian based systems the following commands could be used:

```
sudo su
cp sonatype-clm-server /etc/init.d/
cd /etc/init.d
update-rc.d sonatype-clm-server defaults
service sonatype-clm-server start
```

Depending on the requirements from your system administrator the scripts will have to be modified to fit into your environment and exact deployment scenario.

Tip

Our support team can assist you with operating system and Linux distribution-specific tips and tricks regarding the startup script and installation.

5.2 Configuration

The main configuration file for the CLM server installation is a YML formatted file called *config.yml* found in the installation directory. CLM server is an application running on a [Dropwizard](#) server.

In addition a number of configuration steps can be taken within the running server user interface.

This section will discuss various configuration options in the config file as well as some other configuration scenarios. When editing the file it is important to preserve the indentations, since they are significant for the resulting values created when parsing the configuration file. Generic configuration information can be found in the [Dropwizard User Manual](#).

Tip

The `config.yml` format does not support tab characters. Use an editor that displays special characters like tabs when editing the file.

5.2.1 Initial Configuration of CLM Server

Besides the license installation mentioned earlier, there are a few further configuration steps you should consider before diving right into using the CLM server. You can configure various aspects in the CLM *System Preferences* section of the CLM server user interface, which you can access by clicking on the

System Preferences icon  located in the top right of the CLM Header/Screen (resembles a cog/gear) and choose the desired option to configure:

- Configure *Users* and *Roles* in the *System Preferences* menu, potentially combined with *LDAP* as well. Read more about the security setup outlined in the Security Administration documentation.
- Configure *Proprietary Packages* so that the CLM server can distinguish your own code from other unknown components. For more information, refer to the component match and identification documentation in the report chapter.
- Inspect or update or configure your *Product License*

5.2.2 Running the CLM Server Behind a HTTP Proxy Server

Many organizations filter, control and optimize access to the internet via a proxy server. Any server or even any computer within the organization is forced to connect to the internet via the proxy server. The Sonatype CLM Server needs to communicate with the Sonatype CLM backend services hosted on the internet.

To allow the CLM server to connect via a proxy, you have to specify the connection details in the `proxy` section of the `config.yml` file displayed in [Proxy Configuration in config.yml](#).

Proxy Configuration in config.yml

```
proxy:
  hostname: "127.0.0.1"
  port: 80
```

```
username: "anonymous"  
password: "guest"
```

If your proxy server is based on whitelisted URLs, you can use the following list of URLs to ensure that the CLM server can reach all the required services.

- <https://clm.sonatype.com>
- <http://cdn.sonatype.com/>

5.2.3 Setting the Base URL

If your CLM server is accessed via a https proxy or a proxy server that changes the http port or for other reasons can potentially not determine what the authoritative URL to access the server itself is, you need to configure the `baseUrl` parameter.

```
baseUrl: http://clm-server.example.com/
```

It is used by the server for any user facing links e.g. located in email notifications sent by the server to direct users to the server.

5.2.4 Appending a User Agent String

To address the firewall configurations set by some organizations, Sonatype CLM allows you to customize the user agent header used for HTTP requests. To add a user agent string to Sonatype CLM add the following line to your `config.yml`:

```
userAgentSuffix: "test string"
```

Note

Control characters are not permitted, and the max length of the string is 128 characters.

5.2.5 File Configuration

Sonatype CLM Server stores various files and data related to its operations in a work directory. By default this data is stored in a `sonatype-work/clm-server/` directory in the path the server runs. The directory is configurable using the `sonatypeWork` field in [File Configuration in `config.yml`](#).

File Configuration in `config.yml`

```
sonatypeWork: ./sonatype-work/clm-server
```

In addition, Sonatype CLM Server uses the system temporary directory during its operation. This folder varies by operating system but is usually controlled by an environmental variable. If a specific directory needs to be used, the CLM Server can be started with a command line flag as such:

```
cd /opt/sonatype-clm-server
java -jar -Djava.io.tmpdir=/path/to/tmpdir sonatype-clm-server-*.jar ↔
server config.yml
```

Note that the user account which the server runs under must have sufficient access rights to both the work and temporary directory in order for Sonatype CLM Server to function properly.

5.2.6 Email Configuration

The Sonatype CLM Server can be configured to send email notifications for events such as policy violation notifications. This functionality requires an SMTP server, which is configured along with a number of other options in the `mail` section of the `config.yml` file displayed in [Mail Configuration in `config.yml`](#).

Mail Configuration in `config.yml`

Here's an example configuration:

```
mail:
  hostname: your.mailserver.com
  port: 465
  username: user@company.com
  password: password
  tls: true
  ssl: true
  systemEmail: "SonatypeCLM@localhost"
```

The connection details are established with `hostname` and `port` and optionally with the addition of `username`, `password`, `tls` and `ssl`. The `systemEmail` parameter will be used as the sender email for any emails the CLM server sends. **All fields are required.**

Finally, when setting email configuration, make sure you have also set the [Base URL](#), otherwise sending of notification emails may fail.

5.2.7 Logging Configuration

The CLM server application logging can be configured in the `logging` section of the `config.yml` file. By default a log directory is created in the installation directory and the `clm-server.log` is rotated. Further logging configuration is documented in the [Dropwizard manual](#).

5.2.8 HTTP Configuration

The HTTP configuration in `config.yml` is displayed in [HTTP Configuration in config.yml](#). The `port` parameter for the CLM server allows you to set the port at which the application is available. The `adminPort` exposes the operational menu. Both ports can be freely changed to other values, as long as these port numbers are not used and in the allowed range of values greater than 1024.

HTTP Configuration in config.yml

```
http:
  port: 8070
  adminPort: 8071
```

5.2.9 HTTPS/SSL

One option to expose the CLM server via `https`, is to use an external server like [Apache httpd](#) or [nginx](#) and configure it for reverse proxying the external connections via `https` to internal `http` connection. This reverse proxy can be installed on the same server as the Sonatype CLM server or a different server and numerous tutorials for this setup are available on the internet.

A second option is to directly configure SSL support for Dropwizard by modifying the `http:` segment in the `config.yml` file following the example in [HTTPS Configuration in config.yml](#).

HTTPS Configuration in config.yml

```
http:
  port: 8443
  adminPort: 8471

  connectorType: nonblocking+ssl

  ssl:
    keyStore: /path/to/your/keystore/file
    keyStorePassword: yourpassword
```

The keystore file can be generated and managed with the `keytool`. Further documentation is available in the [Dropwizard documentation](#) and the [documentation for keytool](#).

5.2.10 Anonymous Access

By default the Sonatype CLM Server allows anonymous access when submitting applications for evaluation. If you would like to require users to authenticate their access with a username and password in all situations, add the following line to the `config.yml`:

```
anonymousClientAccessAllowed: false
```

5.3 Backing Up the CLM Server

We highly recommend that you establish a data recovery plan in accordance with your company's policies.

The Sonatype CLM Server keeps all its configuration and data, besides the startup configuration, in the `sonatypeWork` folder as configured in `config.yml`. By default, this folder will be the `sonatype-work/clm-server` folder in your installation directory.

There is a tight coupling between report data stored in the file system and the data stored in the H2 database. Any backup strategy for `sonatypeWork` while the CLM Server is still available to users runs the risk that the database will still be open, leading to a corrupt backup. Therefore, the CLM Server should be shut down before performing the backup.

5.4 Upgrading the CLM Server

As the Sonatype CLM Server is the central element of the Sonatype CLM suite of products, it is often referred to synonymously as Sonatype CLM. The latest version of the Sonatype CLM Server can be downloaded from [the Sonatype support site](#).

The instructions below cover general upgrading instructions. However, depending on the version you are upgrading from, there may be [additional steps](#) needed to complete your unique upgrade path. Please review these accordingly.

Finally, before starting any upgrade, always start by checking our compatibility matrix, [Knowledge Base article](#), and making a [backup](#).

Tip

Before attempting an upgrade, it's best to review all upgrade instructions provided for your current version, as well as any versions that followed.

Upgrade Instructions

1. Determine your [upgrade path](#).
2. Stop the Sonatype CLM Server
3. [Perform a backup](#)
4. Make a copy of sonatype-clm-server/config.yml
5. Copy the new installation bundle into installation folder
6. Extract the bundle
7. Apply all modifications from the backup config.yml to the new config.yml file including e.g. server ports
8. Update any startup scripts as needed
9. Start the Sonatype CLM Server

Note

Before upgrading we highly recommend reviewing the [CLM Server Backup Instructions section](#) above.

5.4.1 Upgrade Paths

It's common for users to upgrade to the latest version from a variety of previous versions. This can present issues in trying to present a clear update path.

We've isolated instructions for upgrading from each previous version. However, if you've been using Sonatype CLM for awhile, it's not a bad idea to review instructions for previous versions you may have used.

5.4.1.1 Upgrading from Sonatype CLM 1.9x or Later

There are no additional steps needed for users of Sonatype CLM 1.9x or higher. Please follow the [standard upgrade instructions](#).

5.4.1.2 Upgrading from Sonatype CLM 1.8x

The biggest change affecting an upgrade from version 1.8x is related to the enforcement that all applications must have a parent organization. Within the interface any applications without an organization were identified as follows:

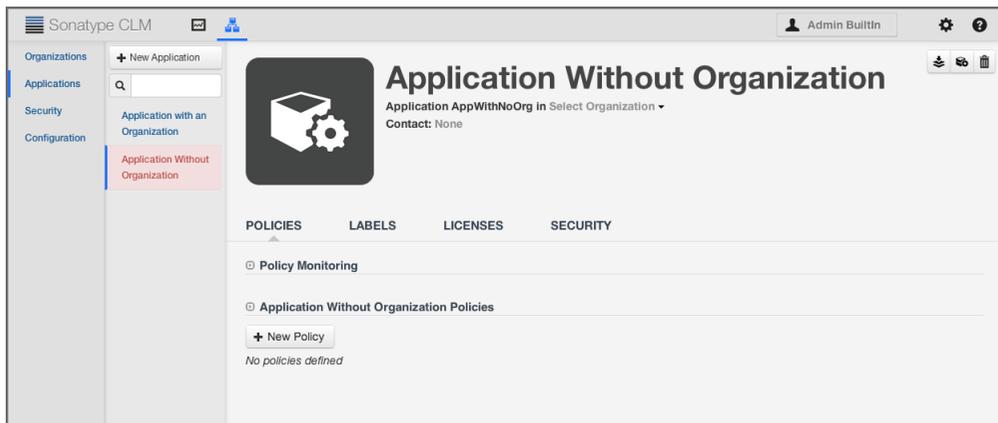


Figure 5.4: Application Without Organization v.1.8 UI

However, as of Sonatype CLM 1.9x and higher, this is no longer permitted, and will prevent the Sonatype CLM Server from starting. Prior to upgrading, make sure all applications have been assigned an organization. For more information on organization, please see the [Organization and Application Management section](#) of the Policy Management chapter.

Tip

If you have been a user of Sonatype CLM prior to version 1.8x, you may want to verify you've followed those previous upgrade instructions. This is especially true for those related to configuration (config.yml) changes.

5.4.1.3 Upgrading from Sonatype CLM 1.7x and 1.6x::

There are two critical changes that will affect any users upgrading from version 1.7x.

Config.yml Change

The introduction of the security administration features require that a specific line be added to your current config file, under the *loggers:* area.

```
"org.apache.shiro.web.filter.authc.BasicHttpAuthenticationFilter": ←  
    INFO
```

After adding, your config should look like this:

```
loggers:  
  "eu.medsea.mimeutil.MimeUtil2": INFO  
  "org.apache.http": INFO  
  "org.eclipse.jetty": INFO  
  "org.apache.shiro.web.filter.authc.BasicHttpAuthenticationFilter": ←  
    INFO
```

**Warning**

Failure to add this line to your **config.yml** file will result in credentials being published to the Sonatype CLM log file and is considered insecure.

Parent Organization Requirement

For those users of Sonatype CLM between, and including, 1.6x and 1.8x, applications were permitted to exist without a parent organization. Within the interface any applications without an organization were identified as follows:



Figure 5.5: Application Without Organization v.1.7 and Earlier UI

However, as of Sonatype CLM 1.9x and higher, this is no longer permitted, and will prevent the Sonatype CLM Server from starting. Prior to upgrading, make sure all applications have been assigned an organization. For more information on organization, please see the [Organization and Application Management section](#) of the Policy Management chapter.

5.4.1.4 Upgrading from Sonatype CLM 1.5x or Earlier

Prior to Sonatype CLM 1.5x there was no way to manage policy globally. This meant each application needed to have its own policy. As of Sonatype CLM 1.6x and the added functionality of organizations, policies could be created at the organization level, and then inherited by any applications attached to that organization.

At this time, there is not a direct path from this version of Sonatype CLM to the latest version. This does not exclude the ability to upgrade, but it will require more steps than is typical upgrades. To best assist with your upgrade, we recommend contacting Sonatype CLM Support support team by email support@sonatype.com or [file a request](#)

Note

In addition to this upgrade, they can also assist with migrating policies from an application to an organization.

Chapter 6

Sonatype CLM - Security Administration

Sonatype CLM uses the term *Security* to broadly refer to user accounts, passwords, and associated permissions. Regardless of the name, it shares a similarity to most systems in that open access isn't something that is permitted. This means, that unless you have a Sonatype CLM user account, you won't be able to view and make edits to policies, policy elements, and/or reports. Of course there are a variety of ways this can be accomplished, including simply using the default admin account that ships with the system.

In this chapter we'll walk you through logging into the server and creating additional users. We will also cover common user management protocols like Light Weight Directory Protocol (LDAP). Ultimately what we provide here is a set of best practices aimed at walking you through both an initial setup of Sonatype CLM security, as well as assist those that may just be looking to make modifications to one that already exists.

**Important**

It is important to note, that this chapter is aimed at system administrators, and isn't a guide for other users of Sonatype CLM. Most areas discussed here will require a user account that has been assigned to the System and/or CLM Administrator role.

6.1 User Management

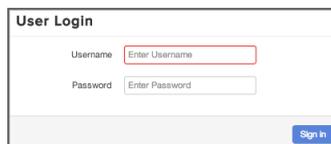
The Sonatype CLM Server requires a username and password before any policies or policy elements can be created, viewed, and edited. When a user is created specific to Sonatype CLM, we consider this user to be part of the *Sonatype Realm*. This is also considered independent of other connected realms such as [LDAP](#).

While Sonatype does suggest using a security protocol such as LDAP for managing users and permissions, the *Sonatype Realm* is still available for those who would like a lighter setup, where all users, groups and rights are stored directly in the Sonatype CLM server.. The function of user management in the Sonatype Realm focuses on managing all the elements of a user account. In this section we will cover:

- Logging In and Logging Out
- Managing the Admin Password / Account
- Viewing Notifications
- Creating, Editing and Deleting Users
 - First and Last Names
 - E-mail Addresses
- Changing Passwords

6.1.1 Logging in to Sonatype CLM

Any user that wants to access Sonatype CLM will need a username and a password. To perform the functions described throughout this section, you will need to use a user account that has been assigned to the [System Administrator](#) role.



The image shows a web form titled "User Login". It contains two input fields: "Username" with a placeholder text "Enter Username" and "Password" with a placeholder text "Enter Password". A blue button labeled "Sign In" is positioned at the bottom right of the form.

Figure 6.1: Login

By default the Sonatype CLM server has a preconfigured Admin account that has been assigned to all Administrator roles. Once you log in with this account for the first time, be sure to [change the admin password](#).

To logout, click on the *Log Out* link located in the upper right corner.

Note

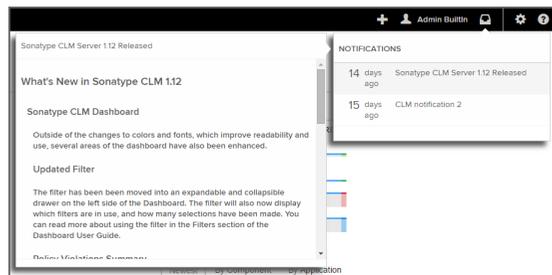
The server will timeout after 30 minutes of inactivity.

6.1.2 Viewing Notifications

Notifications provide a way for Sonatype to distribute important information about Sonatype CLM.

Unread notifications are indicated by a count, in blue, displayed over the notification icon . This icon is located in the Sonatype CLM header, next to the display of the logged-in user.

To view notifications, click on the notifications icon. This will open the Notification Panel, allowing you to click on any available notifications.



6.1.3 Changing the Admin Password

Sonatype CLM ships with a default admin account with a username `admin` and a password `admin123`. If you do nothing else related to security in Sonatype CLM, be sure to change this password. We'll cover this in Section [6.1.5](#) section below in more details, while we detail the process to change this default password now.

1. After logging in with the Admin account, click on the button with your username to the left of the *System Preferences* gear-shaped, icon. For the default administrator the username will show Admin Builtin.
2. A list of options will be displayed, click *Change Password*.
3. Enter the current password (admin123 for the default admin user), the new password, and then confirm the new password.
4. Click the *Change* button to save the new password.

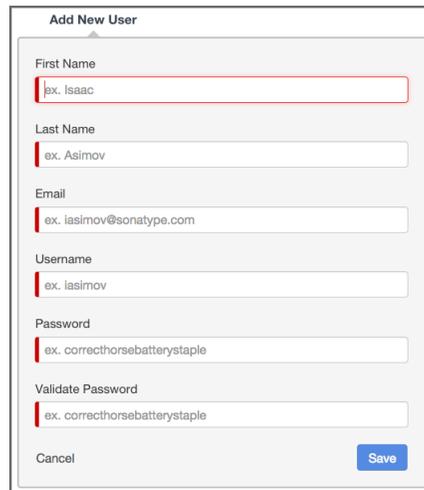
Note

Any user, including an admin, can change their password following the instructions above. However, only an admin can reset a user's password (discussed later in this chapter) without knowledge of the current password.

6.1.4 Creating a User

To create a new user in the Sonatype realm, follow the instructions below.

1. Log into the Sonatype CLM Server with a user that has been assigned to the System Administrator role.
 2. Click the *System Preferences* icon  located in the top right of the header.
 3. Choose *Users* from the drop down menu. The **Users** administration area will now be displayed.
 4. Click the *New User* button located at the top of the list of users.
 5. The **Add New User** form will now be displayed. Enter the following information:
 - a. First Name
 - b. Last Name
 - c. E-mail Address
 - d. Username
 - e. Password
 - f. Password Validation
 6. Click the *Save* button, to save the new user.
-



The image shows a web form titled "Add New User". It has several input fields, each with a red vertical bar on the left side. The fields are: "First Name" with the example text "ex. Isaac"; "Last Name" with "ex. Asimov"; "Email" with "ex. lasimov@sonatype.com"; "Username" with "ex. lasimov"; "Password" with "ex. correcthorsebatterystaple"; and "Validate Password" with "ex. correcthorsebatterystaple". At the bottom left is a "Cancel" link, and at the bottom right is a blue "Save" button.

Figure 6.2: Create User

6.1.5 Editing and Deleting User Information

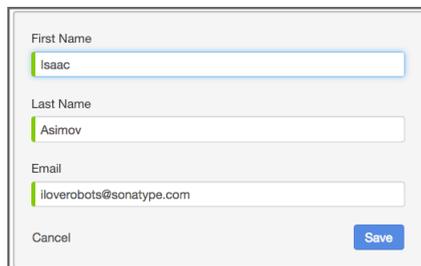
Editing user information is only available to an admin. The information that can be edited includes the first name, last name, email address, and password. To edit an existing user, follow these steps:

1. Log into the Sonatype CLM Server with a user that has been assigned to the System Administrator role.
2. Click the *System Preferences* icon  located in the top right of the CLM header.
3. Choose *Users* from the drop down menu. The **Users** administration area will now be displayed.
4. At least one user - the initial `admin` account - will be displayed. If you hover your pointer over the user record you will notice that there are three icons on the right.
 - a. The icon shaped like a pencil will allow you to edit user information (i.e. first name, last name, and e-mail address).
 - b. The icon shaped like a bag with an arrow back is for resetting a user's password. If you use this option a random, secure password will be generated and displayed in a dialog. Click the icon to the right of the field to copy it to clipboard.
 - c. The icon shaped like a trashcan will allow you to delete the user after you confirm the deletion in a dialog.

5. Make any desired changes, and unless you chose to delete the record, click the *Save* button.

Tip

With regard to changing a user's password, a user can always change their own password. However, this requires knowledge of the existing password. If you encounter a user that has forgotten their password, you can reset it for them.



The screenshot shows a web form for editing a user. It has three text input fields. The first is labeled 'First Name' and contains the text 'Isaac'. The second is labeled 'Last Name' and contains 'Asimov'. The third is labeled 'Email' and contains 'iloverobots@sonatype.com'. Below the fields, there is a 'Cancel' link on the left and a blue 'Save' button on the right.

Figure 6.3: Edit User

6.2 LDAP Integration

Light Weight Directory Protocol, also known more commonly as LDAP, provides both a protocol and a directory for storing user information. In some ways LDAP has become a ubiquitous part of most organizations' efforts to create a single sign on environment, as well as streamline user management for various applications. While we will cover some LDAP basics, the information provided here is limited and should not be considered a full reference.

Sonatype CLM supports a single LDAP realm, which simply means we support connection to a single LDAP server. This connection is configured via the Sonatype CLM Server. There are essentially two parts to integrating Sonatype CLM with LDAP:

- Configure the LDAP Server Connection
- Map LDAP User and Group Elements to Sonatype CLM

Our setup instructions provide an example using the Active directory format, and represent only the most basic approach. What we provide in this chapter assumes a simple authentication method for LDAP.

However, on a standard installation of Sonatype CLM, you would likely not want to use Simple Authentication as it sends the password in clear text over the network. Additionally, we have indicated a search base which corresponds to our organization's top-level domain components "dc=sonatype,dc=com". The structure can vary greatly based on your own LDAP server configuration.

Note

Once the LDAP server is configured and user attributes have been mapped, both LDAP users and users created in the Sonatype CLM Realm will be able to login into Sonatype CLM.

6.2.1 Configuring the LDAP Server Connection

The first step to establish the LDAP connection is to configure Sonatype CLM to point to your LDAP server. Those instructions are pretty straightforward as long as you have the necessary information. For this example, let's assume we have been provided the following information:

Server Name	Test LDAP Server
Protocol	LDAP
Hostname	wind-son04
Port	389
Search Base	dc=sonatype,dc=com
Authentication Type	Simple
Username	testuser
Password	tester

Note

The information provide will not allow you to access an LDAP server, and is provided just for demonstration purposes. In addition, this is only a representation of a simple connection. For an explanation of all available parameters, please see the next section.

Now, access the Sonatype CLM Server:

1. Log into the Sonatype CLM Server (*by default this is available at <http://localhost:8070>*) with a user account assigned to the System Administrator role.
 2. Click the system preferences icon  located in the top right of the CLM Header/Screen (resembles a cog/gear).
-

3. Choose LDAP from the available option. The *LDAP Administration* area will be displayed.
4. Enter the various parameters, and then use the **Test Connection** button to ensure a connection can be made to the configured LDAP Server.
5. Click the **Save** button when finished.

Using the information from the table above, our configuration should look something like this:



The screenshot displays the 'LDAP Server' configuration window, divided into two tabs: 'CONNECTION' and 'USER & GROUP SETTINGS'. The 'CONNECTION' tab is active, showing the following fields:

- Protocol:** LDAP (selected in a dropdown menu)
- Hostname:** i-robot-01
- Port:** 389
- Search Base:** dc=win,dc=steelcaves,dc=local
- Authentication Method:** SIMPLE (selected in a dropdown menu)
- SASL Realm:** (empty text field)
- Username:** cn=testuser1,cn=users,dc=win,dc=steelcaves,dc=local
- Password:** (masked with dots)
- Timeouts:**
 - Connection:** 30
 - Retry Delay:** 300

At the bottom of the form, there are three buttons: 'Cancel', 'Test Connection', and 'Save'.

Figure 6.4: Sample LDAP Server Configuration

Note

If at any point you wish to reset the form, click the reset button and any value that have been entered will be removed.

6.2.2 LDAP Configuration Parameters

As mentioned, the example above is a basic setup. Given this, there are a number of parameters not utilized. This section provides descriptions for all available parameters that can be configured in the Connection section of the LDAP Configuration area on the Sonatype CLM Server. When applicable, required fields have been noted.

General

Protocol

Valid values in this drop-down are LDAP and LDAPS, which correspond to the Lightweight Directory Access Protocol and the Lightweight Directory Access Protocol over SSL.

Hostname

The hostname or IP address of the LDAP.

Port

The port on which the LDAP server is listening. Port 389 is the default port for the LDAP protocol and port 636 is the default port for the LDAPS.

Search Base

The search base is the Distinguished Name (DN) to be appended to the LDAP query. The search base usually corresponds to the domain name of an organization. For example, the search base on the Sonatype LDAP server could be "dc=sonatype,dc=com".

Authentication

Method

Sonatype CLM provides four distinct authentication methods to be used when connecting to the LDAP Server:

- Simple Authentication - Simple authentication is not recommended for production deployments not using the secure LDAPS protocol as it sends a clear-text password over the network.
 - Anonymous Authentication - Used when Sonatype CLM only needs read-only access to non-protected entries and attributes when binding to the LDAP.
 - Digest-MD5 - This is an improvement on the CRAM-MD5 authentication method. For more information, see <http://www.ietf.org/rfc/rfc2831.txt>.
-

- **CRAM-MD5** - The Challenge-Response Authentication Method (CRAM) based on the HMAC-MD5 MAC algorithm. In this authentication method, the server sends a challenge string to the client, the client responds with a username followed by a Hex digest which the server compares to an expected value. For more information, see RFC 2195. For a full discussion of LDAP authentication approaches, see <http://www.ietf.org/rfc/rfc2829.txt> and <http://www.ietf.org/rfc/rfc2251.txt>.

SASL Realm

The Simple Authentication and Security Layer (SASL) Realm to connect with. The SASL Realm is only available if the authentication method is Digest-MD5.

Username

Username of an LDAP User to connect (or bind) with. This is a Distinguished Name of a user who has read access to all users and groups.

Password

Password for an Administrative LDAP User.

Timeouts**Connection**

The number of seconds Sonatype CLM should try and connect to the configured server before returning an error.

Retry Delay

The number of seconds Sonatype CLM should wait before attempting to connect to the configured server again (after an error).

6.2.3 Mapping LDAP Users to Sonatype CLM

Once the LDAP Server has been configured, you can map information attributes of an LDAP user to match those of Sonatype CLM. Similar to configuring the LDAP Server, this will require that you have information related to the location of various user attributes. Here is a sample set of data, that you would likely see:

Base DN	cn=users
Object Class	user
Username Attribute	sAMAccountName
Real Name Attribute	cn
Email Attribute	mail

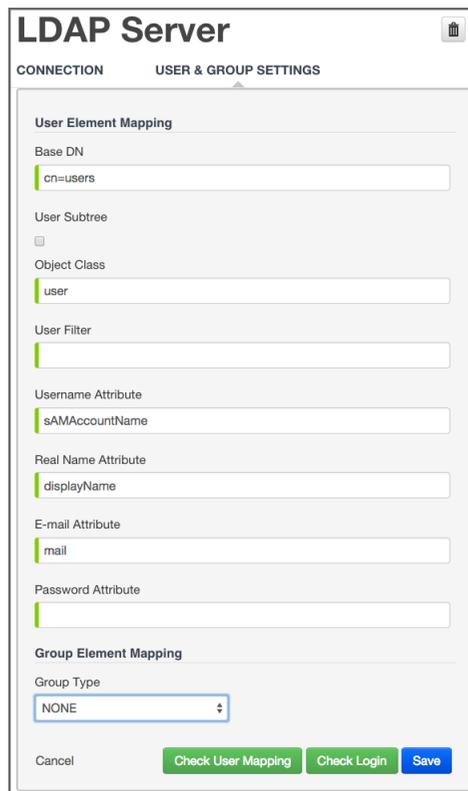
Once you have gathered this information, access the Sonatype CLM Server LDAP Configuration:

1. Log into the Sonatype CLM Server (*by default this is available at <http://localhost:8070>*) with a user account assigned to the System Administrator role.
2. Click the system preferences icon  located in the top right of the CLM Header/Screen (resembles a cog/gear).
3. Choose LDAP from the available option. The *LDAP Administration* area will be displayed.
4. Click on the Second Tab, just below the Server Name, *User and Group Settings*.
5. Enter the various settings, using the Test Mapping button to ensure the correct information has been mapped.
6. Click the **Save** button when finished.

Note

If at any point you wish to reset the form, click the reset button; Any values that have been entered will be removed.

Using the information from the table above, our configuration would look like this:



The screenshot shows the 'LDAP Server' configuration window, specifically the 'USER & GROUP SETTINGS' tab. The 'User Element Mapping' section contains the following fields:

- Base DN:
- User Subtree:
- Object Class:
- User Filter:
- Username Attribute:
- Real Name Attribute:
- E-mail Attribute:
- Password Attribute:

The 'Group Element Mapping' section contains:

- Group Type:

At the bottom, there are three buttons: 'Cancel', 'Check User Mapping', and 'Check Login', followed by a 'Save' button.

Figure 6.5: User Mapping

6.2.4 LDAP User Parameters

As mentioned, the example above is a basic setup. Specifically, we do not turn on the User Subtree option or utilize the User Filter. Descriptions for those fields, as well as all available parameters for mapping LDAP User Attributes to Sonatype CLM have been provided below. When applicable, required fields have been noted.

Base DN *(required)*

Corresponds to the Base DN (Distinguished Name) containing user entries. This DN is going to be relative to the Search Base. For example, if your users are all contained in "cn=users,dc=sonatype,dc=com" and you specified a Search Base of "dc=sonatype,dc=com" you would use a value of "cn=users"

User Subtree

Enable this parameter if there is a tree below the Base DN which can contain user entries. For example, if all users are in "cn=users" this field should not be toggled. However, if users can appear in organizational units below "cn=users", such as "ou=development,cn=users,dc=sonatype,dc=com" this field should be toggled

Object Class (required)

The object class defines what attributes are expected for a given object. What is entered here must be the object class for the Username Attribute, Real Name Attribute, Email Attribute, and the Password Attribute.

User Filter

The user filter allows you to isolate a specific set of users under the Base DN.

Username Attribute (required)

This is the attribute of the Object class which supplies the username.

Real Name Attribute (required)

This is the attribute of the Object class which supplies the real name of the user.

E-Mail Attribute (required)

This is the attribute of the Object class which supplies the email address of the user.

Password Attribute

This is the attribute of the Object class which supplies the User Password. By default it is not toggled, which means authentication will occur as a bind to the LDAP server. Otherwise this is the attribute of the Object class which supplies the password of the user.

6.2.5 Mapping LDAP Groups to Sonatype CLM

In most LDAP implementations users are collected into various groups. This allows for better organization of a larger numbers of users, as well as provides a mechanism to isolate particular groups for specific permissions and integration into other systems such as Sonatype CLM. If LDAP groups are not mapped, Sonatype CLM will pull in all users from the Base DN. This isn't so much an an issue for a small number of users. However, for larger ones it may be a concern and might grant unintended access.

As we've done with the other configuration areas, let's look at a sample set of data. In example below we'll be configuring a static LDAP group.

Group Type	Static
Base DN	ou=groups
Object Class	group
Group ID Attribute	sAMAccountName
Group Member Attribute	member

Group Member Format	
--------------------------------	--

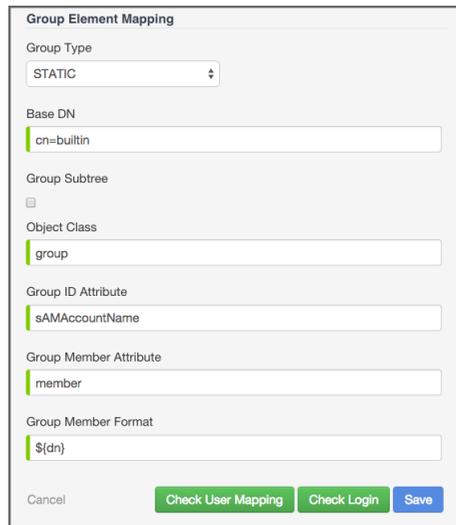
Once you have gathered this information, access the Sonatype CLM Server LDAP Configuration:

1. Log into the Sonatype CLM Server (*by default this is available at <http://localhost:8070>*) with a user account assigned to the System Administrator role.
2. Click the system preferences icon  located in the top right of the CLM Header/Screen (resembles a cog/gear).
3. Choose LDAP from the available option. The *LDAP Administration* area will be displayed.
4. Click on the Second Tab, just below the Server Name, *User and Group Settings*.
5. Just below the User Element mapping, you will see Group Element Mapping. The Group Type field will be set to *none*. Change this to *static* or *dynamic* based on the parameter descriptions below.
6. Enter the various settings.
7. Click the **Save** button when finished.

Note

If at any point you wish to reset the form, click the reset button; Any values that have been entered will be removed.

Using the information from the table above our configuration would look like this:



The screenshot shows a 'Group Element Mapping' dialog box with the following fields and values:

- Group Type: STATIC
- Base DN: cn=bulletin
- Group Subtree:
- Object Class: group
- Group ID Attribute: sAMAccountName
- Group Member Attribute: member
- Group Member Format: \${dn}

At the bottom, there are three buttons: 'Cancel', 'Check User Mapping', and 'Check Login', followed by a 'Save' button.

Figure 6.6: Group Mapping

6.2.6 LDAP Group Parameters

Groups are generally one of two types in LDAP systems - static or dynamic. A static group contains a list of users. A dynamic group is where the user contains a list of groups the user belongs to. In LDAP a static group would be captured in an entry with an Object class `groupOfUniqueNames` which contains one or more `uniqueMember` attributes. In a dynamic group configuration, each user entry in LDAP contains an attribute which lists group membership. This means the available parameters will be different based on whether you've chosen static or dynamic.

Tip

Static groups are preferred over dynamic ones, and will generally perform better if you have a large number of LDAP users.

6.2.6.1 Static Groups

Static groups are configured with the following parameters:

Base DN (required)

This field is similar to the Base DN field described for User Element Mapping. If your groups were defined under "ou=groups,dc=sonatype,dc=com", this field would have a value of "ou=groups"

Group Subtree

This field is similar to the User Subtree field described for User Element Mapping. If all groups are defined under the entry defined in Base DN, this field should not be selected. If a group can be defined in a tree of organizational units under the Base DN, this field should be selected.

Object Class (required)

This is a standard object class defined as a collection of references to unique entries in an LDAP directory, and can be used to associate user entries with a group.

Group ID Attribute (required)

This field specifies the attribute of the Object class that defines the Group ID.

Group Member Attribute (required)

This field specifies the attribute of the Object class that defines a member of a group.

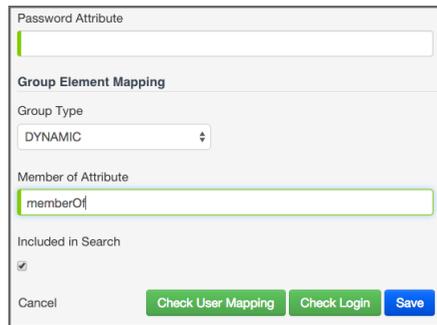
Group Member Format (required)

This field captures the format of the Group Member Attribute, and it is used by Sonatype CLM to extract a username from this attribute. For example, if the Group Member Attribute has the format `uid=brian,ou=users,dc=sonatype,dc=com`, then the Group Member Format would be `uid=${username},ou=users,dc=sonatype,dc=com`. If the Group Member Attribute had the format "brian", then the Group Member Format would be `${username}`.

6.2.6.2 Dynamic Groups

If your installation does not use Static Groups, you can configure Sonatype CLM LDAP integration to refer to an attribute on the User entry to derive group membership. To do this, select Dynamic Groups in the Group Type field in Group Element Mapping.

Dynamic groups are configured via the Member of Attribute parameter. Sonatype CLM will inspect this attribute of the user entry to get a list of groups that the user is a member of. In this configuration, a user entry would have an attribute such as *memberOf* which would contain the name of a group.



The screenshot shows a dialog box titled "Dynamic Group Options". It contains the following fields and controls:

- Password Attribute:** An empty text input field.
- Group Element Mapping:**
 - Group Type:** A dropdown menu currently set to "DYNAMIC".
 - Member of Attribute:** A text input field containing the value "memberOf".
- Included in Search:** A checked checkbox.
- Buttons:** "Cancel", "Check User Mapping", "Check Login", and "Save".

Figure 6.7: Dynamic Group Options

Tip

Depending on the size of your enterprise, LDAP search could be slow. If you find this is the case, uncheck the option to "Include in Search". This will [exclude groups from search results](#) when assigning users to roles. Searching for users will remain unaffected.

6.2.7 Verifying LDAP Configuration

It's easy to make a typo, or even have entered the wrong information when mapping LDAP users or groups. There are a number of tools provided within the LDAP configuration area to assist in making sure everything has been mapped correctly. Each of these is discussed below.

6.2.7.1 Test Connection

Testing the LDAP connection is the first step. If you can't connect to your LDAP server, user and group mapping will fail as well.

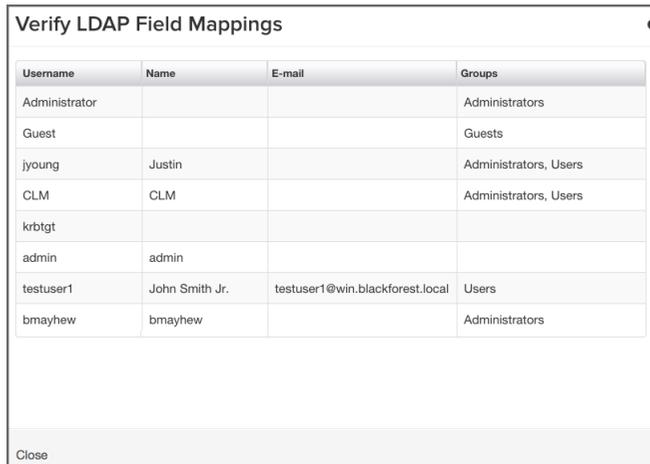


The screenshot shows a dialog box titled "Test Connection" with a green header bar containing the text "Success!". Below the header bar are three buttons: "Cancel", "Test Connection", and "Save".

Figure 6.8: Testing LDAP Server

6.2.7.2 Check User and Group Mapping

Making sure that usernames, real names, email addresses, and groups have been mapped correctly can be verified with the Check User Mapping.

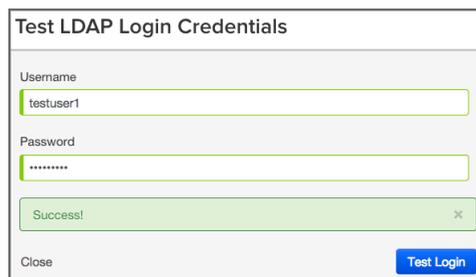


Username	Name	E-mail	Groups
Administrator			Administrators
Guest			Guests
jyoung	Justin		Administrators, Users
CLM	CLM		Administrators, Users
krbtgt			
admin	admin		
testuser1	John Smith Jr.	testuser1@win.blackforest.local	Users
bmayhew	bmayhew		Administrators

Figure 6.9: Checking User Mapping

6.2.7.3 Check Login

As a final test to ensure users can log in, the Check Login allows you to enter a user name and password, and ensure ensure that this can be authenticated with the LDAP server.



Test LDAP Login Credentials

Username
testuser1

Password

Success! x

Close Test Login

Figure 6.10: Checking User Login

6.3 Role Management

Roles provide a set of permissions which grant various levels of access and control over the Sonatype CLM Server, as well as the connected suite of tools. In the following sections we'll describe how to access role definitions, view the associated permissions, create custom roles, and assign users to a role.

6.3.1 Viewing Role and Permission Descriptions

To view role descriptions:

1. Log into the Sonatype CLM Server.
2. Next, click on the *System Preferences* (the gear icon located on the upper right corner of the CLM Server toolbar), and then the *Roles* menu item.
3. Now, a list of available roles will be displayed. Click on a specific role to see the associated permissions.

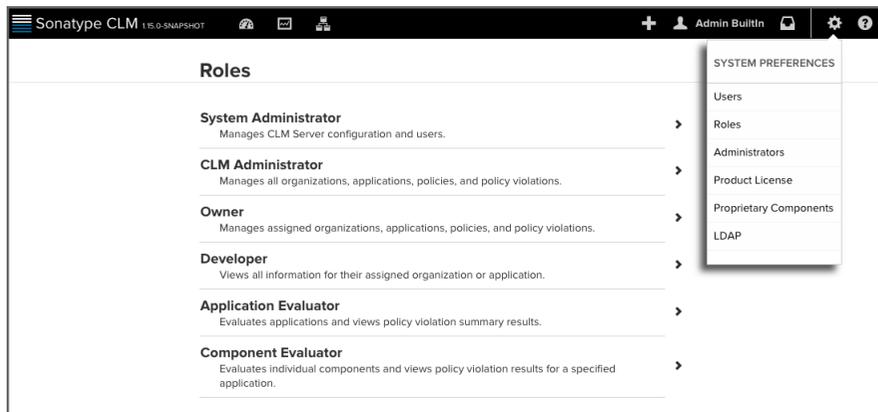


Figure 6.11: Role and Permission Descriptions

**Warning**

You will need to be assigned to one of the administrator roles in order to see the information above. It is important to note that if you are using the built-in Admin user account, it has been assigned to all administrator roles, and we highly recommend [changing the password](#).

6.3.2 Assigning Users to Roles

Users can be assigned to two types of roles: Administrative and/or Organizational. To assign a user to a role...

1. First, determine which type of role you want to assign a user to.
 - a. **Administrators** - click on the *System Preferences* icon and then *Administrators*.
 - b. **Organization/Application** - Select an organization or application, and then click on the *Security* tab.
2. Next, for either type of role you selected, click the *Edit* icon (it resembles a pencil).
3. A search widget will be displayed. In the search field, enter as much of the user's complete name as possible, followed by a trailing wildcard (e.g. Isaac A*), and then click the *Search* button.

Note

Practice caution as use of leading wildcards can greatly impact user search times.

4. Once you see the user you wish to add in the *Available* column, click the *Plus* icon to move them to the *Applied* column. To remove users from a role, follow the same process above, just click the *Minus* icon to move the user from the *Applied* column to the *Available* column. Click the *Save* button to save your changes.

Tip

You may notice that below each user, there is additional information. Most often this is the email. However, to the right of the email you will see the realm (e.g. LDAP). Use this to ensure you add the appropriate account.

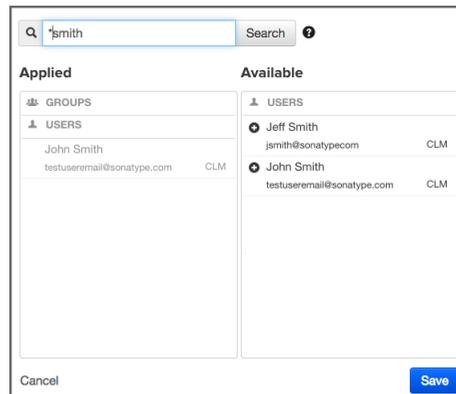


Figure 6.12: Assigning Users to Roles

6.3.3 Creating Custom Roles

The Sonatype CLM Server ships with a set of built-in roles. While these roles cannot be modified, you can create your own custom roles. To perform this action, you will need a user that has the permission to *Edit Custom Roles*, e.g., the default admin account and the CLM Administrator role that ship with the Sonatype CLM Server have this permission.

To create a custom role:

1. First, click on the *System Preferences* icon, and then *Roles*.
2. Next, click on the *Create Role* button.
3. The **New Role** form will be displayed. Enter a name and description for the role, then click the *Can/Cannot* slider to enable/disable a permission.
4. Click the *Save* button.

Roles / Editor

New Role

Role Details

Role Name

Role Description

Permissions

Administrator

<input type="button" value="Cannot"/>	View	All Roles
<input type="button" value="Cannot"/>	Edit	Proprietary Components

CLM

<input type="button" value="Cannot"/>	Claim	Components
<input type="button" value="Cannot"/>	Edit	CLM elements
<input type="button" value="Can"/>	View	CLM elements
<input type="button" value="Can"/>	Evaluate	Applications
<input type="button" value="Can"/>	Evaluate	Individual components

Cancel

6.3.4 Excluding Groups from Search Results

Assigning a group to a role utilizes elements that are configured via the [LDAP System Preferences area](#). If you go with the default options, groups will be included with the search results. That is, when you enter something into the *Find User* field, both groups and single users will be returned.

However, because the size of LDAP implementation can vary, you may want to consider not including groups with your search results. This option can be adjusted when using [Dynamic Groups settings](#).

Making this change will then allow you to manually enter group names. However, when entering groups this way, no search or validation will be performed.

The screenshot displays the 'Groups' configuration interface. At the top, there are tabs for 'Users' and 'Groups'. Below the tabs is a 'Group Name' input field followed by an '+ Add Group' button. The main section is titled 'Applied' and contains a tree view. Under 'GROUPS', 'TestGroup' is selected with a radio button, and 'Administrators' is listed below it. Under 'USERS', 'John Smith' is listed. The interface also includes a 'Cancel' button at the bottom left and a blue 'Save' button at the bottom right.

Figure 6.13: Assigning Groups Manual Search

Chapter 7

Sonatype CLM - Policy Management

When we talk about policy within the paradigm of Sonatype CLM, we refer to it as a way to identify and reduce risk through a concise set of rules for component usage. These rules can be used to assist at every step of the component and development lifecycle, and can be customized for specific applications and organizations. In general, policy, within the context of Sonatype CLM, is a broad term used to encapsulate:

- Conditions
- Actions

In some ways rules as a description is a bit generic, so let's dig a bit deeper, and look at another concept you are likely familiar with, an `If/Then` statement.

In fact, that's one of the easiest ways to break down the various elements of a policy. That is, a policy simply says that if something happens, then perform a certain action. If a component meets a set of criteria, then take a certain action, or in some cases no action at all.

If it's still a bit fuzzy, an example will probably help. Let's say we have a known rule in our development organization that says if a component used in an application has a security vulnerability, the application can not be released. To do this, we tell our development team to review components before release and if a component has a security issue, we don't promote the release. Congratulations, you have formed, at least in the aether, your first policy.

Now, let's take a slightly closer look, and define the basic policy anatomy. There are actually three key parts to a policy:

Conditions

conditions are the `if` part of the `if-then` statements.

Constraints

a constraint is really just a way to organize multiple conditions (if-then statements). Our example only had one so far. Let's say we decided we wanted to add that if a security issue is found and it has a CVSS of 2 or lower, only a warning should occur, but the release should not be prohibited.

Actions

actions are simply the `then` part of the if-then statement. Basically, what you want to have happen.

The above does a good job of telling us what makes up a policy in Sonatype CLM, but you are likely thinking, not all policies should be the same, I need a way to demonstrate which policies are the most important. We thought that too, and that is why in Sonatype CLM, all policies are assigned a threat level ranging from zero to ten (0-10). This score is completely subjective and will be unique in your organization.

OK, so now that we've opened up our concept of policy a bit, exposing the inner workings so to speak, the next question you should have is, "Where do we create policies?"

Policy is managed within a set of containers, organizations and applications, within Sonatype CLM. We'll discuss those in the next section.

7.1 Organization and Application Management

You've likely heard that Sonatype CLM provides you with information about the components inside your applications. In addition to that information, you will see whether or not that component meets the rules for component usage that your organization has established - your policy. In order to provide that information however, there needs to be a link between your application and Sonatype CLM. But, how do we create that link, and where do we start?

Well, let's do a little introspection, and take a look at the idea of organizational structure.

Note

For information on necessary roles and permissions, please see the [Role Management](#) section of the Sonatype CLM Security Administration Chapter.

7.1.1 Organizational Structure

When you launch Sonatype CLM for the first time, even after setting up and configuring your security parameters, there will be little to no information, a blank slate if you will.

Now, you could go off and simply start creating organizations and applications, as it's a fairly simple process. However, it would be wiser to think about how your particular business organizes applications. For many teams this follows a "command and control" structure, or rather one where various business units are responsible for specific applications. For others, applications create more logical categories, such as internal, or perhaps, commercial units, each having sets of applications below them.

In both cases these are units which simply contain applications, and there is some correlation between each application, even if it is only surface level. This idea of containers and correlation is the exact principle behind organizations.

Organizations, when looking simply at their most basic function, serve as a container for applications. While we cover how organizations manage policy and the other policy elements in just a moment, it's important to think about how you will set up your organizations before you begin creating them in Sonatype CLM.

Once you've done that though, you are ready to create an organization.

7.1.2 Creating an Organization

Organizations are created via the Sonatype CLM Server. Make sure you have proper access, which includes being a member of the CLM Administrator role.

There are two main ways to create an organization:

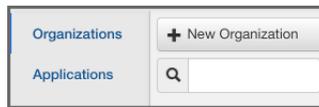


Figure 7.1: Using New Organization button



Figure 7.2: Using Global Create Button

The essential difference between the two options lies in Global Create button, which simply provides access to create an organization from anywhere within the Sonatype CLM Server.

Regardless of the option you choose, to create the organization, you only need to enter a name, and then click the *Save* button.

OPTIONAL ROBOT: As an option, you can add an icon for your organization, but this is not required. The image should be sized to 160 x 160 pixels and use the PNG format. Images with different sizes will be scaled. Alternatively you can press *Want a robot* to use a robot image. Each time you click on the link, a new robot image will appear.

7.1.3 Creating an Application

Earlier, we talked about a link between applications being developed, and the policy (or policies) that will be governing the components used in those applications. That link is provided by creating an application record within Sonatype CLM.

There are two main ways to create an application:

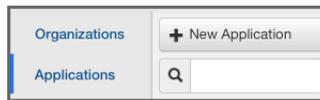


Figure 7.3: Using New Application button

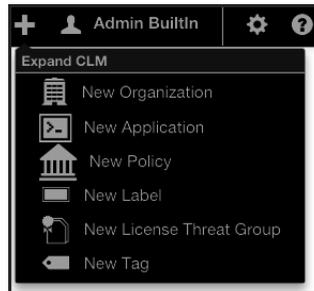


Figure 7.4: Using Global Create Button

The essential difference between the two options lies in Global Create button, which simply provides access to create an application from anywhere within the Sonatype CLM Server.

Regardless of the option you choose, the information necessary to create an application is the same. Each application has three essential parts, which have been described below. Descriptions for optional items have been included as well.

Application Name (*required*)

This can be anything you want it to be, but it should be something people recognize. For example, Employee Intranet Application for Android, or International Bank Transfer Application. It is, quite simply, just a name, and it should be one that your users can identify with easily, as this is the name they will see in the various tools that connect to Sonatype CLM.

Application ID (*required*)

An Application ID, or App ID, is a unique identifier that you define for the application. In many ways, it's like a national identifier for the application. Most users will never see the application ID. However, it is used in a number of manual locations, including the various APIs that Sonatype CLM provides.

Organization (*required*)

Applications can share the same organization, and depending on the organization that you choose this determines a number of things such as which policies the application will be evaluated against.

We'll discuss this more in the next section, for now, just treat this as a logical container helping to group your applications.

Note

Once an organization has been selected for an application, it can not be changed.

Contact (optional)

The contact is the person that is responsible for the application, or at the very least, should be contacted if there is an issue. It will be displayed in the reporting area of Sonatype CLM, as well as the PDF version of the report.

Icon (optional)

You can add an icon for your application, to help make it more easily identifiable. The image should be sized to 160 x 160 pixels and use the PNG format. Images with different sizes will be scaled. Alternatively you can press *Want a robot* to use a robot image. Each time you click on the link, a new robot image will appear.

7.1.4 Organization, Application, and Inheritance

So we understand the difference between organizations and applications, but how should this affect how we manage policy?

In fact the concept of policy may still be confusing. For now though, let's just think about it as a set of rules for components you can or can't use in your applications. Given this, each application is different, why not create policy (rules) for each individual application?

This is actually a common question when it comes to creating your policies for the first time. Our inclination tends to be to match a policy to an application. That is because we can sometimes think of applications as being very unique, and for that reason they will each have their own policies, different from other applications. This might be true. However, inheritance from an organization to an application plays a big role in making policy management much easier. Let's look at how the concept of inheritance works.

Because an organization can have multiple applications attached, and those applications inherit policies as well as labels, and license threat groups, creating a policy at the organization level allows us to manage a single policy across hundreds of applications. With this inheritance, you can make one modification and have that change affect all, or at least large numbers of, applications.

Now, let's put this in contrast with creating a policy at the application level, which seems similar, but the lack of inheritance from one application to another changes things up.

7.1.5 Avoiding Policy Micromanagement

In the case of organization level policy (rules), which appears across many different applications, the application level policy is meant for precise scenarios where the policy is only intended for a single application.

Doing this takes into account something specific we want to identify or keep out of a single application, but not others. The more of these application level policies that are created though, means the more micromanagement, and in turn, opportunity for error, will occur. So, keeping them at a minimum, and only for those unique scenarios is ideal.

This is likely better conveyed in an example. Imagine two applications with 4 policies each. Two policies for these applications are identical. If you have this setup with two separate applications, any change to the identical policies has to be done once for each application. However if we move these policies to the organization that both applications belong to, we only have to change one. Now imagine a similar scenario with a larger number of shared policies as well as applications. Without organizations to inherit from this would become unmanageable quickly.

Alright, now that you have a basic understanding of organizations and applications, let's take a look at the basics behind developing policy.

7.2 Policy Development

Sonatype CLM uses the term **policy** to broadly refer to the set of policies and policy elements (e.g. labels and license threat groups) used to ensure components in an application meet a specific set of standard. In the past, we colloquially compared these to *rules*.

The process of creating this set of rules based on specific factors is considered policy development. Combining this with the ongoing refinement and adjustment is the broader category of policy management. No matter what it is called though, the end result should always be actionable results that are representative of your organization's risk tolerance. Put a bit more simply, Sonatype CLM policy provides a means to organize risk data.

Before we expand on risk, let's dig a little deeper, and really take a look at what we mean when we talk about policy, expanding everything that goes into its development.

7.2.1 Advanced Anatomy of a Policy

If you have taken a look at any of Sonatype CLM documentation or poked around the imported policies in the Sonatype CLM server user interface, you may have already seen what we refer to as basic anatomy of a policy - all the pieces that go together and define your policy.

Security-High
✎ 🗑

9

Application Matching

Which applications should this policy apply to?

- All Applications in My Organization
- Applications with one or more of these tags None selected

Constraints

CVSS >=7 and <10
+

Security Vulnerability Severity >= 7
- +

Security Vulnerability Severity < 10
- +

Security Vulnerability Status is not Not Applicable
- +

All of the above conditions will trigger this constraint.

Actions

Stage	Enforcement Points		Notifications
	Warn	Fail	
Develop			✉
Build			✉
Stage Release			✉
Release		!	✉
Operate			✉

Monitoring Notifications

If you have enabled monitoring, who should be notified?

MySecurityTeam@mycompany.com

Cancel
Save

Figure 7.5: Editing a Policy and its Attributes

So, branching beyond the simple concept of *If/Then* statements, let's break policy down into each part you can interact with keeping an eye on the editing screen for a policy displayed in Figure 7.5.

Policy Name

This name will be displayed on the *Policy* tab in the application composition report. Others will see

this regularly, so it should be unique, clear, and concise.

Threat Level

A number, 10 - 0, that is color coded (red, orange, yellow, dark blue, and light blue), and represents the perceived severity if this policy is violated. The number will also be used to create the order in which policy violations are displayed.

Constraints

Each policy must have at least one constraint. When a constraint is fulfilled, a policy violation occurs. A constraint itself consists of the **Constraint Name** and at least one condition. Make sure it clearly identifies the conditions that you have added for the Constraint.

Conditions

A condition is considered the *if* part of an if-then statement. There are a wide range of conditions possible, that have their own set of values you can choose from.

Any/All

Any/all is required when there are multiple conditions. This tells the policy whether all of the listed conditions, or simply any of them, must be met in order to have a policy violation.

Actions

The *then* part of an if-then statement. The action chosen here will be taken when the policy constraint and its associated conditions have been met.

Stage

These stages represent the enforcement point. They are *Develop*, *Build*, *Stage Release*, *Release* and *Operate*.

Warn and Fail

Each enforcement point for each stage can be configured to cause a *Warn(ing)* or a *Fail(ure)*.

Custom

This action allows you to select an email, or emails, to send notification to when new policy violations have occurred.

Now that we understand the different attributes of an individual policy, you're likely eager to create your own policy. We'll tackle that next in Section 7.3, but first let's take a look at some important items to consider when developing our policy.

Note

When viewing policy violations counts, please keep in mind that despite the number of constraints fulfilled, only one policy violation is counted.

7.2.2 Risk and Organizational Intent

To establish an understanding of risk within the context of CLM, we need to identify the various avenues of risk a component can have. The most common are security, licensing and architectural considerations.

Of course, we shouldn't limit our thinking to these three alone, and in the long term you will define those specific to your organization. However, they will serve us well to get started.

When creating CLM policies, we need to consider what is the risk we want reported and how we want it reported. Take a look at these very simple example policies, one each for licensing, security, and architecture.

Licensing

- Don't allow distributed code to have GPL
- Only allow GPL that has a Commercial license

Security

- Don't allow components with a CVSS score > 7

Architecture

- Don't allow components that are older than 5 years
- Don't allow Struts version 2.3.15.1

The above policies represent an organizational intent to not allow GPL highly insecure components. It further qualifies that old components and a specific version of struts are not to be used. In this way, our three policies are actually working together to form an overall policy approach.

As you move on to create your own policies, it becomes very important to think about how policies can build upon each other. In many ways this is a holistic approach, something that rules simply can't do.

7.3 Policy Creation

If you haven't done so, it's also a good idea to write out the policies you want to create. While you will find the actual creation process is pretty easy, the more thought you put into your policies and their structure the better they will be.

Also, it is important to be sure you have everything in place before you begin creating your own custom policies:

- Organizations and applications are set up as desired in the Sonatype CLM server.
- Users have been assigned to these organizations and applications with the proper roles to fulfill your security requirements.
- You have determined the risks that apply to your organization and/or applications.

We've decided to break policy creation into six total steps. Each step deals with a specific area of a policy and will be described in detail. Before you go to the first step, you should know there are two key ways to create a policy.

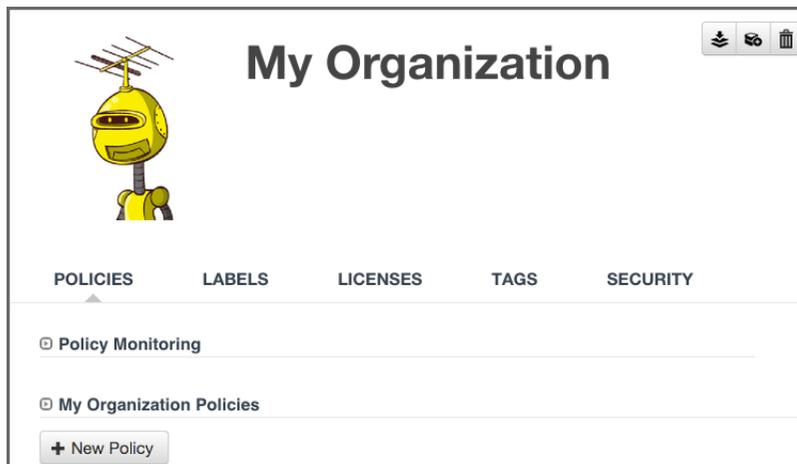


Figure 7.6: Using New Policy Button



Figure 7.7: Using Global Create Button

Note

Our instructions follow the process for creating a policy for an organization. This is where most of your policies will be created. However, if you need to create an application-specific policy, just substitute application for organization.

There is really no difference here, as both require that you have the organization or application open at the time of creation. The one advantage with using the Global Create button is that you can create no matter which tab of the currently selected organization or application you are in.

7.3.1 Step 1: Understand the Policy Intent

So, let's build on the third item we mentioned in the introduction, and first think about the intent of the policy we are going to create. Here are some questions to consider:

- Is this policy for all applications, or should it only be matched against certain ones?
- What rules do we want to create, or in other words, what things do we not want in our applications, or this specific set of applications?
- How do we want to react when violations occur?

These questions are really just the start, but in short you should ask what applications, what causes a violation, and what action should be taken. For our example policy, we want to focus on architecture aspects, with key areas being *Age* and *Popularity* of our components.

7.3.2 Step 2: Decide on a Descriptive Policy Name

Since architecture is our focus we are going to use it as part of the name for our policy. In addition we are going to add *Age* and *Popularity* to the name to be more specific. This will help us later when we start creating additional architecture related policies. Lets go ahead and create this policy:

1. Log into the Sonatype CLM Server (*by default this is available at <http://localhost:8070>*) using a user account with Owner-level permissions for the organization (a member of the Organization's Owner Group).
 2. Next, click the *Manage Applications and Organizations* icon  to access the **Application and Organization Management** area.
 3. In the menu on the left, click on *Organizations*, and then click the organization you want to add a policy to.
 4. To create a new policy, click the *New Policy* button. You should now see the policy editing screen (as displayed in Figure 7.8).
 5. The policy name input field should have focus and contain *Enter Policy Name*.
 6. Enter the new policy name by typing *Architecture Age and Popularity*
-

Architecture Age and Popularity | 5

Application Matching
Which applications should this policy apply to?
 All Applications in My Organization
 Applications with one or more of these tags None selected

Constraints
 [Unnamed Constraint] +

Actions

Stage	Enforcement Points		Notifications
	Warn	Fail	
Develop			<input checked="" type="checkbox"/>
Build			<input checked="" type="checkbox"/>
Stage Release			<input checked="" type="checkbox"/>
Release			<input checked="" type="checkbox"/>
Operate			<input checked="" type="checkbox"/>

Monitoring Notifications
If you have enabled monitoring, who should be notified?

Cancel Save

Figure 7.8: Naming the Policy

7.3.3 Step 3: Choose an Appropriate Threat Level

Threat level is one of the easiest concepts we’ll cover in this chapter, yet it has the greatest chance to cause huge problems. At it’s core, this is simply a value for severity your business associates with a given policy. These values range from 1 to 10, each corresponding to range of severity, and a specific color:

Table 7.1: Threat Levels

High	Red	8-10
Medium	Orange	4-7
Low	Yellow	2-3
Informational	Dark Blue	1
None	Light Blue	0

To assign threat level to a policy, you will need to be in the policy edit mode (click on the editing icon). Next, use the drop down, which is located next to policy screen. This is displayed in Figure 7.9.

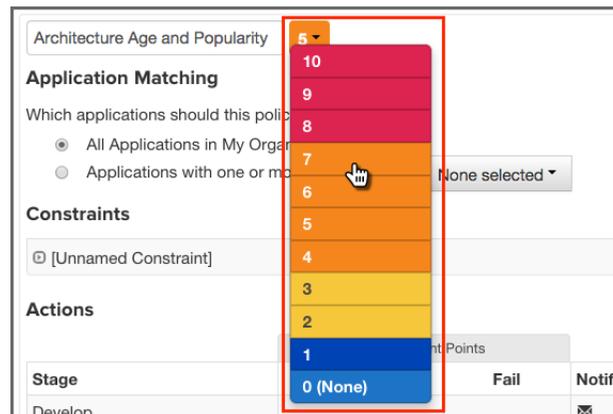


Figure 7.9: Editing the Policy Threat Level

When a threat level is selected by any component that has violated a policy will be displayed in the order of the severity of the violation. This is straight forward, but let's take a look at the details and see where some issues can arise.

Just like taste is subjective, you have to realize that severity is subjective. What may be a level 10 for one part of the business may be quite different for another.

A bigger concern is what message you want to communicate to the recipients of the analysis. For example members of your team will naturally be inclined, to treat high severity as very important, and critical. If they see too many of these, the prospect of ever making any headway could be impacted in a negative way and you might dismay e.g. your developers to even get started on trying to fix the violations since there are just too many. As with any control mechanism it is probably advisable to be a bit lenient at first and start to get stricter at a later stage, when the worst problems are dealt with. Otherwise you could cause undue stress to development and management teams. The important thing is to keep in mind is how you will assign severity, and the impact it has on those reviewing violations.

Especially in the beginning, our recommendation is to reduce, if not exclude the high severity threat levels altogether. Remember, at the core, this number is purely for ordering how a violation will be displayed to a user of Sonatype CLM. Because this needs to be interpreted, the actual value is very subjective. It's

easy to overwhelm members of your team by seeing large numbers of highly severe violations. It might even be reasonable to set severity to the lowest setting for a violation (Low - 1) in the beginning. This way, members of your teams can get used to seeing the results that are produced.

In our example policy, architecture quality is pretty important, but it's not the most important issue, so we'll leave the default threat level of 5. This will give us a violation higher up in the display, but it's certainly not the worst.

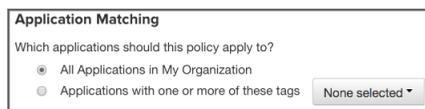
Tip

Remember, threat is subjective, and is most useful for ordering violations.

7.3.4 Step 4: Choose the Application Matching Parameters

In many cases, policies will be evaluated against all applications. However, as your applications become more diverse and there are various groups of applications with similar characteristics, you may find the need to develop policies for these specific cases. This will go hand-in-hand with the [creation of tags](#), which application owners will apply to their applications.

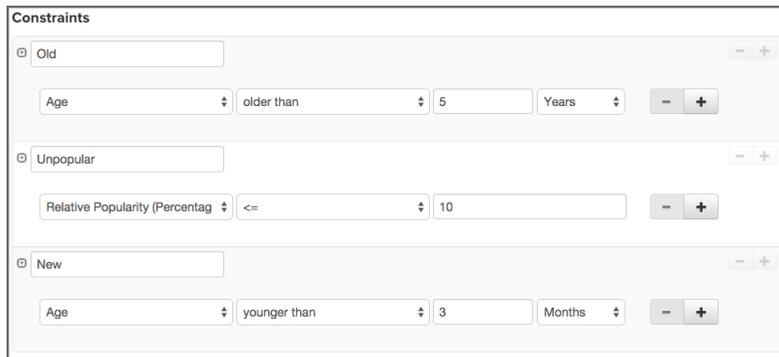
Given the ability to create tags, there is an endless amount of refinement that can be done in order to focus a particular policy on a set of applications with similar characteristics. While we won't get into the creation of tags here, thinking about this as a way to create more finely tuned policy is a good first step. For now, we'll assume we have some pretty generalized architecture requirements for all applications, and want to make a policy that will match all applications in this organization.



The screenshot shows a dialog box titled "Application Matching". Below the title is the question "Which applications should this policy apply to?". There are two radio button options: "All Applications in My Organization" (which is selected) and "Applications with one or more of these tags". To the right of the second option is a dropdown menu currently showing "None selected".

7.3.5 Step 5: Create Constraints with Conditions

By now, we should know that an easy way to look at conditions is to consider them the `if` part of an `if/then` statement. In contrast, a constraint is not part of the statement, but rather a container for conditions. This can come in real handy when you want to consolidate a policy, by grouping similar conditions.



The screenshot shows a 'Constraints' panel with three expandable sections. Each section has a title, a field for the constraint name, and a list of conditions with their respective operators, values, and units. The 'Old' section has a condition 'Age' with operator 'older than' and value '5' in 'Years'. The 'Unpopular' section has a condition 'Relative Popularity (Percentag)' with operator '<=' and value '10'. The 'New' section has a condition 'Age' with operator 'younger than' and value '3' in 'Months'. Each section also has expand/collapse icons.

Figure 7.10: Example Constraint

But why do this?

Well, for one thing, it reduces time. Next, because constraints are displayed on the application composition report, along with violations, it becomes easier for a member of your team to process information that is similar. Of course, it also has the added benefit of not having to create hundreds of small, one-off policies.

For example, in our *Architecture Age and Quality* policy, we may start with two conditions, one for age and one for popularity. This will work, but it's likely better, especially if we want to expand that later on, we isolate these as separate constraints.

As a good practice, grouping similar conditions is the best way to build your constraints. Again, this will help de-clutter a multi-policy environment, as well as help someone reading any associated violations on the application composition report. For our example, we'll be placing both conditions in the same constraint.

1. Click on the *Expand/Collapse* icon (shaped like a right-facing triangle). It's next to the *Constraint Name* and should display *Unnamed Constraint*.
2. Once the constraint is expanded, click the *Constraint Name* field and enter *Age and Popularity* as the constraint name where the field initially displays *Enter Constraint Name*.
3. Using the drop downs and the add (+) button, add two conditions.
 - a. *Age* should be checked to be *older than* the value of 3 *Years*.
 - b. *Relative Popularity (Percentage)* should be set to be greater than or equal to (\geq) 50 percent (50%).

4. Select *All* in the drop down below the conditions as the aggregating rule so that all of the above conditions have to be fulfilled to trigger a policy violation for the constraint.

Figure 7.11: Adding Constraints

Now, before we move on, let's take a look at some of the specific aspects of conditions that can tend to be a bit confusing.

Policy Type Sonatype CLM has an algorithm for determining the type of policy you created. This type is based on the types of conditions you include and is mainly used in reporting purposes. For example the trending report aggregate the different policy types into separate lists. The rules to determine the type of a policy are:

- If there are any security conditions, it is considered a security type policy.
- If there are any license conditions, it is considered a license type policy.
- If there are any age or popularity conditions, it is considered a quality type policy.
- If there are any conditions not mentioned above, it is considered an other type policy.

Any vs. All If a component meets any constraint, the policy is considered violated. However, inside each constraint, when you have more than one condition, you have the ability to require that a component must either meet all conditions or just any of the conditions to trigger a violation.

This is a straightforward concept, but it can produce vastly different results. For example, selecting *Any* as an option will tend to produce a lot more violations, it's the equivalent of placing an *or* between each condition. In addition even if you resolve one violation for a component, if the component meets other conditions later on, it might cause it to still violate the policy. On the other hand, using *All* allows you to establish that all conditions must be met. It's like placing an *and* between each condition. In this case, if you address any of the conditions, the violation will be resolved.

Let's look at an example that would further explain this, and then we will move on to creating conditions in our policy. Below, we have a couple of policies and a component. If we wanted to exclude components that were younger than four years and had less than fifty percent popularity, which would we want to use? Also, would the component listed violate this policy?

Policy 1 with Conditions

- Age is < 4 years
- Relative Popularity is \Leftarrow than 50%
- **Any** of the above conditions trigger the constraint.

Policy 2 with Conditions

- Age is < 4 years
- Relative Popularity is \Leftarrow than 50%
- **All** of the above conditions trigger the constraint.

Component Data

- Age is 3 years
- Relative Popularity is 85%

It may seem like these two policies are exactly the same, but there is a key difference, Any vs. All. So, our first policy states that if *Any* of the conditions are met that a violation will occur. Given that, our component violates this policy. This is good, but we only want a component to violate a policy when Age is less than three years, and popularity is below 50%. We understand that there may be circumstances where a component might need to fall outside one of those boundaries, but if it's both, we know we have a problem. Our component is pretty popular, so it might be a case where this component, even though it is newer than we prefer, actually resolves an issue another component might have. For this reason, the best policy for us to choose is Policy 2, and our component would not actually cause a violation.

Available Conditions

The following list enumerates the available conditions and describes the operator and the value used for the evaluation.

Age

Verify if the components age based on the date it was placed in the Central Repository in months, days or years is older our younger than a specified value.

Coordinates

Verify if the coordinates for a component *match* or *do not match* (e.g. *groupId:artifactId:version*).

The coordinates may be fixed, or a wildcard can be used. For example, *org.sonatype.com:nexus-indexer:1.**. Here, components with a groupId of org.sonatype.com, an artifactId of nexus-indexer, and any version starting with 1. would be matched.

Note

This condition is only applicable for Maven components.

Additional Examples

- A fixed coordinate: `org.sonatype.nexus:nexus-indexer:1.0`
 - A wildcard coordinate to an inclusive group and version: `org.sonatype*:nexus-indexer:1.*`
- this condition would match, for example `org.sonatypepetest` or `org.sonatype.example`, an artifact named `nexus-indexer` in these groups, and any version that starts with `1.` for those groups and that specific artifact.
-

Note

The use of the wildcard is limited to the end of each parameter in the coordinate.

Identification Source

Verify if the component identification *is* or *is not* based on data from *Sonatype* or your *Manual* identification

Label

Verify if a label with a specific label name *is* or *is not* attached to a component

License

Verify if the component license *is* or *is not* a specified license. If the component license has been overridden, anything listed as declared or observed will be ignored. If license status is not overridden, then any occurrence of the selected license (observed or declared) will be considered a violation.

License Threat Group

Verify if the components license *is* or *is not* in one of the groups *Banned*, *Copyleft*, *Liberal*, *Non Standard*, *Not Observed* or *Weak Copyleft*. If the component license has been overridden anything listed as declared or observed will be ignored. If license status is not overridden, then any occurrence of the selected license (observed or declared) in the selected License Threat Group, will be considered a violation.

License Threat Group Level

Verify if the components license threat in its license threat group is either lesser `<`, lesser or equal `<=`, greater `>` or greater or equal `>=` a specified value. Value range from zero for no threat to the highest threat level value of ten.

License Status

Verify if the user defined License Status *is* or *is not* one of the possible values of *Open*, *Acknowledged*, *Not Applicable* or *Confirmed*.

Match State

Verify if the match of the comparison of the component to known components from the Central Repository with the same coordinates *is* or *is not exact*, *similar* or *unknown*.

Proprietary

Verify if a component *is* or *is not* considered proprietary.

Relative Popularity (Percentage)

Verify if a the relative popularity of the component version as compared to other versions of the same component group and artifact parameter is either lesser <, lesser or equal <=, greater > or greater or equal >= a specified percent value.

Security Vulnerability

Verify if a security vulnerability is *present* or *absent*. Keep in mind that just because there is no known, or *present*, security issue, doesn't necessarily mean none exists.

Security Vulnerability Severity

Verify if a security vulnerability with a numeric severity is either lesser <, lesser or equal <=, greater > or greater or equal >= a specified value.

Security Vulnerability Status

Verify if a the components security vulnerability status *is* or *is not* one of the possible values of *Open, Acknowledged, Not Applicable* or *Confirmed*.

Now that you know about all the available conditions and how they can be combined to match all or any condition to create a constraint we are ready to determine what should happen if a constraint is met - the policy actions.

7.3.6 Step 6: Set Policy Actions And Notifications

Policy actions and notifications allow you to tell CLM to perform a specific function when violations occur. In some cases this can include forcing a particular action in one of the Sonatype CLM Tools (e.g. Sonatype CLM for Bamboo - installed separately from the Sonatype CLM Server).

Alternatively, notifications can be set so that when violations occur, any combination of particular email addresses, or the email addresses associated with the members of a particular CLM role, will be notified.

7.3.6.1 Actions

Actions are organized by enforcement point. In Sonatype CLM, enforcement points represent a stage in Component Lifecycle Management (i.e. Development, Build, Stage, Stage Release, and Operate).

For each stage there are two degrees of action you can assign to each enforcement point:

- Warn - generally will not break the process, and only displays violations that have occurred (e.g. Nexus CLM will display a warning during staging).
- Fail - generally will break the process (e.g. prevent a build in Sonatype CLM for Bamboo).

To add actions:

1. First make sure you have:
 - a. The policy open to edit (have clicked the edit icon - shaped like a pencil).
 - b. A minimum of owner rights (a member of the owner role) for the application or organization the policy resides in.
2. Next, click on either the warn or fail action in the column for the particular stage. An icon will confirm your selection.
3. Click *Save*, or proceed to adding notification in the next section.

Stage	Enforcement Points		Notifications
	Warn	Fail	
Develop			<input checked="" type="checkbox"/>
Build			<input checked="" type="checkbox"/>
Stage Release			<input checked="" type="checkbox"/>
Release			<input checked="" type="checkbox"/>
Operate			<input checked="" type="checkbox"/> ArchitectureLead@MyCompany.com

Figure 7.12: Policy Actions Example

Tip

Take care in being too anxious to stop your development cycle using an action. For example, setting the Fail action at the Build enforcement point.

7.3.6.2 Notifications

When a new violation occurs, a notification will be sent. This includes any email addresses entered manually, as well as email addresses for the users that have been added to any roles selected.

To add notifications:

1. Click anywhere within the notifications field. This will display a modal allowing you to select a role or enter a specific email address.
2. Enter an email address manually, and or select any roles you wish to be notified.
3. Click *Save*.

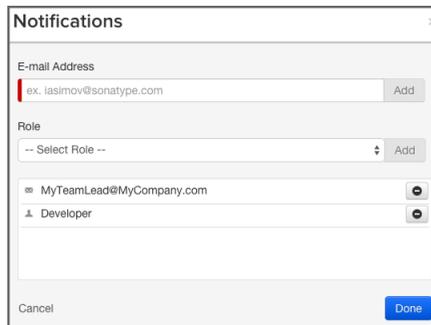


Figure 7.13: Policy Notifications Example

Note

When a notification is sent, it will only display new violations found in the latest scan. If you find yourself not receiving notification, verify there are new violations, as well as confirm you have configured your Sonatype CLM server SMTP settings.

7.3.6.3 Stages

Both Actions and Notifications are set by stage. A description for each stage as well as suggested actions and/or notifications have been listed below.

Develop - (e.g. Sonatype CLM for Eclipse)

Definition

The **Develop** stage represents development.

Suggested Actions

While actions and notifications can be configured for this stage, they will not affect the functionality of Eclipse.

Build - (e.g. Sonatype CLM for Bamboo or Hudson/Jenkins)**Definition**

The **Build** stage represents actions in tools using during the building of your applications.

Suggested Actions

As you manage policy, making necessary adjustments over time, it's best to take an approach that allows for your development teams to be eased into dealing with violations. For this reason, it's better to start by simply warning when the build for an application contains components that violate your policies.

Suggested Notifications

Consider setting up notifications to both inform Application Owners, as well as developers.

Stage Release (Nexus Pro CLM)**Definition**

The **Stage Release** stage is specific to Nexus CLM, and involves placing a near final build into a staging repository prior to having it officially released.

Suggested Actions

Because this stage gives the opportunity to prevent an application from being released with components that have violated policy, setting the action for a Stage Release to Fail is our recommendation. This is especially true for any policies that may include risk associated with security and/or licensing.

Suggested Notifications

If something fails, the development process can't move forward. Make sure to notify any members of the team responsible for the release of the application and capable of researching and addressing any violations.

Release (Nexus Pro CLM)**Definition**

The **Release** stage is the final push for a project into production.

Suggested Actions

While there should be the closest scrutiny of policy violations at this point, there will be a similar recommendation to fail a release based on severe violations. In most cases, you should ideally be finding new violations only.

Suggested Notifications

Similar to **Stage Release** make sure all stakeholders, that is those members responsible for ensuring an application does not go into production with policy violations, are notified.

Tip

If you have setup policy monitoring, it is a good idea to monitor your release stage, as this is likely the best representation of your production application.

Operate**Definition**

The **Operate** stage represents the best known example of the application in its production state. It can be set via a variety of tools, but in all of these cases, it is set manually.

Suggested Actions

For this reason, providing any warning or fail actions will not produce any different result.

Suggested Notifications

Typically the application owner, or anyone responsible for ongoing maintenance of an application in production should be notified. However, remember that evaluation in the **Operate** stage is manual.

7.4 Policy Elements

One of the more interesting pieces of Sonatype CLM Policy, and something we've only alluded to in a number of our other chapters, is policy elements. At this time there two specific policy elements we can work with:

- Labels
- License Threat Groups
- Tags

These elements are unique in that they have both links to policy as well as general Sonatype CLM functionality. Let's take a closer look at each one of these.

7.5 Labels

We just learned all about creating policies. Constraints, conditions and even actions, all make a lot of sense, but what in the world could labels have to do with policy?

Well, labels are actually one of the more powerful features of Sonatype CLM. They should have a familiar look, since you've likely used other systems that employ a sort of tagging or labeling.

Labels are metadata. More specifically a label is metadata that is assigned to a component within the context of a particular application or organization. Labels can assist with identifying components you want to review, approve, or even avoid altogether. We call this label assignment.

When labels are assigned, this is an action that takes place in the application composition report. Before it can be assigned though, a label needs to exist for a particular organization or application.

As we learned in our section regarding Organizations vs. Applications, inheritance plays a big role in policy. The same thing is true for labels, in that if a label is created in an organization, any application attached to that organization will also have the label available for use when assigned. In fact, the system will prompt you to choose the scope (organization or application) a label should exist in when it is assigned.

7.5.1 Creating, Editing and Deleting a Label

We've determined that assigning a label is an important action, but how do we build policy around this? That's actually simple, we just add a condition, based on a specific label that you have created, being present. The one caveat, is that the label needs to exist within the application or the organization in which you are creating the condition.

There are two key ways to create a label:

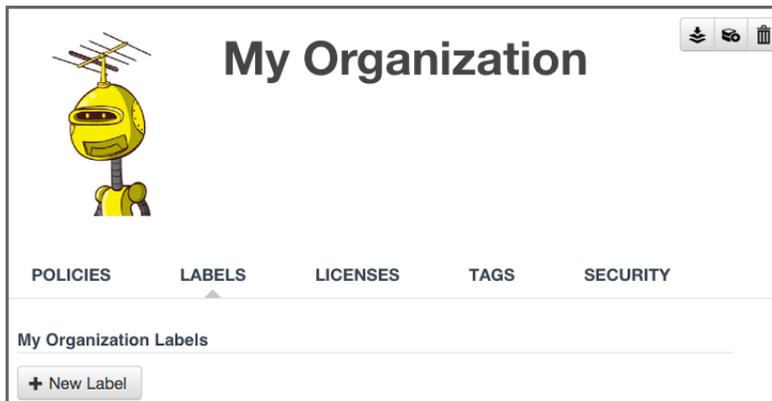


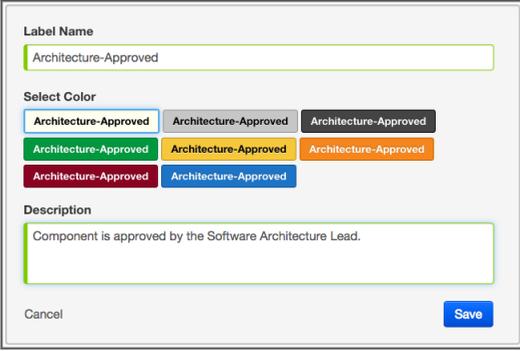
Figure 7.14: Using New Label Button



Figure 7.15: Using Global Create Button

There is really no difference here, as both require that you have the organization or application open at the time of creation. The one advantage with using the Global Create button is that you can create no matter which tab of the currently selected organization or application you are in, whereas you will need to be on the Label tab otherwise.

When creating your label, remember to use something that is easily identifiable. If you're following along with our example in the next section, *Architecture-Approved* is a good example.



The screenshot shows a dialog box for configuring a label. It is titled 'Label Name' and contains a text input field with the value 'Architecture-Approved'. Below this is a section titled 'Select Color' which displays six buttons, each labeled 'Architecture-Approved' and colored differently: cyan, grey, green, yellow, red, and blue. The third section is titled 'Description' and contains a text area with the text 'Component is approved by the Software Architecture Lead.'. At the bottom of the dialog are two buttons: 'Cancel' on the left and 'Save' on the right.

Figure 7.16: Label Example

Once you have entered the name, you can enter an optional description, and then click the *Save* button.

Editing Labels

To make changes to a label, click on the label and the label information will be displayed below.

Deleting Labels

To delete a label, just click on the X next to the label name.

A few things to remember:

- An organization's labels can be seen by any of its applications, the reverse is not true.
- Labels can only be edited (or deleted) at the level they were created.

7.5.2 Creating a Condition Based on a Label

In the example below a new condition for the label *Architecture-Approved* will be added to an existing policy with an existing constraint and condition.

In our instructions, we've made an assumption that you understand how to [create a policy](#).

The screenshot shows a 'Constraints' panel. At the top, there is a search box containing 'Age' and a plus icon. Below this, there are three rows of configuration. The first row has a dropdown menu with 'Age' selected, followed by a dropdown with 'older than', a text input with '3', and a dropdown with 'Years'. To the right of this row are minus and plus buttons. The second row has a dropdown with 'Label' selected, a dropdown with 'is not', and a green button labeled 'Architecture-Approved'. To the right of this row are minus and plus buttons. The third row has a dropdown with 'All' selected, followed by the text 'of the above conditions will trigger this constraint.'

Figure 7.17: Creating a Label Condition

1. Open an existing policy.
2. In the *Constraints* area of the policy, click on the + icon, located next to the right of an existing condition.

Tip

Make sure you use the correct + icon, as it can be easy to add a new constraint by mistake.

3. Now, in the *Conditions* area, change *Label* in the first drop down menu from *Age* to 'Label'.
4. Next, in the second drop down menu select *is not* for the operator.
5. Finally, in the third drop down menu, select the *Architecture-Approved* label you just created.
6. Click the *Save* button to finish.

Tip

Because our example uses a constraint with an existing condition, we have also chosen to force a violation only when all conditions have been met. In this scenario it may be appropriate to consider a [waiver](#) as an alternative.

7.6 License Threat Groups

License threat groups, are simply groups of licenses, broken into categories of severity for the various types of licenses. They can help you to achieve your goals related to enforcing the usage of components with licensing that matches the scope of your application.

Their primary purpose is to serve as the data points for the License section of the application composition report. Moreover, they are a way to group risk, associated with licensing. By default, there are four license threat groups included with Sonatype CLM:

Copyleft

Strong copyleft licenses go a step further from weak copyleft licenses and mandate that any distributed software that links or otherwise incorporates such code be licensed under compatible licenses, which are a subset of the available open-source licenses. As a result, these licenses have been called viral.

Non Standard

Something out of the ordinary (e.g. If we ever meet, give me a beer license).

Weak Copyleft

Free software licenses that mandate that source code that descended from software licensed under them, will remain under the same, weak copyleft, license. However, one can link to weak copyleft code from code under a different license (including non-open-source code), or otherwise incorporate it in a larger software. Otherwise, weak copyleft licenses allow free distribution, use, selling copies of the code or the binaries (as long as the binaries are accompanied by the (unobfuscated) source code), etc.

Liberal

These licenses allow you to do almost anything conceivable with the program and its source code, including distributing them, selling them, using the resultant software for any purpose, incorporating into other software, or even converting copies to different licenses, including that of non-free (so-called “proprietary”) software.

Note

Consult with your legal department for EXACT definitions. Information provided above is from the following [reference](#).

7.6.1 Creating, Editing, and Deleting a License Threat Group

An important aspect of license threat groups is that each one also has a threat level, just like policy (from zero signifying no threat all the way up to 10). Unless you have specific legal recommendation / council, the default license threat groups will suffice, especially in the beginning.

If you desire, you can edit these default groups, or create entirely new ones. When creating license threat groups, keep in mind that they will be inherited from the organization to all associated applications.

There are two key ways to create a license threat group:



Figure 7.18: Using New License Threat Group Button



Figure 7.19: Using Global Create Button

There is really no difference here, as both require that you have the organization or application open at the time of creation. The one advantage with using the Global Create button is that you can create no matter which tab of the currently selected organization or application you are in, whereas you will need to be on the License tab otherwise.

The following information needs to be completed before a license threat group can be saved.

Name

This is the name for your license threat group. When creating or editing the name of a license threat group, remember to use something that is easily identifiable. If you're following along with our example in the next section, use *Banned Licenses*.

Threat Level

This is the level of threat this group of licenses should represent.

Applied and Available Licenses

Adding licenses to the license threat group is not an actual requirement, but there really isn't much use for simply creating a group as a placeholder. So this is treated as a required field.

On the left are licenses that are included in the license threat group. Click on a license to remove it.

+ On the right are the licenses that can be added the group. Click on a license to add it.

When everything is done your screen should look like Figure 7.17 and you can click the *Save* button to finish.

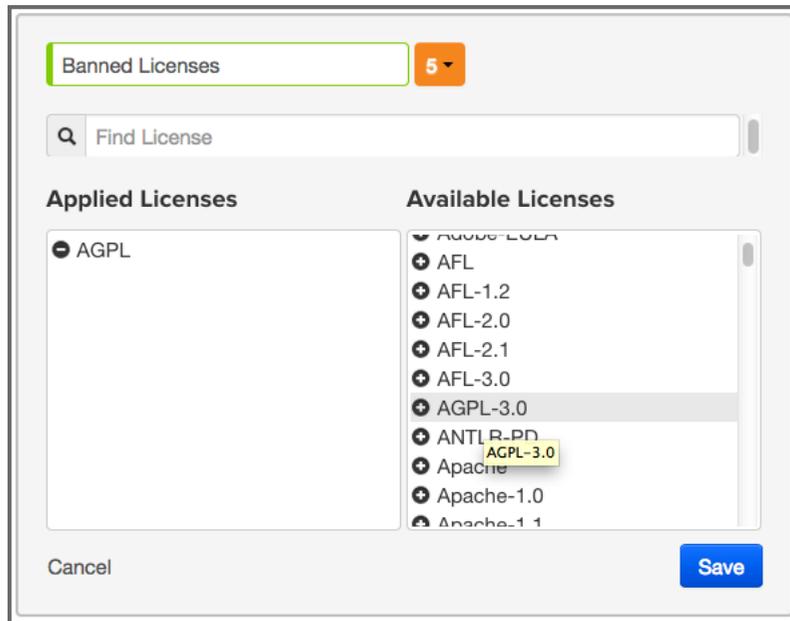


Figure 7.20: Creating a License Threat Group

Editing

To make changes to a license threat group, click on the *Edit* icon (shaped like a pencil).

Deleting

To delete a license threat group, just click on the *Delete* icon (shaped like a trash can) next to the label name.

A few things to remember:

- A set of four default license threat groups are provided.

- Applications inherit license threat groups from their organization.
- An organization's license threat groups can be seen by any of its applications, the reverse is not true.
- License threat groups can only be edited (or deleted) at the level they were created.

7.6.2 Creating a Condition Based on a License Threat Group

In the example below a new condition for the license threat group, *Banned Licenses*, will be added to a new policy.

In our instructions, we've made an assumption that you understand how to [create a policy](#).

1. Create a new policy.
2. In the *Constraints* area click on the *Expand/Collapse* icon (shaped like a right-facing triangle). It's next to the *Constraint Name* and should display *Unnamed Constraint*.
3. Once the constraint is expanded, click the *Constraint Name* field and enter *Banned License*.
4. Now, in the *Conditions* area, change *Age* in the first drop down menu to *License Threat Group*.
5. Next, in the second drop down menu choose *is* for the operator.
6. Finally, in the third drop down menu, find and select the *Banned License* label you just created.
7. Click the *Save* button to finish.



Figure 7.21: Creating a Condition Evaluating a License Threat Group

7.6.3 Creating a Condition Based on an Unassigned License Threat Group

In most cases, a license is associated with one or more [License Threat Groups](#). However, it is possible for a license to have no association with any [License Threat Group](#). You can create a Policy to detect when a component has a license that is not assigned to any License Threat Group.

In the example below a new condition for detecting components with licenses not assigned to any License Threat Group will be added to a new policy.

In our instructions, we've made an assumption that you understand how to [create a policy](#).

1. Create a new policy.
2. In the *Constraints* area click on the *Expand/Collapse* icon (shaped like a right-facing triangle). It's next to the *Constraint Name* and should display *Unnamed Constraint*.
3. Once the constraint is expanded, click the *Constraint Name* field and enter *Unassigned LTG*.
4. Now, in the *Conditions* area, change *Age* in the first drop down menu to *License Threat Group*.
5. Next, in the second drop down menu choose *is* for the operator.
6. Finally, in the third drop down menu, find and select *[unassigned]*.
7. Click the *Save* button to finish.



Figure 7.22: Creating a Condition Evaluating an unassigned License Threat Group

A violation of the policy above can be remediated simply by assigning the license involved to a [License Threat Group](#).

To remediate a specific component, use the Component Information Panel (CIP) *License* tab to set the license *Status* to *Selected* or *Overridden*, and then select a license that is associated with at least one [License Threat Group](#). Managing component licenses is discussed further in the [editing License Status and Information](#) section of the Application Composition Report chapter.

7.6.4 Tags

In any given business, you could have hundreds, maybe even thousands of applications. Even if you are just getting started, it's likely you have a handful of applications. However, as unique as applications can be, they tend to share some similarities.

For example, you might have applications that process or store sensitive information, maybe even personally identifiable information for your users. Since attacks are often aimed at these types of applications, you will definitely want to make sure your policies that identify high and critical threat security vulnerabilities are included during the evaluation of these types of applications.

Unfortunately, especially as the number of applications in your business increases, identifying an application by name may not be helpful. To address this, tags provide a way to quickly identify characteristics of an application.

Using specific text and color, tags can help group particular applications with similar attributes. While the tag can ultimately be anything you want, and attached to any application, you will want to take a much more thought-out approach, similar to what is recommended for labels.

As we will see later, in order to maximize the benefits tags can offer, you will want to take advantage of tag matching between policies and applications. For now though, let's see how to create, apply, and delete tags.

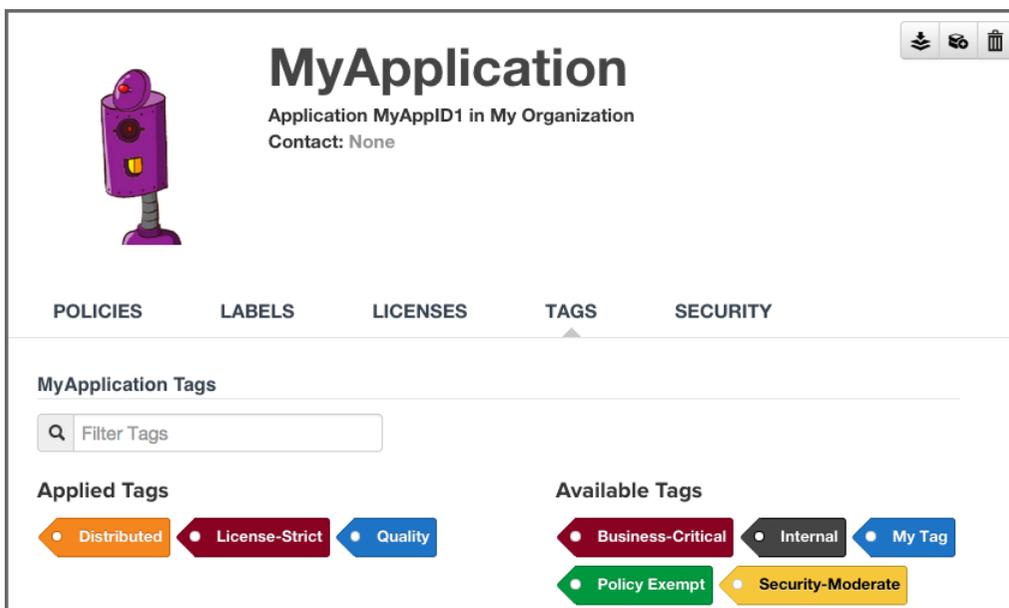


Figure 7.23: Example of Applied Tags

7.6.5 Creating, Editing, and Deleting Tags

Tags are created, edited, and deleted at the organization level and then applied individually for each application. There are two key ways to create a tag, again, only done at the organization level.

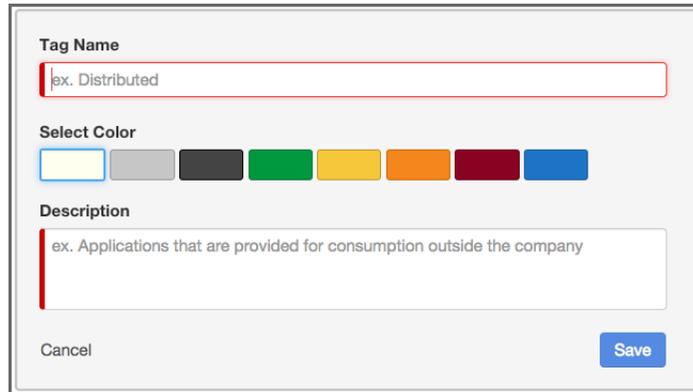


Figure 7.24: Using New Tag Button



Figure 7.25: Using Global Create Button

There is really no difference here, as both require that you have the organization open at the time of creation. The one advantage with using the Global Create button is that you can create a tag no matter which tab of the currently selected organization you are in, whereas you will need to be on the Tags tab otherwise.



The image shows a dialog box for creating a tag. It has three main sections: 'Tag Name' with a text input field containing 'ex. Distributed'; 'Select Color' with a row of eight color swatches (white, grey, black, green, yellow, orange, red, blue); and 'Description' with a text input field containing 'ex. Applications that are provided for consumption outside the company'. At the bottom left is a 'Cancel' button and at the bottom right is a 'Save' button.

Figure 7.26: Creating a Tag

There are three elements of tag:

Tag Name

When creating your tag, keep in mind that the tag describes characteristics of an application, and will be used to match an application to corresponding policies. The name should be easily identified by the user.

Tag Description

The tag description is displayed when a user hovers the mouse over the tag. This can offer additional information, such as the types of policies that will be matched to applications that have applied the tag.

Tag

Color

The color selection is left to however your organization chooses to implement. The default is white.

If you made a mistake and want to edit the tag, simply click on the tag body (anything but the *x*), and you can edit the tag information. However, if you want to permanently delete the tag, click on the *x*.

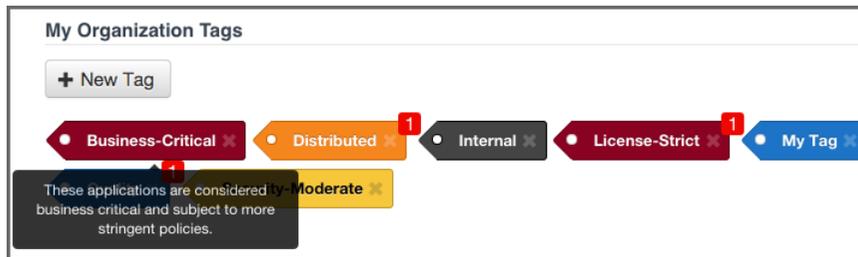


Figure 7.27: Example of Tags with Description

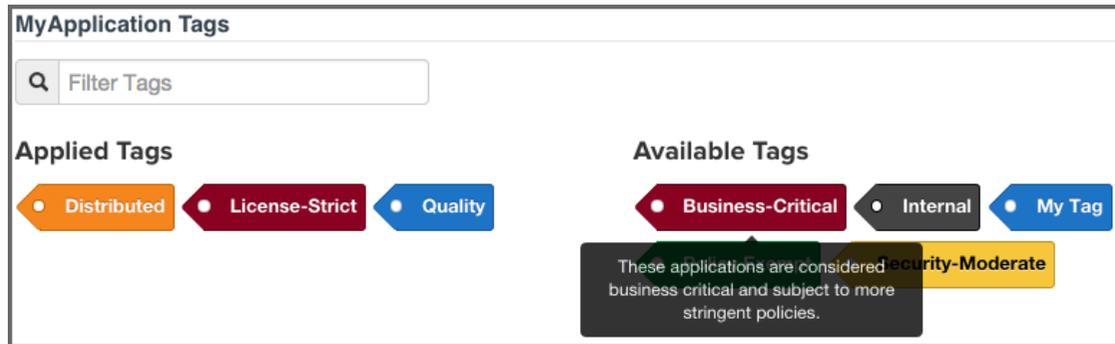
Note

Deleting any tag will ask for you to confirm, since that action can not be undone. If the tag is currently applied to an application you will be shown the names of all applications that would be affected before you confirm the deletion. You will not be able to delete a tag that has already been related to a policy, and will be shown the names of any related policies if you try. Should you still wish to delete the tag, you will have to disassociate it from any related policies first.

7.6.6 Applying a Tag

Depending on how your business uses tags, and establishes control within CLM, the people applying tags may be different from those creating them. It is important though to understand that while tags are provided to identify characteristics of an application, a more important usage is to provide a way for policy managers to create specific policies that consider those application characteristics. For this reason, when applying a tag, your application may be evaluated by a specific set of policies. This is a good thing, but it also makes the application of tags an act that requires careful consideration. To apply a tag to an application, follow the instructions below.

1. First, log in to the Sonatype CLM Server using a user account with at least Owner-level permissions for the application (a member of the Owner Group).
 2. Next, click on the *Application* link, and then click on the *application* you want to apply the tag to. The **Application Management** area will be displayed.
 3. Now, click on *Tags* tab. There are two columns, one for available tags, as well as those that have already been applied. Simply click on the tag to move it from one column to the other. If there are a lot of tags, and you are having trouble locating a specific one, simply type in the filter the name of tag you would like to use.
-



Tip

Mouse over a tag to see the full description.

7.6.7 Matching Policies to Specific Applications

By now, you have likely created tags, and perhaps even applied some to your applications. Those are great features, but the real power of tags comes when we match a policy to a specific set of applications.

Up to this point (before tags), an organization-level policy would apply to all applications. To address this, you could create a new organization, or develop specific policies for each application, but in both cases, that results in a lot of micromanagement. In contrast, tags provide an opportunity to create a policy and then pick unique groups of applications (based on their applied tags) the policy should be evaluated against.

Given this, it is important to think about the applications your business develops, as well as the types of policies you will use to evaluate your applications. Elements like the type of data, the exposure (public or private), as well as whether or not the application interfaces with the Internet, are a great place to start.

When you create your tags, make sure that it's clear to users that will be using the tags. In other words, it shouldn't be ambiguous as to the type of applications the tags represent. For example instead of creating the tag, *External*, a more descriptive tag would be *Distributed*. Some additional tag suggestions might be:

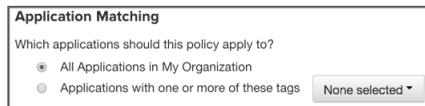
- Sensitive Information
 - Personal Information
-

These are just suggestions of course, but you should get the key point. When adding a tag to an application, you can expect policies that have identified the same tag to be evaluated against your application.

Now, that's quite a bit of discussion on the theory and proper way to utilize tags, let's take a look at how to make the match happen.

To select the tag a policy will be evaluated against:

1. Create a new policy, or edit an existing one.
2. In the Application Tags area of the policy editor, choose the tags that represent the applications you want to evaluate the policy against. By default, no tags will be selected. This means the policy will apply to all application regardless of their tags.
3. Finish creating or editing your policy and click the *save* button. From this point forward, the policy will be evaluated against applications based on the tags you selected. In addition, applications will only see the policies they are evaluated against.



The screenshot shows a dialog box titled "Application Matching". Below the title is the question "Which applications should this policy apply to?". There are two radio button options: "All Applications in My Organization" (which is selected) and "Applications with one or more of these tags". To the right of the second option is a dropdown menu that currently displays "None selected".

7.6.8 Viewing Tag-based Policies

Policies that have been set to match applications with specific tags are visible in the same area as all other policies. However, there is a slight difference between what is displayed at the organization level and the application level.

At the organization level

All policies for the organization will be displayed. Policies that have selected specific applications, based on a matching tags applied to those applications, will be indicated by a special icon.

At the application level

Only the policies that an application is evaluated against will be displayed in the Policy tab. This includes:

- Policies created at the organization level, and set to match all applications.
- Policies created at the organization level, and set to match specific tags currently applied to the application.

- Policies created at the application level.

Tip

When viewing policies at the application level, be sure to look for the special tag icon , which indicates the application is evaluated against the policy given a tag (or tags) applied to the application.

7.7 Evaluating via the CLM Server

In order to evaluate an application, you need to have created at least one organization and one application, as well as created or imported at least one policy at either the organization or application level. You will also need to make sure you have the proper permissions to view report information for the application you wish to evaluate.

While evaluations can be initiated from various tools featuring CLM integration (e.g. Sonatype CLM for CI, IDE, and Nexus Pro), the quickest way to get started is to perform an evaluation via the CLM server.

This will generate a report for your application quite easily, and is a great way to create a quick baseline of your application's health.

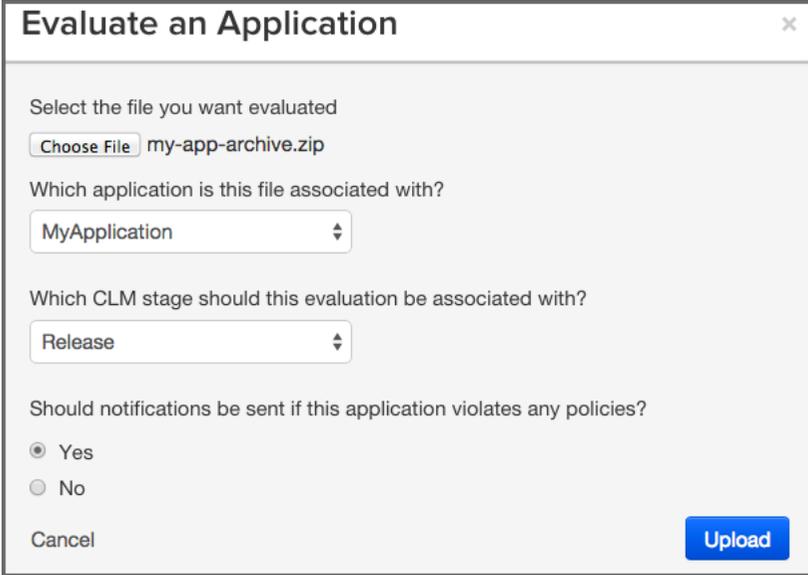
As mentioned previously, before you can evaluate an application, you will need to make sure you have:

- Created an organization
- Created an application
- Imported or created a policy

With the above complete, you are ready to evaluate an application via the CLM Server.

1. First, log in to the CLM Server. At a minimum you will need to be a member of the Application Owner role.
 2. Next, click the *Manage Applications and Organizations* icon  to access the **Application and Organization Management** area. Once there, click *Applications* (located in the menu on the left side of the screen), and then choose an application.
-

3. In the top right of the **Application Management** area, click the *Evaluate an Application* icon .
4. A modal dialog will display providing a number of required fields.
 - a. First, choose the bundle (application) you want evaluated. Clicking *Choose File* will allow you to browse your directories for the application you wish to evaluate.
 - b. Next, choose the application in Sonatype CLM you want to associate with the evaluation. By default, this will be pre-populated with the name of the application you first selected.
 - c. After choosing the application to evaluate, you will need to specify the stage, this will affect where the report is displayed, and will overwrite the most recent report for the application and stage selected.
 - d. Finally, if you have configured notifications for your policy, or policies, you can choose whether or not you want those notifications sent.
5. Click the *Upload* button to begin evaluating the chosen application.
6. The Evaluation Status will display, showing you the progress of your evaluation. When complete, you can click the *View Report* button to view the results of your evaluation.



Evaluate an Application ×

Select the file you want evaluated

my-app-archive.zip

Which application is this file associated with?

MyApplication

Which CLM stage should this evaluation be associated with?

Release

Should notifications be sent if this application violates any policies?

Yes

No

Figure 7.28: Evaluate an Application

Note

You can also evaluate an application via the *Organizations* area, simply click on *Organizations* instead of *Applications* and follow the instructions from there. You will still need to have created an application, and the application won't be pre-filled for you in the form.

If your evaluation completed successfully, a report will be generated for the application.

The log output of the command execution will provide a summary as well as a link to the produced results similar to

```
[INFO] Policy Action: Warning
[INFO] Summary of policy violations: 4 critical, 85 severe, 46 moderate
[INFO] The detailed report can be viewed online
at http://localhost:8070/ui/links/application/my-app/report/95c4c14e
```

This report is available on the Sonatype CLM Server in the *Reports* section. If you kept our defaults, the report will be listed under the Build Stage. So, what are you waiting for? You should see something similar to the results displayed in Figure 7.29

Application Name ▼	Build Violations	Stage Release Violations	Release Violations	Contact	Organization
MyApplication	28 159 3 1 minute ago	28 159 3 1 minute ago			My Organization
My Application 4			6 5 1 month ago		My Organization 3
My Application 3	6 11 1 5 months ago				My Organization 7
My Application 2	6 11 1 5 months ago	6 11 1 1 month ago	6 11 1 6 months ago	John Smith	My Organization 4

Figure 7.29: Violations Report after Scan

Note

As mentioned previously, if you specify a stage not represented by in the Sonatype CLM Server, there will not be a visible link to the report.

7.8 Reviewing Evaluation Results

The Application Composition Report provides the results of an evaluation of your application. The results are broken into three key categories:

- Policy Violations
- Security Vulnerabilities
- License Issues.

As mentioned previously, this will be the same report, whether you are using the stand-alone scanner, the CLM Maven plugin, the manual evaluation, or any of the integrated enforcement points (e.g. Sonatype CLM for CI, IDE, Nexus Pro).

Let's take a look at how to access the report first.

Note

Depending on the enforcement point, or the stage options you manually selected, your report may be listed under different stages in the *Reporting* area of the Sonatype CLM Server. For example, the default location for the stand alone scanner, is the build stage.

No matter how the scan was performed, all reports reside on the Sonatype CLM Server and are automatically associated with the corresponding application (via the application identifier). However, there are two distinct ways to access the Application Composition reports.

Via the Reports Area

When you log into the Sonatype CLM Server the Dashboard is displayed by default. Click the *Reports* icon . If multiple applications have been scanned, you will see all of them here.

Note

You will need to be a member of at least the developer group for the application you wish to see a report for.

**Important**

Users of Nexus CLM Edition do not have access to the Sonatype CLM Dashboard. Because of this, these users will not be taken to the dashboard after logging in, nor will they see the dashboard icon. Rather, the reports area will display by default.

Each application has a separate row with columns for:

- Application Name
- Violations (by stage)
- Contact
- Organization (for the corresponding application)

Each Violation column contains a Violation Summary (total counts for *Critical*, *Severe* and *Moderate* policy violations). In addition, the time the last report was generated (e.g. *2 minutes ago*) is provided.

To access the Application Composition report, click the Violation Summary for the corresponding application and stage.

Application Name ▼	Build Violations	Stage Release Violations	Release Violations	Contact	Organization
 MyApplication	28 159 3 1 minute ago	28 159 3 1 minute ago			My Organization
 My Application 4			6 5 1 month ago		My Organization 3
 My Application 3	6 11 1 5 months ago				My Organization 7
 My Application 2	6 11 1 5 months ago	6 11 1 1 month ago	6 11 1 6 months ago	John Smith	My Organization 4

Figure 7.30: Reporting Area

Tip

By default this view will be sorted alphabetically by the application name. In addition to the filter, you can also click on the application or organization columns to sort alphabetically ascending/descending.

Via the Application Area

The Application area is the same place where you can manage policy for your application, reviewing policies unique to the application, as well as those inherited from the organization. Located just below the application identifier and organization, you will see three columns:

- Build
- Stage Release
- Release

These represent the Sonatype CLM stage where the report was generated for/from. For example, if you use the Sonatype CLM stand-alone scanner and don't specify the CLM Stage, it will default to build. When your scan completes and the report is uploaded, it would appear below *Build*. This is highlighted in Figure 7.31.

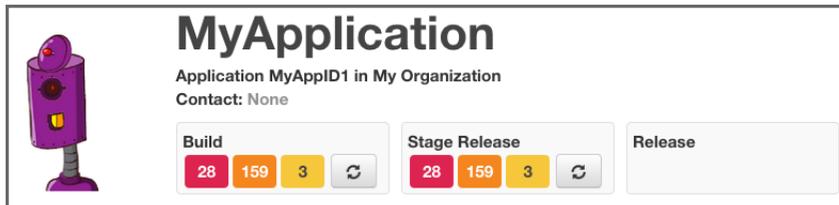


Figure 7.31: Application Area

At first glance, you may be surprised at what you see. If you expected an application to have no issues, and now see it has a great deal, don't get upset. . . yet.

In many cases, a policy can be too stringent or may indicate issues that are not exactly applicable to your application. For example, you may have a security issue that would only affect applications exposed to the public, while your application is for internal use only. Another great example is a license that constrains your code in the event you intend to sell the application.

With that worry out of the way, let's take a look at what's actually in each report.

The *Summary* tab of the report shows a breakdown of what was found. This includes counts for policy violations, security vulnerabilities and license-related issues.

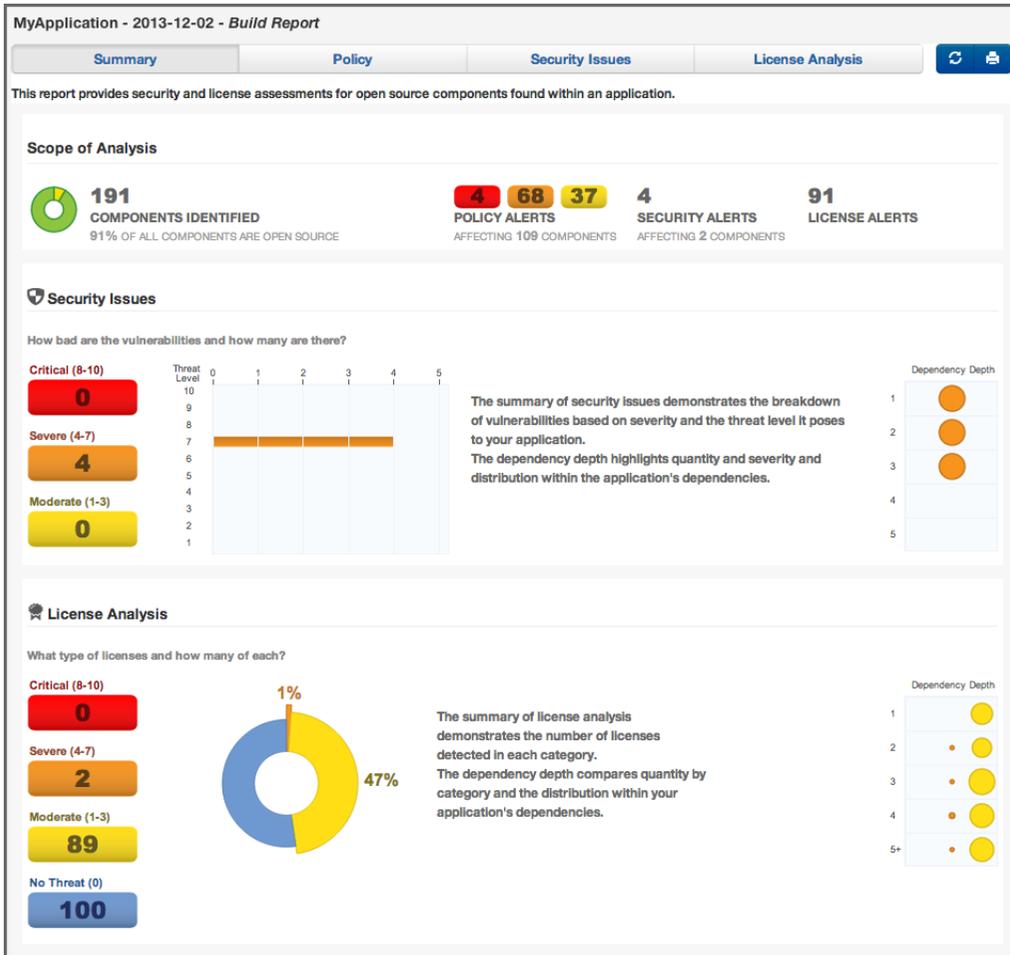


Figure 7.32: Summary Tab of an Application Composition Report

The *Policy* tab provides a list of all components that were found in your application. An example is displayed in Figure 7.33. The list of components is ordered by the level of the threat violation that has been assigned to the policy. In instances where a component has violated multiple policies, only the violation with the highest threat is displayed.

To view the other violations you can use the component information panel (described below), or change what is displayed using the Violations filter on the right. This will allow you to see all violations for your component, though that may result in the appearance of duplicated components.

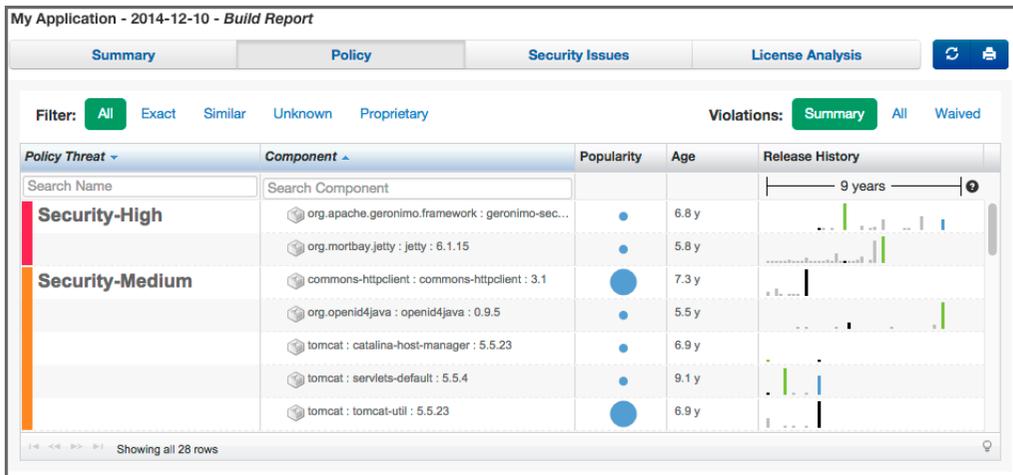


Figure 7.33: Policy Tab of an Application Composition Report

The *Security Issues* tab displayed in Figure 7.34 displays all components containing security issues.

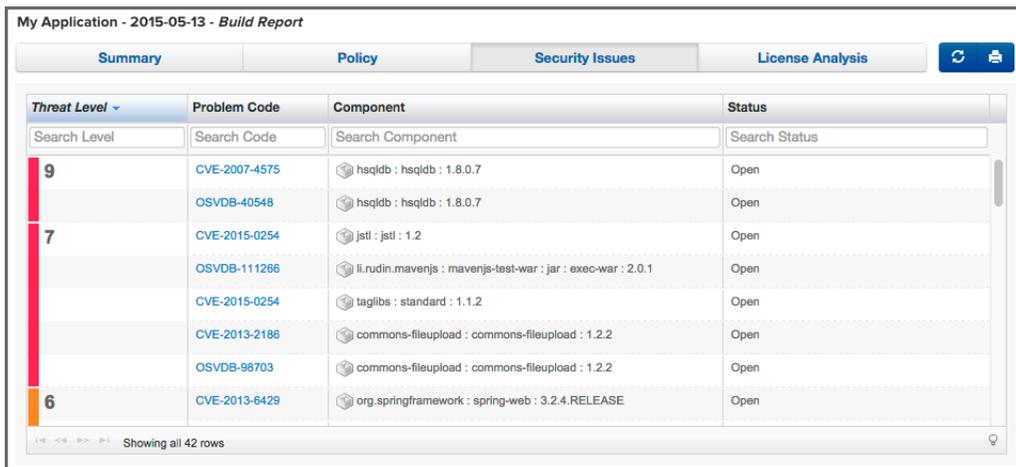
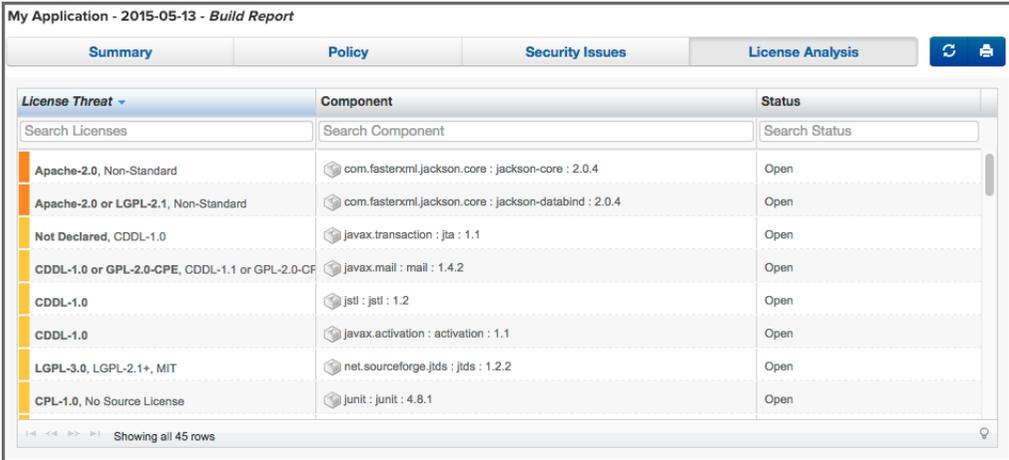


Figure 7.34: Security Issues Tab of an Application Composition Report

The *License Analysis* tab displayed in Figure 7.35 displays all components and the determined details about their license(s).



The screenshot shows a web interface titled "My Application - 2015-05-13 - Build Report". It has four tabs: "Summary", "Policy", "Security Issues", and "License Analysis". The "License Analysis" tab is active. Below the tabs is a table with three columns: "License Threat", "Component", and "Status". The table contains 10 rows of data, each representing a different license threat for a specific component. The status for all listed threats is "Open".

License Threat	Component	Status
Apache-2.0, Non-Standard	com.fasterxml.jackson.core : jackson-core : 2.0.4	Open
Apache-2.0 or LGPL-2.1, Non-Standard	com.fasterxml.jackson.core : jackson-databind : 2.0.4	Open
Not Declared, CDDL-1.0	javax.transaction : jta : 1.1	Open
CDDL-1.0 or GPL-2.0-CPE, CDDL-1.1 or GPL-2.0-CF	javax.mail : mail : 1.4.2	Open
CDDL-1.0	jstl : jstl : 1.2	Open
CDDL-1.0	javax.activation : activation : 1.1	Open
LGPL-3.0, LGPL-2.1+, MIT	net.sourceforge.jtds : jtds : 1.2.2	Open
CPL-1.0, No Source License	junit : junit : 4.8.1	Open

Figure 7.35: License Analysis Tab of an Application Composition Report

In the *Policy*, the *Security Issues* as well as the *License Analysis* tabs, you can get access to more information about a particular component by clicking on a row in the table representing the component you are interested. The Component Information Panel CIP, with an example displayed in Figure 7.36 shows more specific information about the component.

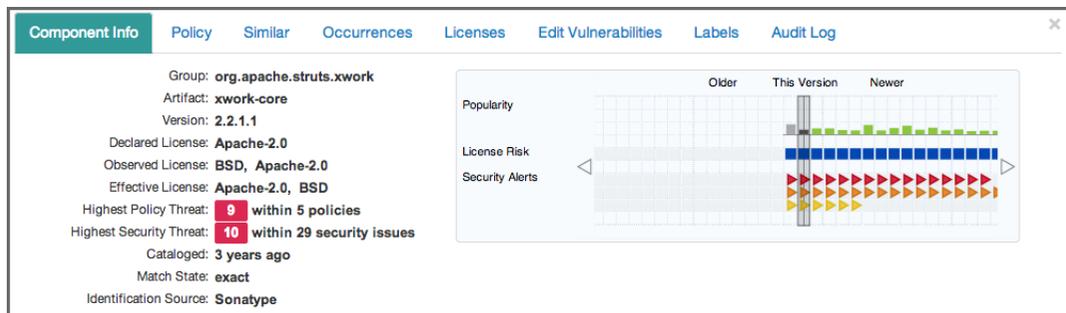


Figure 7.36: Component Information Panel CIP for a Specific Component

Clicking on the *Policy* header in the component information panel displays all policy violations for the selected component. As you can see from the example displayed in Figure 7.37 the policies as well as the constraints and the condition values that triggered the policy violation are displayed.

Policy/Action	Constraint	Condition Value	Waivers
Security-High	CVSS >=7 and <10	Found 15 Security Vulnerabilities with Severity >= 7 Found 27 Security Vulnerabilities with Severity < 10 Found 29 Security Vulnerabilities with Status OPEN	Waive
Security-Medium	CVSS >=4 and <7	Found 25 Security Vulnerabilities with Severity >= 4 Found 14 Security Vulnerabilities with Severity < 7 Found 29 Security Vulnerabilities with Status OPEN	Waive
Security-Unscored	CVSS = 0	Found 2 Security Vulnerabilities with Severity = 0 Found 29 Security Vulnerabilities with Status OPEN	Waive
Security-Low	CVSS < 4	Found 4 Security Vulnerabilities with Severity < 4 Found 29 Security Vulnerabilities with Status OPEN	Waive

Figure 7.37: Policy Section for a Specific Component Displayed on the Component Information Panel

A number of specifics used in the tabs and the panel are detailed in the following:

Threat Level

We briefly mentioned above, that policy violations are organized by threat level. The threat level breakdown is as follows.

- Red / High (10 - 8) - Indicates a component with a severe threat, and should be treated seriously.
- Orange / Medium (7 - 5) - Indicates a component with a moderate threat, and should be treated seriously.
- Yellow / Low (4 - 2) - Indicates a component with a low threat, and may not pose any serious threat to your application.
- Dark Blue / Informational (1) - Indicates that there is a very low threat, and you should just be aware of a possible issue.
- Light Blue / None (0) - Indicates that no policy has been violated by the component.

Matching

It's likely that you started seeing an area that indicates *matching*. As a quick definition, matching employs a series of in-depth algorithms to determine if a component found in your application matches anything known to the Central Repository, or known to the Sonatype CLM Server. That's right, through a claiming process and a proprietary component configuration, you can teach Sonatype CLM to recognize components it may not have otherwise.

PDF Printing

The application composition report can be printed to PDF simply by clicking the print icon located in the upper right corner of the report.

Re-evaluation

Eventually, when you begin to manage and modify policies, you may simply want to compare the results from the most recent report with your policy modifications. The re-evaluate button, located to the left of the pdf/print icon will allow you to refresh the results without having to generate a whole new report.

7.9 Importing Policies

Setting up policies can be quite complex and labor intensive. To make the process easier and give you a head start we have created some sample policies and provide an import feature.

We actually recommend you don't begin using Sonatype CLM by creating a bunch of policies right out of the gate. Instead, we've created a set of policies, which include other policy elements such as labels and license threat groups, that you can import into your Sonatype CLM installation.

Eventually, and there is a very short time between now and eventually, you will need to create, or at least modify, policies. For now, we'll want to focus on populating your organizations and applications with policies provided by Sonatype.

7.9.1 Sonatype Sample Policy Set

The easiest way to establish policies for your applications is to use the sample policy set provided by Sonatype. While the included policies are not meant to be a perfect match for every business, they have been created with our extensive experience working with customers and developing policy for our own internal practices.

Our sample policy set can be downloaded here:

[Sonatype-Sample-Policy-Set.json](#)

This policy set is an example of managing components for security, licensing, and architectural issues. It also introduces the detection of unknown and patched components used in building your applications. The sample policy set can be used to gather information about the components used to build your applications without warnings and failures occurring in the developer, build, or Nexus environments.

This is the perfect set of policies to use in order to gather information and understand how policy management will work for your environment, without potentially distracting the people who are building and delivering your applications.

Note

The sample policy set includes several, preset tags. The tags have been used in the Application Matching area for a number of the included policies. Policies using the tags will be indicated by a special tag icon. In order to utilize the policies, you must have applied the corresponding tag to your application(s). For more information on tags, please see the [Policy Elements](#) section of this chapter.

7.9.2 Importing a Policy to an Organization

Once you have acquired a policy file to import, you can follow these steps:

1. Log into your Sonatype CLM server with a user account that has proper permissions to import policy for a specific organization (at least a member of the owner group for the organization would be required).
2. Next, click the *Manage Applications and Organizations* icon  to access the **Application and Organization Management** area.
3. Click on *Organizations* in the left menu, and then click the organization you wish to import the policy to.
4. Click the *Import* button in the top right corner of the organization view displayed in Figure 7.38.
5. Click the *Choose File* button in the **Import Policy** dialog displayed in Figure 7.39 and select the policy JSON file in the file browser.
6. Click the *Import* button in the **Import Policy** dialog.
7. Confirm that the list of policies contains the imported policies.

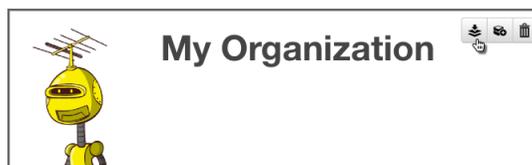


Figure 7.38: Organization View with Import Button

If you are importing to an organization, that already has some policies, labels, license threat groups, and/or tags set up, consider the following rules:

- Existing policies will be deleted during the import procedure.
- Importing policies also includes an import of associated policy elements (labels, license threat groups, and tags). The following logic will be used for Policy Elements:
 - Labels - the CLM server attempts to match labels against existing ones in a case-insensitive manner. This allows for updating the description or color of existing labels, while preserving any triage effort already done to apply these labels to components. If your import contains labels that aren't already present in the system, they will be created.
 - License Threat Groups - the CLM server will delete all existing license threat groups, and then import the new ones.
 - Tags - the CLM Server attempts to match tags against existing ones in a case-insensitive manner. This allows for updating the description or color of existing tags, while preserving any current matching of tags between policies and applications.

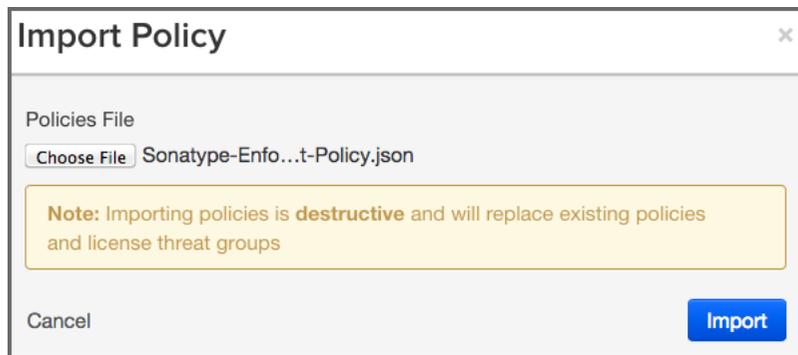


Figure 7.39: Import Policy Dialog

7.9.3 Importing a Policy to an Application

An application inherits policies from the organization. However it can be useful to have additional policies for fine grained control.

1. Log into the Sonatype CLM Server with a user that has been assigned to the CLM Administrator role, or an Owner role for the application you wish to import policy to.

2. Next, click the *Manage Applications and Organizations* icon  to access the **Application and Organization Management** area.
3. Two columns will be displayed on the left. Click on *Applications*, and then click the application you chose to import the policy to.
4. Click the *Import* button in the top right corner of the application view, which is identical to the organization view displayed in Figure 7.38.
5. Click the *Choose File* button in the **Import Policy** dialog displayed in Figure 7.39 and select the policy JSON file in the file browser.
6. Click the *Import* button in the **Import Policy** dialog.
7. Confirm that the list of policies contains the imported policies.

The policy information will be imported, and the following rules will be applied:

- Duplication of organization policies is invalid, so you will not be able to import the same policy file into an organization and then into an application associated to it.
- When a policy is imported, any existing application policies will be deleted and replaced with the imported configuration.
- For label imports, the same logic as during imports at the organization level described in Section 7.9.2 applies.
- Attempting to import policies that contain tags will cause the entire import to fail.

7.10 Policy Monitoring

At some point, your applications will be out of development, have completed their final build, moved beyond staging, and have been officially released. However, while there shouldn't be changes to your application that is now considered to be in production, new security vulnerabilities and license issues could arise. For this reason, as well as any other, Sonatype CLM allows you to monitor individual policies for each application.

When a policy is monitored, you pick an application, as well as a Sonatype CLM stage to monitor to use as a base for evaluating policy against. After that we'll show you how to configure which policies you would like to receive a notification for, given a component is found to be in violation.

If some of this sounds familiar, that's good, because it is nearly identical to standard policy evaluation, component violations, and the notification option for policies. There really is no difference other than being able to choose which Sonatype CLM stage you will use for monitoring. Though that is a powerful option.

In this section, we'll cover everything you need to setup policy monitoring at the organization and application level. In general, we make a few assumptions, including:

- You have your Sonatype CLM Server up and running, and accessible.
- You have created an organization and an application, as well as setup or imported some basic policies.
- You are somewhat familiar with the Sonatype CLM Server.

If any of these sounds like strange concepts, you'll want take a few steps back and go over those topics first. With that said, let's go monitor some policies.

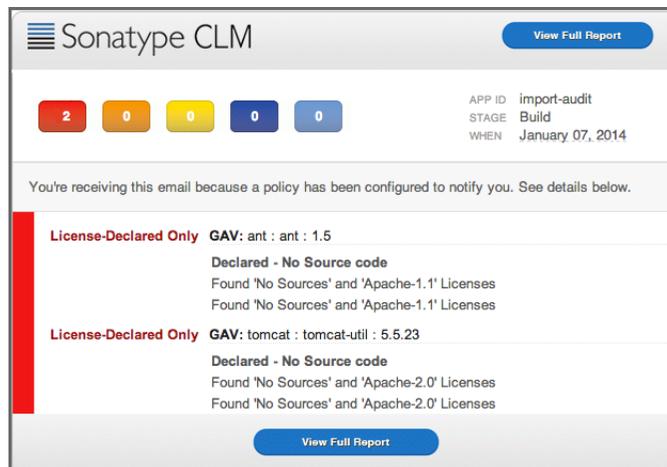


Figure 7.40: Example of a Policy Monitoring Email

Note

Policy Monitoring is not available to customers with a Nexus CLM License. Contact your administrator for additional information.

7.10.1 Setup Policy Monitoring for an Application

The setup for policy monitoring is straightforward. In general you will likely want to avoid monitoring every single policy. Not only is that a lot of policies to monitor, your signal to noise ratio will be off. That is, you might possibly get a lot of notifications for things like old component, or components that are now unpopular.

That's not to discourage you from monitoring these policies, they are important to. However, monitoring, and in turn, the notifications that are associated with monitoring are best reserved for policies that deal with elements like security vulnerabilities and license issues - that is, those representing a high level of risk.

First, choose an application to monitor

While you can choose any application to monitor, most people start by monitoring an application in production. In many cases production applications have likely been around longer than your implementation of Sonatype CLM. However, you are not prevented from choosing any application to monitor.

1. Log into your Sonatype CLM server with a user account that has proper permissions to make changes to an application policy (at least a member of the owner group for the application would be required).
2. Next, click the *Manage Applications and Organizations* icon  to access the **Application and Organization Management** area.
3. Click *Applications* in the menu on the left, and then click on the application you want to monitor.
4. The **Application Management** area will be displayed.

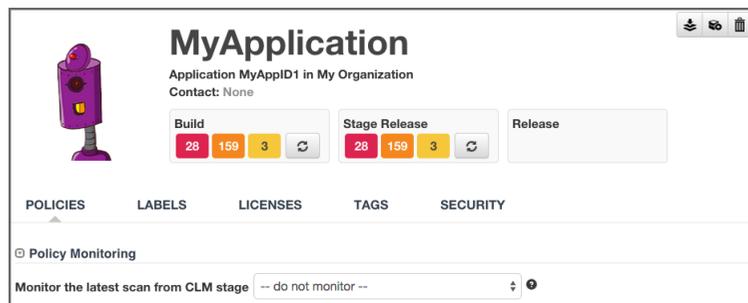


Figure 7.41: Access Application Management Area

Great, now you can move on to picking the stage you want to monitor.

Next, choose a Sonatype CLM Stage to monitor

Each of the Sonatype CLM Stages available for monitoring, are identical to the Sonatype CLM stages displayed when setting policy actions. In general, if you are going to be monitoring a production application, you will want to choose the stage that represents the most recent data. For our example, we'll choose Build, but again, you can choose any stage.

1. First, click on the *Policy Monitoring* section to reveal the options for selecting a stage to monitor.
2. Next, using the drop down menu, select the stage you wish to monitor. You will notice this drop down selection completes the sentence, " *Monitor the latest scan from CLM stage...* "

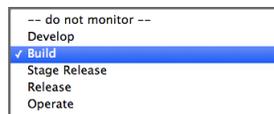


Figure 7.42: Selecting a Sonatype CLM Stage to Monitor

You are almost there, now all you need to do is add notifications.

Each policy you want to monitor will need to have someone added to the notifications. These are set below the **Actions** section of each policy.

1. From the *Application Management* area, click on the edit button of the policy you want to monitor.
2. Next, just below the **Actions** section, click the *Email* icon to open the Monitoring Notification dialog.
3. Enter an email address manually, and or select any roles you wish to be notified.
4. Click *Done*, and then *Save* to save your edits.

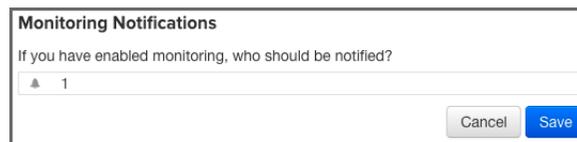


Figure 7.43: Adding Email Recipient

Tip

Remember, you can only edit a policy based on your permissions and where it was created, if you don't see the edit button for a policy, you either need to adjust your permissions, or switch to the organization the policy was inherited from.

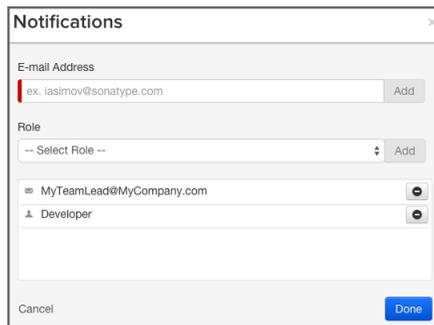


Figure 7.44: Policy Monitoring Notification Example

Congratulations! Your application now has a policy that will be monitored. To monitor more applications and/or policies, simply repeat the steps above.

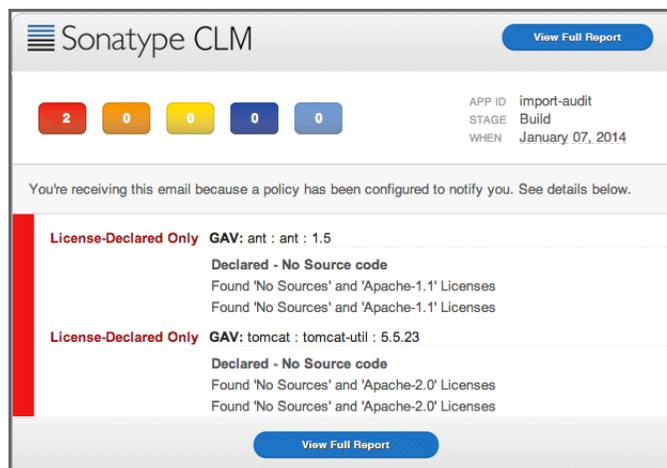


Figure 7.45: Sample Email Notification

Tip

While it is possible to follow these same steps and set policy monitoring at the organization level, you may want to think through that a bit more before blanketing all applications within a particular organization with policy monitoring notifications. In many cases, we find that monitoring is best done on a few, high risk, production applications.

7.10.2 Configuring Notification Times

By default any new notifications for policies that are being monitored will be sent out at 12 AM (per the CLM Server time). If you would like to update this, simply edit your config.yml file for the Sonatype CLM Server. The lines you will need to look for are as follows:

```
# Hour of the day(0-23) to schedule Policy Monitoring execution. The
default is midnight.
```

```
#policyMonitoringHour: 0
```

Chapter 8

Sonatype CLM - Dashboard

As you manage policy and evaluate applications, you are gathering a fair amount of data related to policy violations, security vulnerabilities and license issues. Ideally, you want to focus on areas that represent the highest risk, and resolve those quickly. This can be easy to do if you only have to work on a single application; however, the moment you have much more than that, the necessity for a more cumulative approach is needed.

The Sonatype CLM Dashboard provides the quickest way to review the overall health of the applications you manage. Whether you are looking to find the worst violations or the newest, the dashboard should be the first place you review each day.

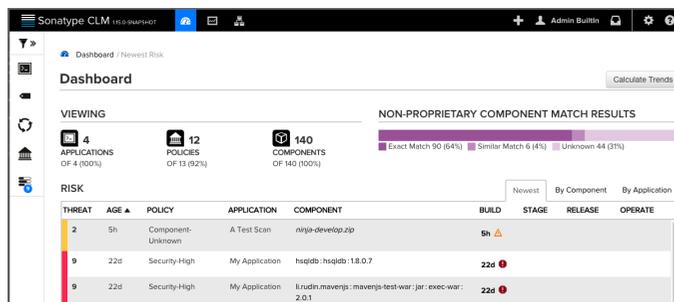


Figure 8.1: Dashboard Default View

**Important**

Users of Nexus CLM Edition do not have access to the Sonatype CLM Dashboard. Because of this, these users will not be taken to the dashboard after logging in, nor will they see the dashboard icon. Rather, the reports area will display by default.

8.1 Accessing the Dashboard

Once logged into the CLM Server, the Sonatype Dashboard will display by default. If you are in any other location of the CLM Server, simply click the Dashboard icon  located in the header.

Note

The dashboard is only available via the Sonatype CLM Server, and only displays information for applications you are permitted to see. This requires that you, at a minimum, be in the [Developer role](#) for at least one application.

8.2 Viewing CLM Data in the Dashboard

Data displayed in the dashboard is based primarily on violations found during the evaluations of your applications. It is organized into three distinct areas:

- Filters
 - Visual Overview
 - Highest Risk Violations
-

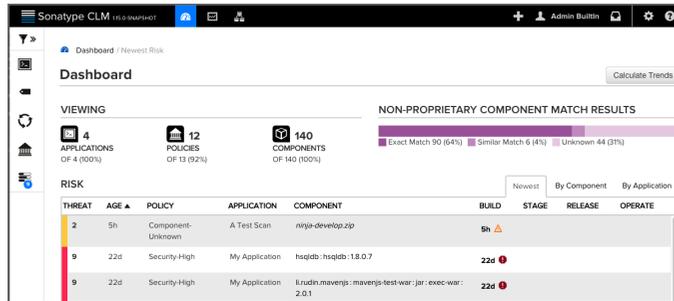


Figure 8.2: Accessing the Dashboard

Note

When navigating the dashboard, clicking on the breadcrumb link will return you to the most recent tab.

8.2.1 Filters

Filters allow you to adjust the data that is displayed in the dashboard. While this gives you greater control over what is viewed, in some cases this may limit the display of certain information.

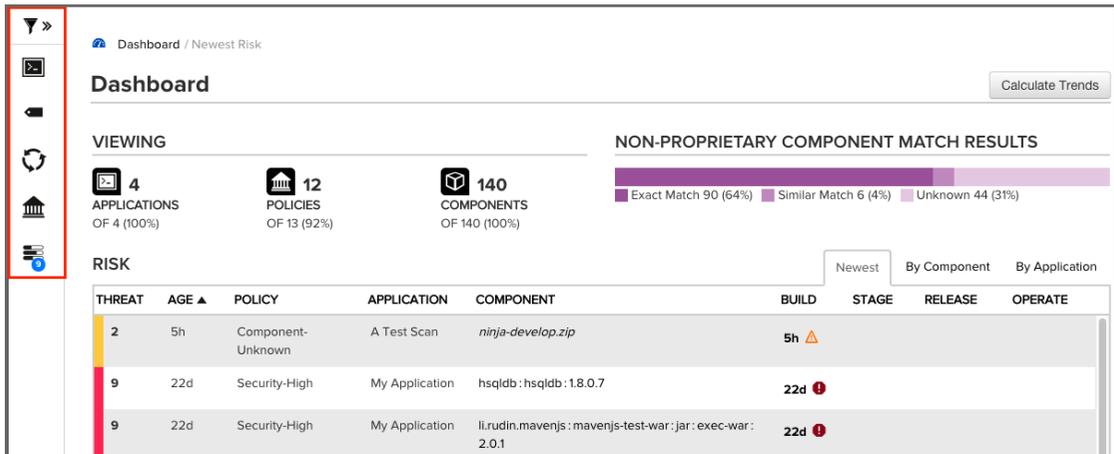


Figure 8.3: Dashboard Filter Example

This is most apparent with regard to the display of threat level ranges (Critical, Severe, Moderate, and Low). Based on what filters are set, any columns that display this data may be completely hidden from view.

For example, by default the threat level filter is set to exclude any violations of policies with a threat level less than or equal to 1. Given this, the low threat level column will not be displayed.

The *Filter* icon is located to the left of the Sonatype CLM screen, just below the Sonatype logo. To edit the various filters that are available, click on the chevron to the right of the *Filter* icon . This will cause the *Filter* menu to slide out to the right.

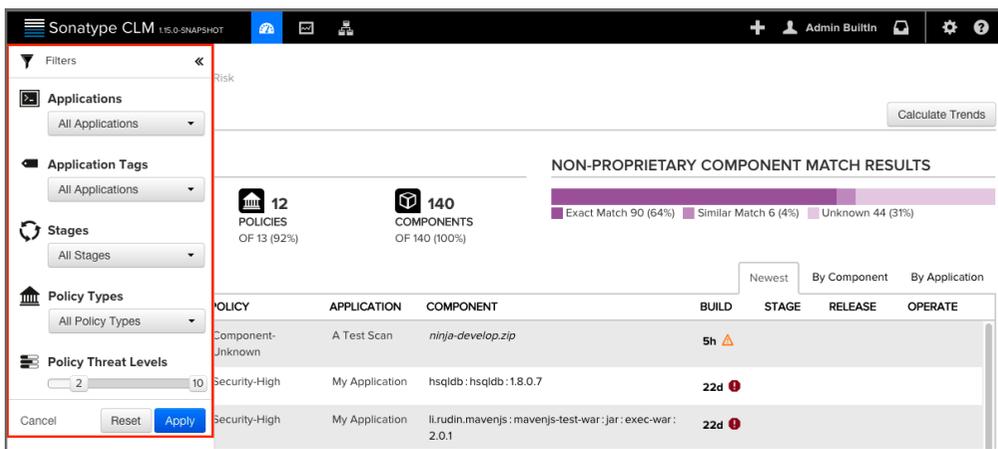
Note

To close the *Filter* menu, click the chevron next to the *Filter* icon.

With the *Filter* menu open, make selections using the five available filters, and then click the *Apply* button. Any filters that are not set to *All*, will have a blue circle, with the total count of selected filter options.

For example if you selected five applications, the *Applications* filter would have a blue circle with the number five. The same is true with all the filters, including threat level. In the case of the latter, the total number of threat levels selected in the filter, not the actual threat level, is displayed.

To reset the filters, click the *Reset* button, and then *Apply*.



POLICY	APPLICATION	COMPONENT	BUILD	STAGE	RELEASE	OPERATE
Component-Unknown	A Test Scan	ninja-develop.zip	5h			
Security-High	My Application	hsqldb:hsqldb:1.8.0.7	22d			
Security-High	My Application	il.rudin.mavenjs:mavenjs-test-war:jar:exec-war:2.0.1	22d			

Figure 8.4: Filtering the Dashboard

Tip

After exiting the Sonatype Dashboard area and/or logging out, your most recent filters will persist for your account when you return.

The available filters have been described below.

Applications

The application filter allows you to select which applications you want displayed in the violation lists.

Application Tags

The tag filter allows you to isolate violations for applications associated with a particular tag.

Policy Type

The policy type filter allows you to select which types of policies you want displayed in the violation lists. Sonatype CLM automatically assigns type based on conditions included within the policy. The following rules are used to determine a policy's type:

Security

if there are any security conditions, it is considered a security type policy.

License

if there are any license conditions, it is considered a license type policy.

Quality

if there are any age or popularity conditions, it is considered a quality type policy.

Other

if there are any conditions not mentioned above, it is considered an other type policy.

Note

A policy can only ever be of one type. In cases where a policy has conditions that meet more than one of the rules above, the order above dictates the type of policy. For example, if a policy has security and license conditions, it would be considered a security type of policy.

Stages

Violations can occur in different stages, and this will likely affect how much attention you decide to give at a particular point in time. Using this filter, you can show violations for a specific stage. The available stages include:

- Build
 - Stage Release
 - Release
-

- Operate

Note

Access to stages is limited by your product license, and the filters will reflect this. In addition, when specifying a stage with the filter, those not selected will be hidden from view.

Policy Threat Levels

The Policy Threat Level filter functions as a slider that allows you to select the threat level or a range of threat levels. This corresponds to the threat level of the policy that has been violated.

Note

By default, the Policy Threat Level filter has already been set to only display policy violations with a threat greater than or equal to 2. This means only those violations in the Critical, Severe and Moderate threat ranges will be displayed. As a result, the *Low* threat category column is hidden.

8.2.2 Visual Overview

Underneath the dashboard header there are two visual summaries of the data matching the current filter.



Figure 8.5: Dashboard Visuals

Viewing

While much of the dashboard focuses on policy violations, the information provided in the *Viewing* area covers all components. There is only one exception, proprietary components. That is, the match results will not include any components that are excluded as a result of your proprietary component settings.

The first display shows counts for the number of applications, policies, and components the data in the dashboard represents, identified by their corresponding icon and text label.

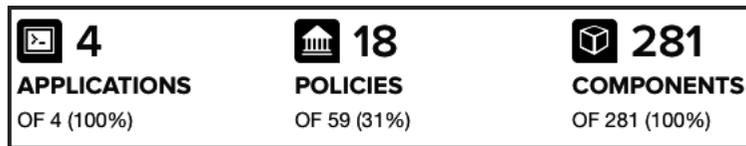


Figure 8.6: Counts

Note

In cases where data has been filtered, the counts may not represent all data. This will display as a percentage less than 100%.

The second displays the non-proprietary component matches.



Figure 8.7: Matches

When reviewing match data, it is important to remember the types of matches that can occur. It may also be a good idea to review the section of the Application Composition Report chapter [focused on Component Identification](#). A brief overview is included below.

Exact Match

Sonatype CLM has matched a component exactly to the one in your application.

Similar Match

Sonatype CLM has found at least one component that may match the component in your application.

Unknown

Sonatype CLM has been unable to identify the component in your application.

Note

In instances where an unknown or similar component has been claimed, it will be considered an exact match.

Policy Violation Trends

To the right of the dashboard header is a button titled *Calculate Trends*, clicking this will open a modal which will retrieve a view of trends for policy violations matching your current filter.

Note

This could take some time to calculate depending on the number and size of evaluations that match.

In contrast to the count and match data, the rest of the Sonatype CLM Dashboard, including the Policy Summary visualization, is geared towards identifying which components in your applications present risk so you can address them accordingly. This is because understanding how your business is handling risk over time is extremely important. As you are likely already asking, questions such as "How many new violations have been encountered, waived, or fixed," as well as "How many remain unresolved," are just the beginning.

Given this, the main goal of the Policy Summary visualization is to provide a quick, twelve-week look at how risk is entering your applications, and how you are handling that risk.

The Policy Summary area is divided into four categories, with each category having four metrics over a twelve-week period.

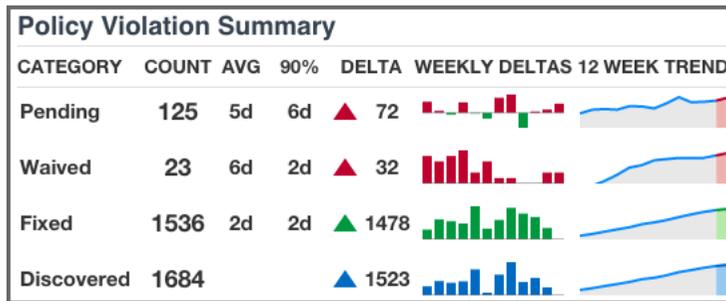


Figure 8.8: Policy Violation Trends

These categories and metrics have been described below:

Trend Categories

Pending

A policy violation that has been *Discovered*, but not yet *Fixed* or *Waived*, is *Pending*.

Tip

Reducing the number of pending violations is a critical task with Sonatype CLM. Weekly deltas above the x-axis indicate there were more discovered violations than those fixed; green bars below the x-axis represent more violations were fixed than discovered.

Waived

This represents a count of policy violations that have been waived. This count is not included in Pending or Fixed, but is included in Discovered.

Note

For more information on waivers, see the [waivers Section](#) of the Sonatype CLM Application Composition Report chapter.

Fixed

A policy violation is *Fixed* when it no longer exists in any Sonatype CLM stage.

Note

When determining the *Fixed* state of a component, any filtered stages are not considered. That is, if you exclude a stage where a violation has occurred, the count for fixed may increase even though the violation is still present in the other stage.

Discovered

A policy violation is considered *Discovered* when it has been observed for the first time.

Policy Summary Metrics**Count**

the total (all-time) count for the category.

AVG

the average age of violations in the category

90%

indicates 90 percent of violations have been in the category less than this time.

Delta

the count for the current week (week twelve), over the first week.

Weekly Deltas

the visual representation of each week's unique delta.

12 Week Trend

the trend over twelve weeks.

Tip

It is not uncommon to see discovered violations trend upwards steeply, especially in the early phases of your Sonatype CLM implementation, and then plateau as you start developing a better component consumption process. Using your mouse to hover over values in the graphs will display the individual values for each week.

8.3 Highest Risk Violations

The Highest Risk Violations display is separated into three different views/tabs.

- Newest
-

- By Component
- By Application

All risk information is based on the state the policy was in at the time of the most recent evaluation, while information regarding the age is taken from the first occurrence of the violation. If policy changes have been made, and a new evaluation has not been conducted, the changes will not be reflected in the currently displayed information.

8.3.1 Newest

This is the default view for the dashboard. It displays the first one hundred, newest component violations found in your applications. The data in this view can also be adjusted using the filters, and is organized into a number of columns. These have been described below.

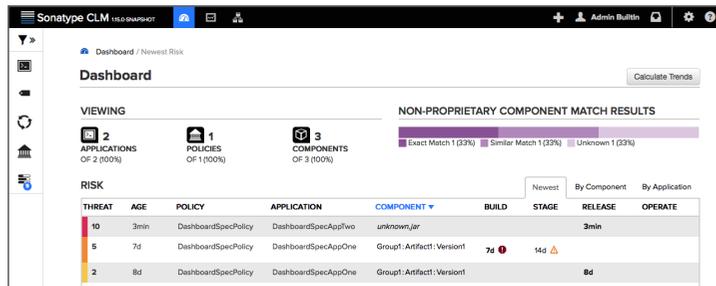


Figure 8.9: Newest Risk

Note

A violation is only considered new the first time it is discovered, even if it is found in different stages. For example, if a violation is found at the first of the month during an evaluation at the Build stage, and then again at the end of the month at the Release stage, only the occurrence at the build stage is considered new.

Threat

The assigned threat level of the violated policy.

Age

Displays the age of the violation based on the most recent date it occurred.

Policy

The name of the policy violated.

Application

The name of the application the component violating the policy was found in.

Component

The identifying information for a component. For known components, all available coordinate information will be displayed, while unknown components will have the filename. Clicking on the component will display the [Component Detail Page](#).

CLM Stages

The CLM stages follow the four stages that Sonatype CLM employs (Build, Stage (Stage Release), Release, Operate). The amount of time that has passed since discovery of the component in violation of a policy will be displayed in the corresponding column and row. Abbreviations for time is as follows:

- min = minute
- h = hour
- d = day
- m = month
- y = year

If any actions were taken in the stage (i.e. warn or fail), an icon will be displayed. Only the stages which your CLM server is licensed for will appear.

+ TIP: Clicking on the time stamp for the violation will open the most recent [Application Composition Report](#) for the corresponding component and application.

8.3.2 By Component

This view displays the first 100 highest risk components based on any filters that have been set and your level of access. Risk is represented in several ranges (Total, Critical, Severe, and Moderate), which corresponds to a color (Black, Red, Orange, Yellow). In addition, shading represents the severity of the risk within a particular column. That is, darker shading indicates the value is more severe relative to the other items in the column.

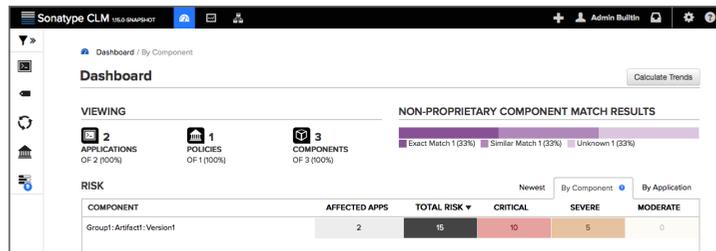


Figure 8.10: Highest Risk - By Component

Note

By default only policy violations greater than 1 (i.e. all but low/blue) are displayed and included in the calculations. Given that data excluded by filters is not displayed on the dashboard, the *Low* violations column will not be present. This can be modified by setting the Policy Threat Level filter to include violations below these levels (0/1).

To calculate the total risk for each component, the threat level of all policies the component has violated are added together. In other words, component risk is the sum of policy violation threat levels for the component. A similar calculation is done for each risk range.

Now, this may leave you wondering, "What about the duplication of violations across stages, or even in the same stage?"

Good question.

For all calculations, a violation is only counted once. When there are multiple instances of the same violation, only the most recent occurrence is counted, regardless of stage. Because of this, in cases where a policy has been changed in between evaluations, the violation from the latest evaluation will be included. This will be true, even if the change to the policy included threat level.

Now, let's take a look at each individual column, which has been described below.

Component

The identifying information for a component. For known components, all available coordinate information will be displayed, while unknown components will have the filename. Clicking on the component will display the [Component Detail Page](#).

Affected Apps

The sum of applications that are affected by a policy violation due to this component.

Tip

Clicking on this value will open the Component Detail Page.

Total Risk

The sum of the threat level for each policy the component has violated. In cases where the same violation is found in multiple stages, only the newest violation is included in this total.

Critical

The sum of the component's policy violations with a threat level of eight or higher.

Severe

The sum of the component's policy violations with a threat level higher than three, but less than eight.

Moderate

The sum of the component's policy violations with a threat level higher than one, but less than four.

Low

The sum of the component's policy violations with a threat level of one.

Tip

Remember, if your filters exclude data in any of these categories, this information will not be displayed.

8.3.3 By Application

This view displays the first 100 highest risk applications based on any filters that have been set, and your level of access.

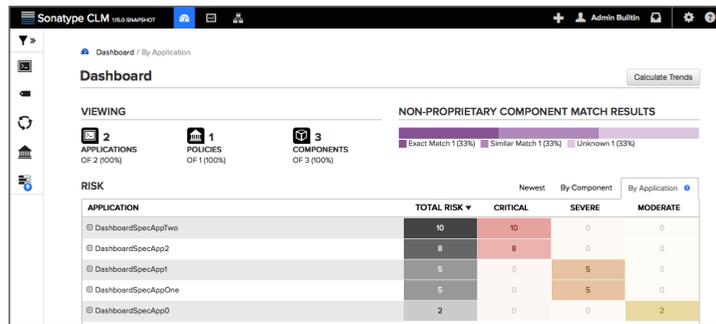


Figure 8.11: Highest Risk - By Application

Note

By default only policy violations greater than 1 (i.e. all but low/blue) are displayed and included in the calculations. Given that data excluded by filters is not displayed on the dashboard, the *Low* violations column will not be present. This can be modified by setting the Policy Threat Level filter to include violations below these levels (0/1).

Like a component, risk for an application is associated with the threat level of a policy. In the case of application risk, it is the sum of policy threat levels that correspond to unique policy violations for the components in an application.

This produces a total count by stage. The unique occurrences are then added together to create the total risk of an application. Put another way, application risk is the sum of all unique policy violation threat levels across all stages and policies the application is evaluated against.

Similar to the *By Component* view, for all calculations, a violation is only counted once. When there are multiple instances of the same violation, only the most recent violation is counted, regardless of stage. Because of this, in cases where a policy has been changed in between evaluations, only the violation from the most latest evaluation will be included. This will be true, even if the change to the policy included threat level.

Given the logic behind the calculation, risk is then broken down into five columns (six when low violations are included). Each application record can also be expanded to see the risk breakdown by stage.

Tip

Click on the stage name to see the most recent Application Composition Report for the corresponding application and stage.

For additional detail, take a look at the descriptions of each column below.

Application

The name of the application is displayed here. Click the expand icon (the small triangle icon), to display the results for each stage.

Total Risk

The sum of the threat levels for all policy violations in the application. In cases where the same violation is found in multiple stages, only one violation is included in this risk score.

Critical

The sum of policy violations in the application with a threat level of eight or higher.

Severe

The sum of policy violations in the application with a threat level higher than three, but less than eight.

Moderate

The sum of policy violations in the application with a threat level higher than one, but less than four.

Low

The sum of the component's policy violations with a threat level of one.

Tip

Remember, if your filters exclude data in any of these categories, this information will not be displayed.

8.4 Viewing Component Details

As components are used across various applications, and then evaluated with Sonatype CLM, it is very likely some of those components will violate your policies. When violations occur, this creates risk. The Component Detail page presents the known coordinates for the component and then below this,

all violations that have been found, organized by application. In addition, risk information for each component is provided.

Clicking on the icon to the top left of each application name will expand or collapse the detail for all policy violations related to the corresponding component and application.

commons-fileupload : commons-fileupload : 1.2.2						TOTAL RISK
APPLICATION	SHARE OF RISK	RISK	BUILD	STAGE	RELEASE	OPERATE
My Organization : MyApplication	2%	16	4m	4m		
9 Security-High	1%	9	4m	4m		
7 Security-Medium	1%	7	4m	4m		
My Organization 3 : My Application 4	2%	16	5m			
9 Security-High	1%	9	5m			
7 Security-Medium	1%	7	5m			
My Organization 4 : My Application 2	2%	16	3m	3m		
9 Security-High	1%	9	3m	3m		
7 Security-Medium	1%	7	3m	3m		

Figure 8.12: Component Detail Page

Similar to previous views, separate columns display pertinent information related to the component and violations associated with each application it is used in. These have been described in additional detail below.

Application

The name of application, preceded by its parent organization.

Share of Risk

The share of risk is displayed as a total for the application, as well as a breakdown for each violated policy.

For the Application

This is the percentage of risk for the displayed component in relation to a specific application. It is calculated by taking the sum of the threat levels for policies an application is evaluated against (and the component has violated), and then dividing by the sum of threat levels for all policies violated across all applications displayed.

For the Policy

This is the percentage of risk for a particular policy violation as it relates to the total risk for the component. It is calculated by taking the threat level of the violated policy, and dividing it by the sum of the threat levels for all violated policies for the displayed component and applications.

Risk

Risk represents the sum of the threat levels for the policies the component has violated.

CLM Stages

The CLM stages follow the four stages that Sonatype CLM employs (Build, Stage (Stage Release), Release, Operate). The amount of time that has passed since discovery of the component in violation of a policy will be displayed in the corresponding column and row. Abbreviations for time is as follows:

- min = minute
- h = hour
- d = day
- m = month
- y = year

In addition, if any actions were taken in the stage (i.e. warn or fail), an icon will be displayed.

Tip

Clicking on the time stamp for the violation will open the most recent [Application Composition Report](#) for the corresponding component and application.

Chapter 9

Sonatype CLM - Report

The Application Composition Report represents the health of your application. Ultimately, it serves as a snapshot, a point-in-time report representing risk associated with component usage for a specific application. The report includes information on how the application complies with the policies your team, or business, has established. In many ways, it's the final connector between policies and the components of your application.

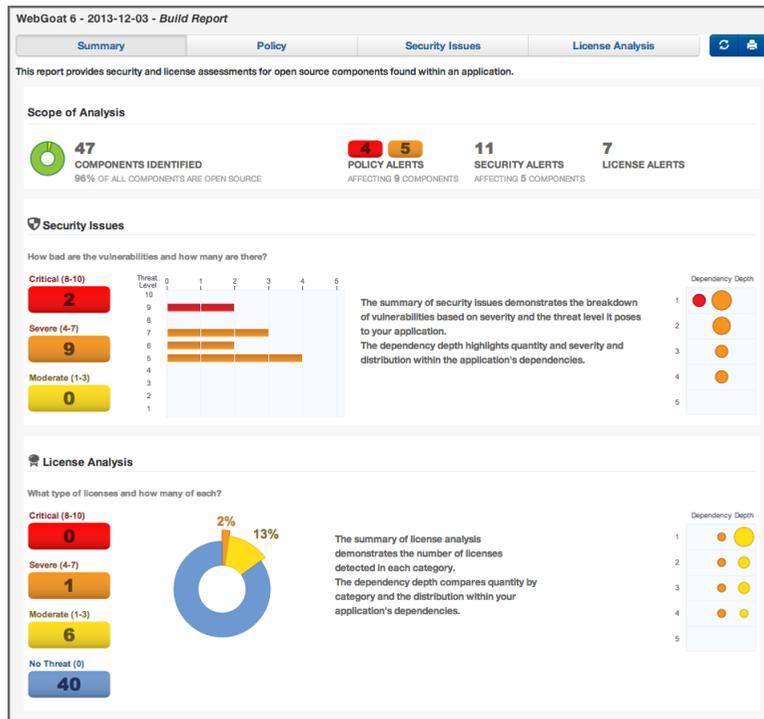


Figure 9.1: Summary Tab of the Application Composition Report

When looking at the report the first time, it can be daunting. If you see tons of red, you may quickly be dismayed. Or perhaps, you don't see enough red and are worried in a different way. These feelings aren't uncommon, and they reveal another important aspect of the Application Composition Report - it contains a lot of information.

More than just reporting the violations components in your application have triggered, it also provides a way to improve policy management. These reports don't show false positives... ever. If there is a red, severe policy violation that should really be much lower, communicate back with the team in charge of managing the policies. In fact, of all its uses, the ability to communicate findings to a wide audience is perhaps the most important task of this report.

In this section, we will provide an overview of the various areas of the report and therefore serve as an robust introduction.

For those of you that prefer bulleted lists, here's what we'll cover in this chapter:

- Accessing the application composition report
- Overview of the four tabs and the component list
- Importance of component and violation counts
- Various policy, security, and license related data points
- Printing a bill of materials
- Overview of component information panel (CIP)

This chapter is meant to provide a detailed look at how to access the application composition report, as well as what information is provided.

9.1 Accessing an Application Composition Report

You can access the application composition report in your Sonatype CLM Server in two ways:

Via the Reports Area

When you log into the Sonatype CLM Server the Dashboard is displayed by default. Click the *Reports* icon . If multiple applications have been scanned, you will see all of them here.

Note

You will need to be a member of at least the developer group for the application you wish to see a report for.



Important

Users of Nexus CLM Edition, do not have access to the Sonatype CLM Dashboard. Because of this, these will not be taken to the dashboard after logging in, nor will they see the dashboard icon. Rather, the reports area will display by default.

You will notice, that there are several columns:

Application Name

Links to the **Application Management Area** for the specific application.

Build, Stage Release, and Release Violations

These three columns display the violation counts for the most recent evaluations. The counts are broken down by *Critical*, *Severe*, and *Moderate* with text indicating the time (e.g. *2 minutes ago*) of the most recent evaluation.

Contact

This is the contact for the corresponding application.

Organization

Links to the parent organization for the corresponding application.

To access the Application Composition report, click the Violation Summary for the corresponding application and stage.

Application Name ▼	Build Violations	Stage Release Violations	Release Violations	Contact	Organization
MyApplication	28 159 3 1 minute ago	28 159 3 1 minute ago			My Organization
My Application 4			6 5 1 month ago		My Organization 3
My Application 3	6 11 1 5 months ago				My Organization 7
My Application 2	6 11 1 5 months ago	6 11 1 1 month ago	6 11 1 6 months ago	John Smith	My Organization 4

Figure 9.2: Reporting Area

Tip

By default this view will be sorted alphabetically by the application name. In addition to the filter, you can also click on the application or organization columns to sort alphabetically ascending/descending.

Via the Application Area

The Application area is the same place where you can manage policy for your application, reviewing policies unique to the application, as well as those inherited from the organization. Located just below the application identifier and organization, you will see three columns:

- Build
- Stage Release
- Release

These represent the Sonatype CLM stage where the report was generated for/from. For example, if you use the Sonatype CLM stand-alone scanner and don't specify the CLM Stage, it will default

to build. When your scan completes and the report is uploaded, it would appear below *Build*. This is highlighted in Figure 9.3



Figure 9.3: Application Area

Note

Reports can also be accessed via enforcement point tools like Sonatype CLM for CI, and Nexus Pro - Sonatype CLM Edition. However, in each of the tools, they will connect to the Sonatype CLM Server.

9.2 Reviewing a Report

When you look at the application composition report for the first time, you will likely notice the four tabs:

- *Summary*
- *Policy*
- *Security Issues*
- *License Analysis*



Figure 9.4: The Four Tabs

These tabs represent the basic navigation for the report, and serve to divide information into specific sections. In a sense, the name of each tab represents the theme of the data that will be displayed.

The *Summary* tab displays a summary of violation, security, and license risk information for components in your application and provides a good first overview. The *Policy* tab displays violation data for components in your application. The *Security* tab displays security related risk for components in your application. And the *License Analysis* tab displays license-related data for components in your application.

We'll cover each of these in a bit more detail below. However, it's important to first understand a little bit about what a report represents and the basic sets of data it contains.

In general, each report...

- Corresponds to a single, specific application, indicating the application name, date of the report, and the stage the scan took place in.
- Includes components found during a scan of the application, in most cases, including any dependencies.
- Records violations linked to an application's policies, or the policies inherited from the application's organization.
- Displays available security information for any components found matching components in the Central Repository.
- Displays available license information for any components found to exactly, or partially, match components in the Central Repository, as well as any data recorded manually (e.g. through the claiming process).
- Distinguishes between, external, proprietary and internally identified/claimed components.

Now that you know what forms the basis of the report, let's take a look at each tab individually.

9.2.1 Summary Tab

The *Summary* tab is always the first section of the report displayed. It is broken into three sections:

Scope of Analysis

This section shows counts, giving you an idea of the volume of components that were found during the scan. It also gives a breakdown of those that were identified, including a specific percentage that is represented by open source components. In addition to these numbers, you will also see:

- A count of components with policy violations, displayed by threat level. Only the most severe violation for each component is counted.
-

- The total number of security alerts found, and the number of affected components.
- The total number of license alerts. Each license alert corresponds to a single component.

Security Issues

The Security Issues section provides three visualizations. The first visualization displays the number of security issues by their particular Common Vulnerability Scoring System CVSS score, breaking the issues into three threat levels - *Critical*, *Severe* and *Moderate*.

Next to this raw count, the same numbers are represented in a bar graph to help distinguish the relative impact for each threat level.

Finally, a dependency depth chart shows where the security issues occur, relative to how many there are, indicated by the size of the circles, as well as what level of dependency they are found in.



Figure 9.5: Security Issues Summary

License Analysis

As with *Security*, the *License Analysis* section breaks the data into four threat level categories. However, these threat levels do not come from an external source, but rather the user-configurable license threat groups that are managed via the CLM Server.

There are four threat level categories:

- Critical (Copyleft)
- Severe (Non Standard)
- Moderate (Weak Copyleft)
- No Threat (Liberal)

These categories used in the report are static and not not configurable.

The first counts that are displayed represent the total number of licenses found in each threat level. Next to this list, a graph indicates percentage of licenses in each threat level category, compared to the total number of licenses found. Finally, a dependency depth chart indicates the volume of licenses found at each dependency level, as well as the color corresponding to the threat level.

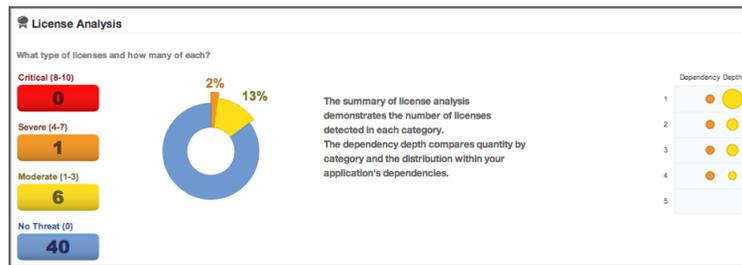


Figure 9.6: License Analysis Summary

9.2.2 Policy Tab

The *Policy* tab displays a list of all components found during the scan of the application. By default components are ordered by their worst policy violation. This is an important distinction, because a component may have more than one violation, and the threat level severity for those violations could vary. If you wish to see all violations there are two options, using the Violation Filter, or the Component Information Panel (CIP). In this chapter we'll discuss both options. However, below we have highlighted the available filters.

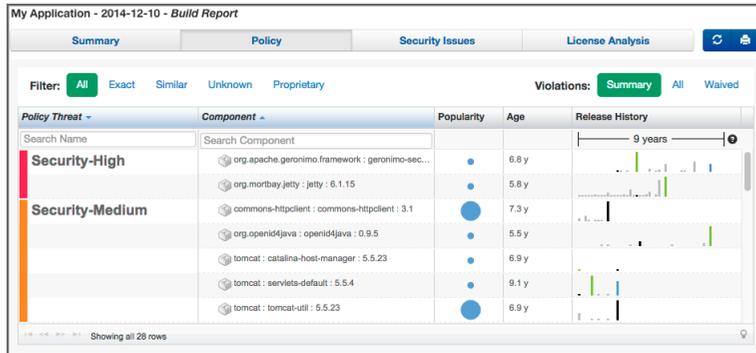


Figure 9.7: Policy Tab

Filter

The filter lists five categories:

- All (*default*)
- Exact
- Similar
- Unknown
- Proprietary

In addition to the main set of filters, you can also filter by violations, including those that have been waived. The available options include:

- Summary (*default*)
- All
- Waived

Clicking on any of these will change the components in the list. We'll discuss each of these in further detail in the sections corresponding to [component matching](#), [claiming components](#), and [waiving components](#) sections.

Component List

The list of components, below the filter, displays the *Threat level* posed by the components. The

Policy Threat column displays the name of the worst violated policy for the component and the severity using a colored bar. The *Component* column displays all available coordinate information for the component.

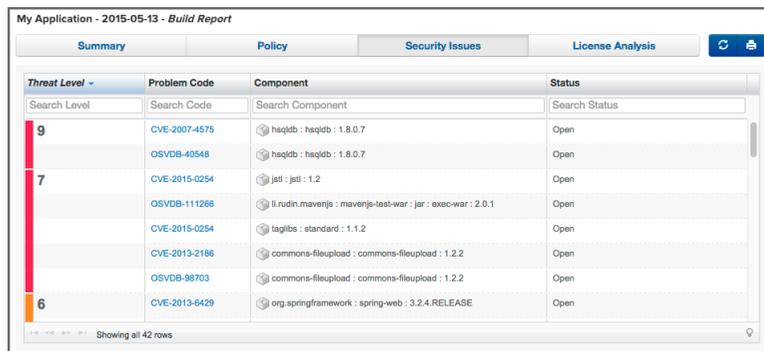
In addition the list displays the *Popularity* and the *Age* of the component in the Central Repository in separate columns. The *Release History* is displayed in a visualization that includes the most popular version, the most recent version, your version and any other available versions in a timeline.

By clicking on the column header, the list of components can be sorted. If you are looking for a specific policy, or component, you can use the search fields located at the top of each of those columns, directly below the header.

Clicking on a row for a component in list displays the Component Information Panel (CIP), which we will discuss in Section 9.4.

9.2.3 Security Issues Tab

The important thing to remember about the *Security Issues* tab is that information displayed there is related specifically to security vulnerabilities data that has been collected by Sonatype. This data however, is separate from policy violations, which are based on policies that you have created (or imported), and are displayed on the *Policy* tab. That is, you could certainly have a situation where there is a security vulnerability, and no policy violation. Because of this, it is important to treat them independently.



Threat Level	Problem Code	Component	Status
9	CVE-2007-4575	hsqldb : hsqldb : 1.8.0.7	Open
	OSVDB-40548	hsqldb : hsqldb : 1.8.0.7	Open
7	CVE-2015-0254	jstl : jstl : 1.2	Open
	OSVDB-111266	il.rudin.mavenjs : mavenjs-test-war : jar : exec-war : 2.0.1	Open
	CVE-2015-0254	taglibs : standard : 1.1.2	Open
	CVE-2013-2186	commons-fileupload : commons-fileupload : 1.2.2	Open
	OSVDB-98703	commons-fileupload : commons-fileupload : 1.2.2	Open
6	CVE-2013-6429	org.springframework : spring-web : 3.2.4.RELEASE	Open

Figure 9.8: Security Issues Tab

The way components are displayed is actually quite different as well. In the *Security Issues* tab, only those components with a security vulnerability are displayed. The data provided for each component is broken into several columns:

- Threat Level
- Problem Code
- Component
- Status

By default the list of components with security vulnerabilities is organized by threat level. This helps you isolate the most critical issues you need to address. However, you may notice that components in this list are repeated. This is because a component may have more than one security vulnerability, and those vulnerabilities in fact may have different scores, thus different threat levels.

To sort the list, simply click the corresponding header. For example, if we wanted to sort by components, finding a component with multiple vulnerabilities, we would simply click on the *Components* column. Additionally, you can search for a specific component by typing in the search field located directly below each header.

9.2.4 License Analysis Tab

The *License Analysis* tab displays all identified components found in the application scan and their license threat details. Unknown components are not displayed. Similar to the security issues, a license threat does not necessarily correlate to a policy, and as such should be treated independently.



License Threat	Component	Status
Apache-2.0, Non-Standard	com.fasterxml.jackson.core : jackson-core : 2.0.4	Open
Apache-2.0 or LGPL-2.1, Non-Standard	com.fasterxml.jackson.core : jackson-databind : 2.0.4	Open
Not Declared, CDDL-1.0	javax.transaction : ja : 1.1	Open
CDDL-1.0 or GPL-2.0-CPE, CDDL-1.1 or GPL-2.0-CF	javax.mail : mail : 1.4.2	Open
CDDL-1.0	jdk : jdk : 1.2	Open
CDDL-1.0	javax.activation : activation : 1.1	Open
LGPL-3.0, LGPL-2.1+, MIT	net.sourceforge.jids : jids : 1.2.2	Open
CPL-1.0, No Source License	JUnit : junit : 4.8.1	Open

Figure 9.9: License Analysis Tab

For each component listed, the license related data is displayed. This data is based on information collected during a scan. By default, components are listed based on the threat of the corresponding *License*

Threat Group that identified license is in. However, like the other tabs, clicking on a column in list will sort the components by that column. Additionally, specific components can be isolated using the search located below each header. The columns displayed include:

- License Threat
- Component
- Status

9.3 Printing and Reevaluating the Report

The top right corner of the report displays two buttons that give you access to refreshing the report as well as printing the report.

The refresh button on the left triggers a re-evaluation of the report. It will take the existing list of components in the report and reevaluate them against the your application policy. This comes in handy when you are making policy changes and want to see how that would affect the current data without having a to rerun a build.

The second icon on the right, the printer icon, allows you to create a PDF version of the report that is nearly identical to the HTML version. You can use this report for actual printing on paper, distribution to recipients without access to the CLM server or simply for archival purposes. And of course it also works as a great bill of materials for your application.

While these are both small elements, they can prove to be very useful.



Figure 9.10: Application Composition Report Buttons For Printing and Reevaluation

9.4 The Component Information Panel (CIP)

In our previous sections, we briefly indicated that clicking on a specific component causes the Component Information Panel (CIP) to be displayed. We promised to discuss it further, and this section makes good

on that promise.

The first thing you should notice, is that the CIP can be accessed for a component on the *Policy*, *Security Issues*, and *License Analysis* tabs. No matter which of these tabs you are on, simply click on the component, and the panel is displayed. Even better, the information displayed is the same, regardless of the tab in which you clicked on the component.

The CIP itself is divided into two areas. The top has a list of various sections, each providing more specific details and functionality related to the component. Below these sections, the panel will display information for the corresponding section. A brief description of each section is included below.

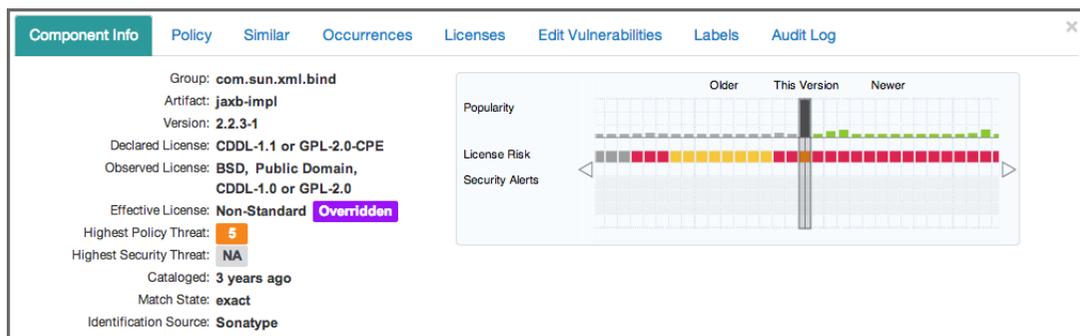


Figure 9.11: Component Information Panel CIP Example

COMPONENT INFO

Declared License

Any license that has been declared by the author.

Observed License

Any license(s) found during the scan of the component's source code.

Effective License

Either any licenses included in the Declared or Observed Group, or the overridden license.

Coordinates

The identifying information for a component. For known components, all available coordinate information will be displayed.

Highest Policy Threat

The highest threat level policy that has been violated, as well as the total number of violations.

Highest Security Threat

The highest threat level security issue and the total number of security issues.

Cataloged

The age of the component based on when it first was uploaded to the Central Repository.

Match State

How the component was matched (exact, similar, or unknown).

Identification Source

Whether a component is identified by Sonatype, or claimed during your own process.

Website

If available, an information icon providing a link to the project is displayed.

The graph itself is laid out like a grid, with each vertical piece representing a particular version. The selected version being identified by a vertical line. The information displayed in the graph includes:

Popularity

The popularity for each version is shown as a bar graph. the larger the graph the more popular the version.

License Risk

This will display the license risk based on the application that is selected, and the associated policy and/or license threat groups for that application. Use the application selector to change the application, and corresponding policies the component should be evaluated against.

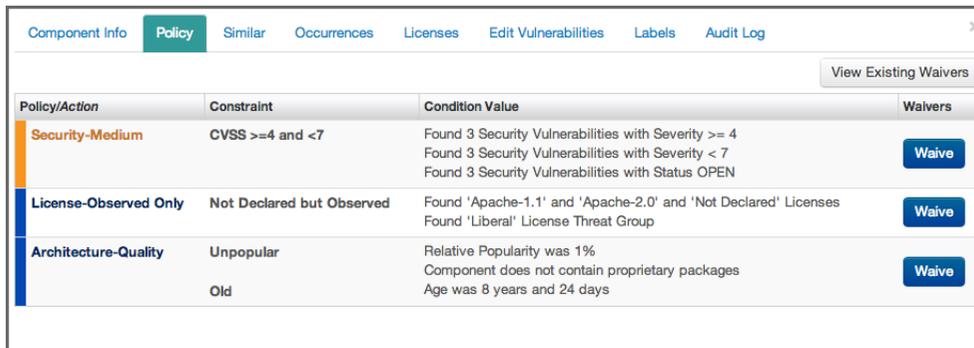
Security Alerts

For each version, the highest security threat will be displayed by color, with the highest shown as red, and no marker indicating no threat.

Policy

The *Policy* section displayed in Figure 9.12 has the details of any policy violations for the component. Here you can see the name of the policy that has been violated (and any action that was taken), the name of the constraint that has been violated, and the value that was found. While the *Policy/Action* and *Constraint* names are straight forward, the *Condition Value* may be a little confusing at first.

As we know from our other chapters, a condition is simply the *if* part of an *if/then* statement. *If* a certain condition value is found which is equivalent to a condition being met, *then* the policy will be violated. E.g. if we have a policy that has a condition such that if a security vulnerability is found, our Condition Value column would indicate, *Found x Security Vulnerabilities*. In the same regard, *Constraints* are simply multiple conditions joined together.



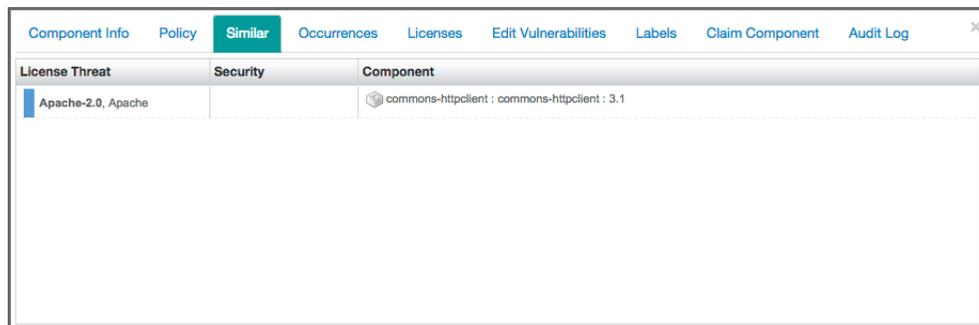
Policy/Action	Constraint	Condition Value	Waivers
Security-Medium	CVSS >=4 and <7	Found 3 Security Vulnerabilities with Severity >= 4 Found 3 Security Vulnerabilities with Severity < 7 Found 3 Security Vulnerabilities with Status OPEN	Waive
License-Observed Only	Not Declared but Observed	Found 'Apache-1.1' and 'Apache-2.0' and 'Not Declared' Licenses Found 'Liberal' License Threat Group	Waive
Architecture-Quality	Unpopular Old	Relative Popularity was 1% Component does not contain proprietary packages Age was 8 years and 24 days	Waive

Figure 9.12: CIP, Policy Section

In addition to simply viewing the policy information details, a policy violation can also be waived in this section of the CIP using the *Waive* button.

Similar

You likely have already noticed the *Similar* filter that is available on the *Policy* tab. These two are related, and both are a function of the matching algorithms that Sonatype CLM uses when scanning and identifying components. We won't go into the details of matching at this time. So, for now, know that any components found to be similar to the selected component will be listed in the *Similar* section displayed in Figure 9.13. A similar component could for example be a component that a developer has built locally using the source code of an open source component with minor modifications or additions.



License Threat	Security	Component
Apache-2.0, Apache		commons-httpclient : commons-httpclient : 3.1

Figure 9.13: CIP, Similar Section

Occurrences

When a file is scanned, it has a filename and location where it was found. In some cases, it may have more than one filename and location. Either way, the path to the location(s), as well as the filename(s), of the component that was scanned is included in this section. In short, the *Occurrences* section displayed in Figure 9.14 lists the file names and locations where the component was encountered. This section can be especially useful to detect accidental shipping of duplicate components archives or a misconfiguration of your actual report creation target e.g. you might be scanning the deployment archive (e.g. a war file) as well as the build output folder used to create the archive.

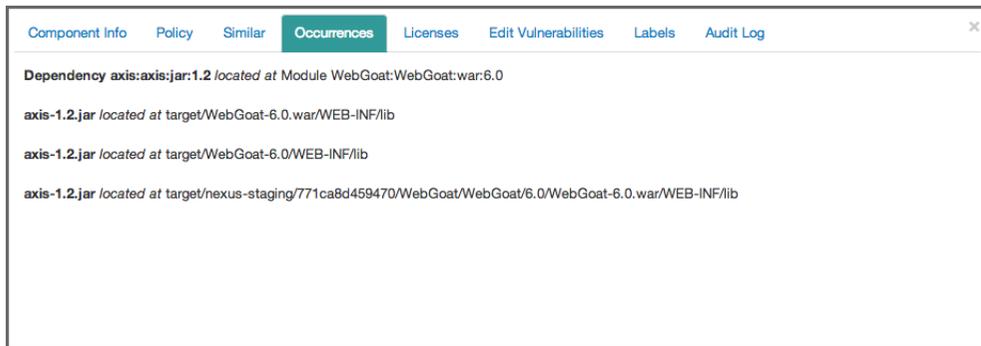


Figure 9.14: CIP, Occurrences Section

Licenses

The *Licenses* section displayed in Figure 9.15 is split into two areas. On the left, any licenses that were identified as declared by the author of the component, as well as any license found during the scan of the component source code are listed. On the right, is the license status area. This functionality directly correlates to the blue *Edit* button we mentioned in the *License Analysis* tab overview. It allows you to set the *Status* of the component license information.

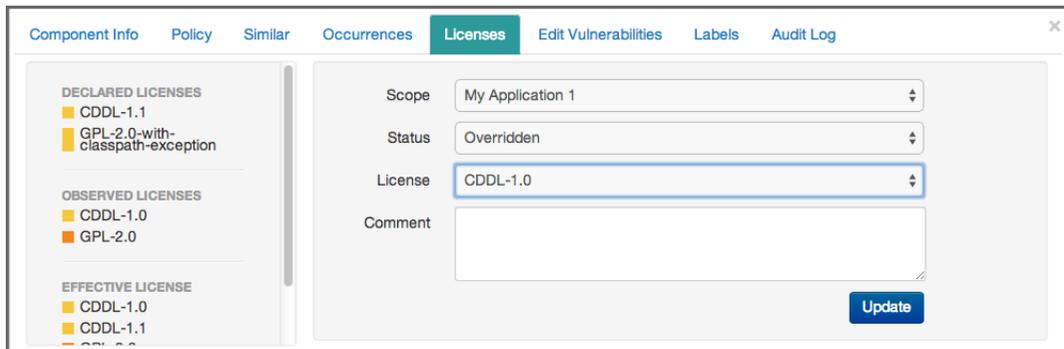


Figure 9.15: CIP, Licenses Section

Edit Vulnerabilities

In much the same way as the *Licenses* section, *Edit Vulnerabilities* displayed in Figure 9.16 is separated into two areas. On the left, all security vulnerabilities related to the component are displayed. On the right, the security vulnerability status area. This functionality, which we will discuss later, directly correlates to the blue *Edit* button we mentioned in the *Security Issues* tab.

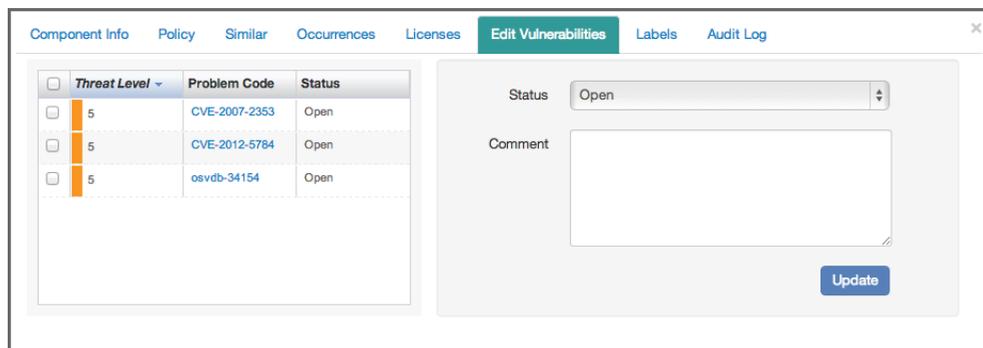


Figure 9.16: CIP, Edit Vulnerabilities Section

Labels

Labels are discussed in more depth later in this chapter. However, the important item to note here, is that the assignment of labels to a components is done in this section of the CIP displayed in Figure 9.17.

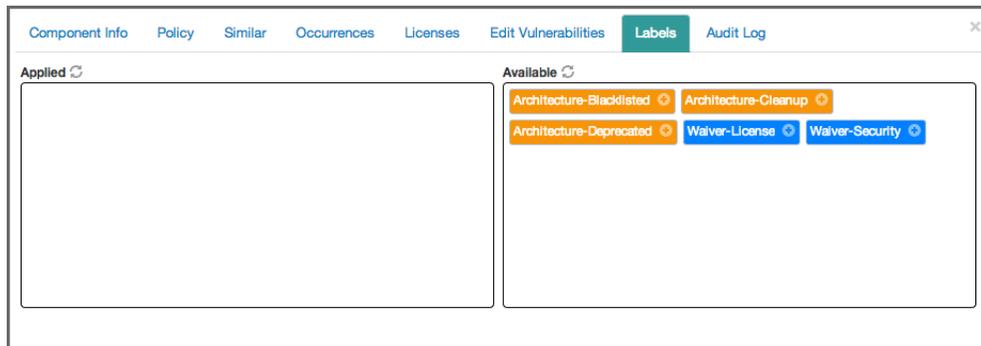


Figure 9.17: CIP, Labels Section

Claiming Components

The Claim Component section displayed in Figure 9.18 is only available for unknown or similar component matches. During a scan, some components are identified as unknown or similar to components known by Sonatype CLM. Since we realize that in many cases, you actually recognize these components, we provide this section to claim these components.

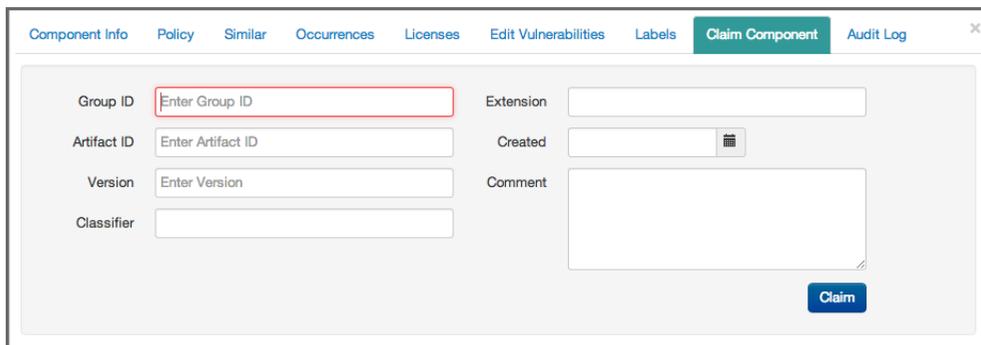
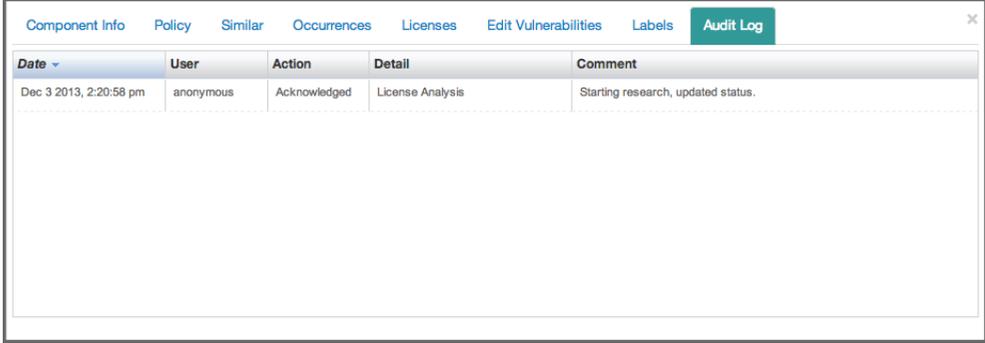


Figure 9.18: CIP, Claim Component

Audit Log

When changes are made to the status of a security vulnerability, or the status of a component's license within the scope of a particular application, that information is recorded in the *Audit Log* section of the CIP for that component displayed in Figure 9.19. As is the case for these last few sections, we'll discuss

the *Audit Log* in greater detail along with our upcoming discussion of *Security Vulnerability* and *License Analysis* status.



Date	User	Action	Detail	Comment
Dec 3 2013, 2:20:58 pm	anonymous	Acknowledged	License Analysis	Starting research, updated status.

Figure 9.19: CIP, Audit

9.5 Resolving Security Issues

Perhaps one of the most disconcerting experiences with Sonatype CLM, is scanning your application for the first time, and seeing a huge number of critical security vulnerabilities indicated on the *Summary* tab. It can be a sobering experience, and in some ways it should be a little worrisome. More importantly though, it should create motivation for further investigation.

The key word there being investigation. That's because even though we've provided accurate data, you still need to have a process to review all available data, and then track your progress. It is not completely uncommon, and quite possible that a vulnerability doesn't apply to your application or, at the very least, isn't a concern given the particular application you are developing, and it's relative exposure points. Where do you start your investigation though?

9.5.1 Security Issues

The component list on the *Security Issues* tab (see example displayed in Figure 9.20) only shows components that have a security vulnerability. In addition, when a component has multiple security vulnerabilities, it is displayed multiple times.

There are a total of four columns: *Threat Level*, *Problem Code*, *Component*, and *Status*. Initially the list

of vulnerabilities is ordered by the *Threat Level* column. However, you can sort the list by any other column by simply clicking on a header.

While the *Threat Level* and *Component* columns should be self-explanatory, the two other columns, *Problem Code* and *Status*, deserve a bit more explanation.



The screenshot shows the 'Security Issues' tab in the Sonatype CLM interface. The table displays a list of vulnerabilities with columns for Threat Level, Problem Code, Component, and Status. The Threat Level column is color-coded: red for 9, orange for 7, and yellow for 6. The Problem Code column contains links to CVE and OSVDB. The Component column shows the affected component and version. The Status column indicates the current state of the vulnerability.

Threat Level	Problem Code	Component	Status
9	CVE-2007-4575	hsqldb : hsqldb : 1.8.0.7	Open
	OSVDB-40548	hsqldb : hsqldb : 1.8.0.7	Open
7	CVE-2015-0254	jstl : jstl : 1.2	Open
	OSVDB-111266	il.rudin.mavenjs : mavenjs-test-war : jar : exec-war : 2.0.1	Open
	CVE-2015-0254	taglibs : standard : 1.1.2	Open
	CVE-2013-2186	commons-fileupload : commons-fileupload : 1.2.2	Open
	OSVDB-98703	commons-fileupload : commons-fileupload : 1.2.2	Open
6	CVE-2013-6429	org.springframework : spring-web : 3.2.4.RELEASE	Open

Figure 9.20: Security Issues Tab

Problem Code

The *Problem Code* column provides a link to available details for the security vulnerability on the CVE and OSVDB web sites. This information is provided via the CVE and OSVDB security information sites, and is managed independently of Sonatype CLM data. These public security databases allow you to get quick information about the security issue and nature of the vulnerability.

Status

The *Status* column allows you to track the state and progress of research of the effect of a security vulnerability with respect to your application. We'll focus on the *Status* column in a bit more detail when we cover the CIP. A key point to remember, is that as long as the status is set to *Open*, *Acknowledged*, or *Confirmed*, the vulnerability will be included in the counts on the summary page. In addition, a policy with a condition related to the presence of a security vulnerability will be met, as long as the status is set to *Open*. That means it's very important to research these issues, so that only those affecting your application remain.

9.5.2 The Component Information Panel (CIP)

To access the CIP as displayed in Figure 9.21, simply click on a component row in the list. There are three sections you should use during your security vulnerability investigation - *Component Info*, *Edit Vulnerabilities*, and *Audit Log*.

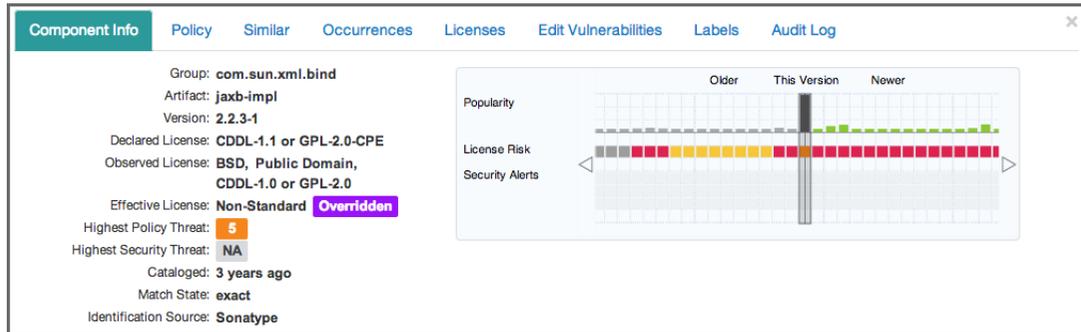


Figure 9.21: Component Information Panel (CIP)

Component Info

One of the first things you should notice in the *Component Info* section, is the *Highest Security Threat*. This field, located on the left side of the panel, displays the highest threat and the threat value (on a scale of 1-9). In addition, it will display the total number of security issues for that particular component.

Component Graph

Next, you should take a close look at the graph to the right of the panel. On the graph, locate the *Security Alerts* field, taking into consideration the other fields as well. This graph will display security vulnerabilities by version, with the current version identified as *This Version*. In some cases there are clear points where security issues have been resolved, as can be seen above. Often this tends to coincide with more popular version, although, that is not necessarily always the case.

9.5.3 Editing Vulnerability Status

After clicking on a component row to display the CIP, click the *Edit Vulnerabilities* section.

Here, the left side will display all security vulnerabilities. Depending on how many, this list may scroll. The list is then organized into three columns:

Threat Level

Indicates the threat assigned to the security vulnerability and is determined based on the source. This is **not** associated to any policy threat level.

Problem Code

This is the unique identifier of the security issue as assigned by the source (e.g. CVE-2000-5518). It will change depending on the source of the data.

Information

Sonatype provides information from public sources, as well as information from our own research team. Clicking on the icon in the corresponding row will display additional details provided about the issue.

+



Figure 9.22: Security Information Modal

Status

The status of the security issue as assigned by the drop down to the right. See below for information on changing this status.

To the right of the list of security vulnerabilities is the status drop down and a comments section. To change the status simply select one from the drop down, select the vulnerabilities the status will apply to, enter any associated comments, and finally, click the *Update* button. It is important to mention the status can be changed to any status at any time.

There are four statuses available:

Open

The default status, represents no research being done.

Acknowledged

Represents that the security vulnerability is under review.

Not Applicable

Indicates that research was conducted, and the particular vulnerability does not affect the application.

Confirmed

Demonstrates research was conducted, and it has been determined the security vulnerability is valid and applicable.

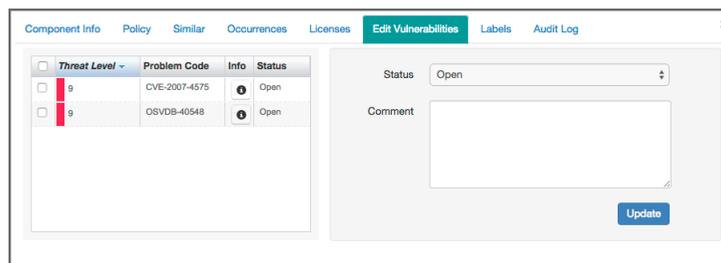


Figure 9.23: Editing Vulnerabilities

9.5.4 Matching to Violations

In some cases, just because there is a security vulnerability, that does not necessarily mean there is a corresponding policy violation. For this reason, it's important to refer back to your *Policy* tab as well. If you are finding that critical security issues you are troubleshooting do not show up as a policy violation as well, you may need to refine your policy so that future security issue trigger a policy violation and thus ensure that they get your attention.



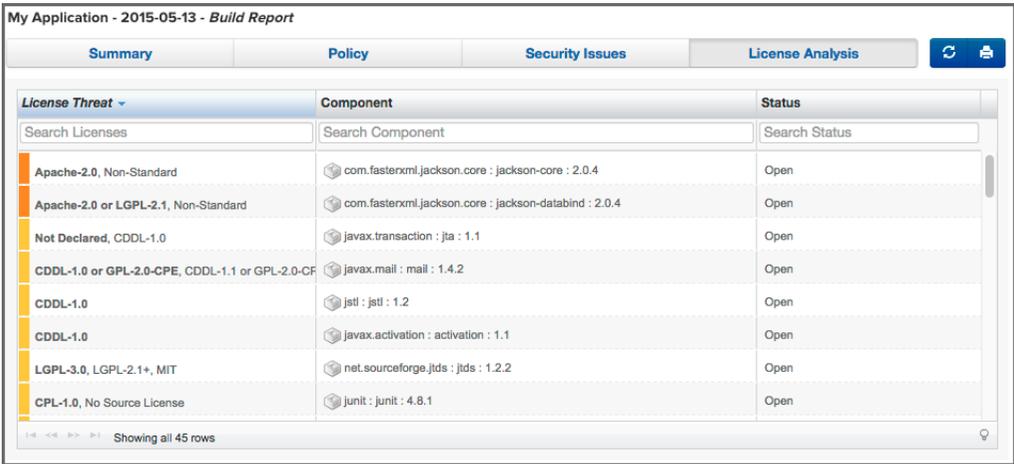
Figure 9.24: Example of Component with Security Issue, but No Policy Violation

9.6 License Analysis Tab

In some cases, the licenses of a component is the last thing a development team will think about. This could simply be due to a misunderstanding of *open source*, or a situation where it's nearly impossible to do the exhaustive research needed to determine the license for a given component, especially dependencies. With Sonatype CLM this problem is a thing of the past.

Even if you haven't built policies around licenses the *License Analysis* tab provides license information about every component found in during a scan of your application.

This license information is provided via data collected from the Central Repository, as well as research conducted by Sonatype. In addition to the license information for each component, we'll also assess a threat of each license, based on a set of default *License Threat Groups*. As with *Security Issues*, the best place to start is with the component list in the *License Analysis* tab, and then move into looking at additional details for individual components, making any license status changes as you see fit.



The screenshot shows the 'License Analysis' tab in the Sonatype CLM interface. The title bar reads 'My Application - 2015-05-13 - Build Report'. The interface has four tabs: 'Summary', 'Policy', 'Security Issues', and 'License Analysis'. The 'License Analysis' tab is active, showing a table with three columns: 'License Threat', 'Component', and 'Status'. There are search filters for 'Search Licenses', 'Search Component', and 'Search Status'. The table lists several components with their respective license threat groups and status.

License Threat	Component	Status
Apache-2.0, Non-Standard	com.fastxml.jackson.core : jackson-core : 2.0.4	Open
Apache-2.0 or LGPL-2.1, Non-Standard	com.fastxml.jackson.core : jackson-databind : 2.0.4	Open
Not Declared, CDDL-1.0	javax.transaction : jta : 1.1	Open
CDDL-1.0 or GPL-2.0-CPE, CDDL-1.1 or GPL-2.0-CF	javax.mail : mail : 1.4.2	Open
CDDL-1.0	jstl : jstl : 1.2	Open
CDDL-1.0	javax.activation : activation : 1.1	Open
LGPL-3.0, LGPL-2.1+, MIT	net.sourceforge.jtids : jtids : 1.2.2	Open
CPL-1.0, No Source License	junit : junit : 4.8.1	Open

Showing all 45 rows

Figure 9.25: License Analysis Tab

9.6.1 License Threat Group

License threat groups are based on what is configured for each organization or application. Additional information can be found in the [License Threat Groups section](#) of the Policy Management chapter.

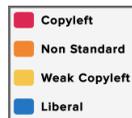


Figure 9.26: The Default License Threat Groups

Tip

How you manage your license threat groups directly impacts how threat is translated in the reports.

9.6.2 License Analysis

The component list on the *License Analysis* tab is more similar to the list on the *Policy* tab, because it is a list of all components, not just those that have a license issue.

The list itself includes columns for *License Threat*, *Component*, and *Status* of the license issue. Clicking on the column provides sorting, while specific items can be searched using the field just below the column heading.

License Threat

The list of components is ordered by license threat which is based on the threats assigned to the license threat groups. Though a single component may actually have several licenses, license threat will only show the highest threat. This threat, as we mentioned earlier, is based on four default categories, which correspond to four default license threat groups of the same name.

- Critical
- Severe
- Moderate
- No Threat

Status

License status, like status for security vulnerabilities, allows you to track the process for license related research. In addition it provides a way to override a license in situation where you believe the license to be incorrect, or there is an option to choose a specific license. We'll discuss that process a little bit further down.

9.6.3 The Component Information Panel (CIP)

To access the CIP for a component on the *License Analysis* tab, simply click on the component row. It will expand providing details in a number of sections. You will likely notice this looks the same as other CIP panels when clicking on other tabs of the application composition report, and you would be correct. There is nothing additional provided by accessing the CIP via the *License Analysis* tab of the report. However, for this section, we want to focus on the license related information in the Component Info section, as well as the entire *Edit Licenses* and *Audit* sections.



Figure 9.27: Component Information Panel (CIP)

Component Info

Again, the information contained here would be the same, whether or not you clicked on the component in the *License Analysis* tab. However, this gives us the context to talk about the License related fields in this section.

License Identification Types

On the left side of the Component Info section, you should pay attention to three fields, which are described below.

Declared License

these are the licenses that the developer of the component has identified.

Observed License

these are the licenses that have been observed during Sonatype's research.

Effective License

The effective license displays license information based on one of two scenarios. In cases where multiple licenses are found, including any that are observed, these will all be included as effective. If a license is selected, or overridden, then that license will be considered effective, and listed here.

License Identification Values

In cases where there is no declared and/or observed licenses, a message will be displayed. There are several options, each with specific meaning:

No Source License

sources were provided, but there was no license data found.

No Sources

indicates we have no sources for the component.

Not Declared

indicates nothing was declared by the author/developer.

Not Provided

will appear when the license is actually null, and is unique to claimed components, but might also happen while new components are being processed by Sonatype.

Component Graph

The graph itself is laid out like a grid, with each vertical piece representing a particular version. The selected version being identified by a vertical line.

While the information displayed in the graph includes popularity, and security information, right now, just take a look at License Risk. This will display the license risk based on the application that is selected, and the associated policy and/or license threat groups for that application. Use the application selector to change the application, and corresponding policies the component should be evaluated against.

9.6.4 Editing License Status and Information

Editing a license can be used for different purposes. One addresses the workflow of your research into a license related issue, while the other allows you to completely override a license all together. We'll cover all this below, but first let's take a look at the information displayed.

After clicking on a component in the list, and then the *Licenses* section of the CIP, the left side of the CIP displays the license(s) declared by the developer of the component, those that have been observed, and what is considered effective (a combination of the previous two). That is, unless they have been manually overridden or a specific license has been selected.

Next to each of these licenses is a box, displaying the severity of the license. This list can get long, so you may have to scroll to see all the licenses. Then, to the right of the license list, there are four drop down lists.

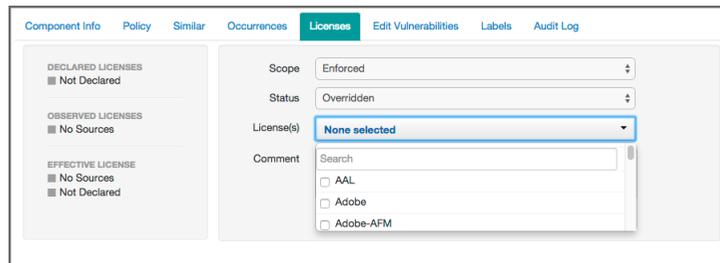


Figure 9.28: Editing License Using the Select Option

Scope

Scope allows you to apply the license status to this component by choosing application or to all components attached to the current application's organization by choosing organization.

Status

As we mentioned previously, *Status* provides a way to track your research, override a license, or select from an option. The available options are included below.

Open

This is default status, and will be included in the count of license issues.

Acknowledged

Acknowledged indicates the issue is being researched, and will still be included in the count of license issues.

Overridden:

This status will allow you to select one or more licenses from the *License(s)* dropdown (located just below the *Status* dropdown). This will override any licenses that have been declared or observed.

Selected

In cases where there are multiple licenses, this option will populate the *License(s)* dropdown with any licenses found in the component, declared or observed. Multiple licenses can be selected.

Confirmed

Confirmed simply indicates that the license(s) found are indeed correct, and will be included in any count of license issues.

License(s)

The *License(s)* drop down only displays given that a status of selected or confirmed has been chosen. Given that it will present either a list of all licenses (if override is chosen) or only the declared and observed licenses (if selected is chosen). The license that is chosen will be displayed in the *Effective License* field in the *Component Info* section of the CIP. In addition, any overridden/selected license will be indicated with a label of same name, next to the license in this field.

Comment

A comment is not required, but is a good element to include whenever you are making changes to the *License Status*. This is because it provides a way to understand, as well as audit, the decisions made to change a license status. This comment will be included with the record in the *Audit Log* section of the CIP.

Once you have made all your selections, and entered any necessary comments, click the *Update* button to save the *License Status* change.

9.7 Component Identification

One of the most important things you can do with regards to understanding the components in your application, is to identify them. What remains unidentified is of obvious concern.

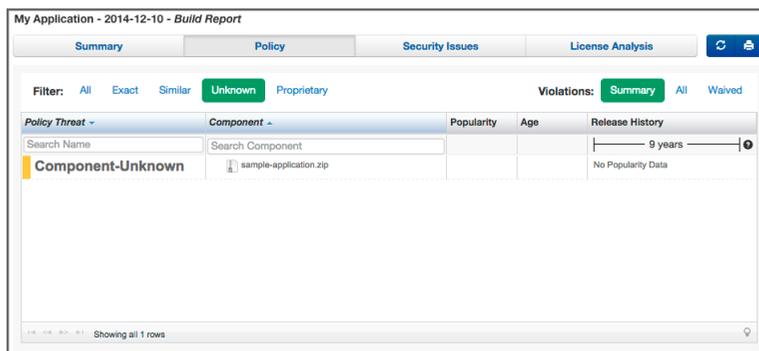


Figure 9.29: Unknown Component

Sonatype CLM allows you to identify components in a number of ways, including:

- Extensive matching via Sonatype CLM algorithms
- Claiming components
- Establishing proprietary components

In this section, we'll describe all of these in detail, within the context of identifying components using the application composition report, as well as offer our suggestion for best practices.

9.7.1 Matching Components

When a Sonatype CLM analysis is performed, hashes of the components in your application are created. This in many ways is like a fingerprint, which is unique to a component. We then compare that fingerprint (hash), back to our database of components. This database includes general component info, usage statistics, security vulnerability, and license information.

All of this information can be used as parameters in your policy, which translates to more understanding of the component usage in your organization. That data however, can only be linked based on a matching of hashes, which can be exact or similar, and in some cases, unknown. We discuss these three match types below.



Figure 9.30: Filter and Matching Options

Exact

An exact match means that a one-to-one link was found between a component hash in your application, and our database. All the data you see represented corresponds to a component we have identified and collect information for. This is the best case scenario with regard to component identification, and most components should fit in this category.

Similar

A similar match is found using proprietary matching algorithms, and is our best guess for a component that you have in your application. In some cases, multiple matches may be found, and this is where the *Similar* section of the CIP is important. While the most likely match is used to display any information about a similar matched component, you can see all other matches in this section of the application composition report. An example is displayed in Figure 9.13.

Unknown

There are instances where not even a similar component match can be determined. This should be considered a serious situation, at least one that needs to be investigated. This could be a case of a component being recompiled and modified so that a match to our database is no longer possible.

However, there is a chance that component is something malicious introduced into the application. Either way, an unknown component is one that Sonatype CLM has no information for. Of course, if during your investigation, you are able to identify the component, you can claim that component, via the *Claim Components* section, which we will walk you through in more detail a little bit later. An example is displayed in Figure 9.29.

Note

Unknown components will not be displayed in the *License* tab until they have been claimed.

In addition to the main filters above, you can also control whether all violations for each component will be displayed. By default the summary of violations is shown. This means that only the worst violation for a component will be shown, and the component will only appear once in the list. Choosing All or Waived, will show all violations (including those waived), or only the waived violations, respectively.

Note

Changing the Violations filter can result in the components being displayed in the component list more than once.

9.7.2 Managing Proprietary Components

Simply put, proprietary components are those components that are unique to your organization. In many cases these are actually developed by your organization and distributed among the applications you develop.

As with matched components, proprietary is one of the options included in the *Filter* on the *Policy* tab of the application composition report. Unfortunately, there is often a little bit of confusion around identifying a proprietary component, so let's start first with what a proprietary component is.

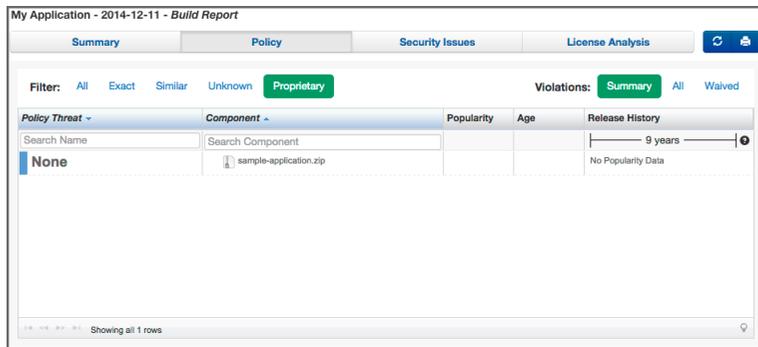


Figure 9.31: Proprietary Component

First and foremost, it's important to point out that a component identified as proprietary is not "matched" as proprietary.

Identifying a component as proprietary is separate from our matching process. In this, a component identified as proprietary will also be assigned to one of three match states: similar, exact, or unknown. In most cases a proprietary component is unknown. As a suggestion for figuring out which of your components are proprietary, a good place to start is by reviewing unknown components.

Note

Policy can be set in such a way as to exclude components marked as proprietary from triggering violations. Care should be taken in using that condition.

To set up proprietary identification, follow these steps:

1. Log into the Sonatype CLM Server with a user that has been assigned to the CLM Administrator role.
2. Click the *System Preferences* icon , and then the *Proprietary Components* option.
3. Sonatype CLM provides two methods for identifying proprietary components.
 - a. **Method 1:** Add packages that are considered proprietary. For example, if we entered `com.sonatype`, all components that contain a package `com/sonatype` would be marked as a proprietary component, and therefore not evaluated. Care should be taken to be as specific as possible here, as the provided package is compared greedily against your scanned binaries. For instance, if you specify `com.sonatype` it will match all of the following content locations:
 - `com/sonatype`
 - `com/sonatype/anything`
 - `com/sonatype/anything/more`
 - `shaded/and/relocated/com/sonatype`
 - `shaded/and/relocated/com/sonatype/anything`On the other hand, the following locations would not be matched for our example:
 - `org/sonatype`
 - `com/sonatypestuff`
 - `com/sonatypestuff/anything`
 - b. **Method 2:** Enter a regular expression, which will be compared against the paths of all files scanned. This is provided as a means for recognizing components as proprietary based on the existence of a specific file within them.

If you choose this option, make sure to click the *Regular JAVA Expression (RegEx)* check box. For more information on regular expressions, check out [Oracle's Java documentation](#).

An example of a regular expression might be `test\.zip`. In this example anything in the top level directory named `test.zip` would be marked as a proprietary component. If you wanted to apply this to match a file named `test.zip` nested anywhere within the scanned binaries, use `.*\/test\.zip`.

Note

Occurrences inside the identified archive will make the binary proprietary as well. For example, if a proprietary zip is found inside a jar, the jar is also considered proprietary.

4. After entering your proprietary component identification, click the *Add* button. This will queue your new proprietary component identifier for saving. Additionally, click any remove icon (resembles a minus symbol) in the list to remove an entry. No changes will be persisted to the server until you click the *Save* button.

Proprietary Components

These entries are provided to CLM scanners and are used to help identify internal components. If a specified package or file is contained within a component, it will be considered proprietary.

Enter Regex or Package Prefix for Proprietary Components Add

Regular expression

org.owasp.webgoat	<input type="button" value="✖"/>
com.sonatype	<input type="button" value="✖"/>
.*data\.zip	(regex) <input type="button" value="✖"/>
test\.zip	(regex) <input type="button" value="✖"/>

Reset Save

Figure 9.32: Proprietary Packages Configuration via the Sonatype CLM Server

Once your proprietary components are configured Sonatype CLM will look at the component and the directory structure of the application being evaluated. If it matches your proprietary component configuration, it will be identified as proprietary and displayed accordingly in the reports.

Remember, proprietary is not a type of match. Most proprietary components will still be identified as unknown. That's not a hard fast rule, but it is the most common case.

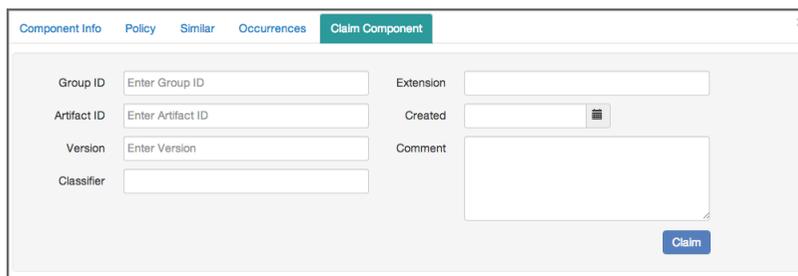
Tip

The proprietary component changes will not be evaluated against existing reports, but will be picked up on the next evaluation.

9.7.3 Claiming a Component

When a component is similar or unknown, yet you are certain the component is recognized by your organization, Sonatype CLM includes functionality to prevent that component from being identified as similar or unknown in future reports. In other words, Sonatype CLM allows you to claim the component as your own.

Once claimed, that component will be known to the CLM server. It will no longer be treated it as *Similar* or *Unknown*, and instead result in an *Exact*.



The screenshot shows a web application window titled "Claim Component". The window has a tabbed interface with tabs for "Component Info", "Policy", "Similar", "Occurrences", and "Claim Component". The "Claim Component" tab is selected. The form contains the following fields:

- Group ID: Enter Group ID
- Artifact ID: Enter Artifact ID
- Version: Enter Version
- Classifier: [Empty text box]
- Extension: [Empty text box]
- Created: [Empty text box] with a calendar icon
- Comment: [Large empty text area]

A blue "Claim" button is located at the bottom right of the form.

Figure 9.33: Claim a Component

1. Access an application composition report.
2. Click the *Policy* tab, and then click the *Unknown* or *Similar* component filter.
3. Click the row of component you wish to claim in the list - the Component Information Panel is displayed.
4. Click on the *Claim Component* section of the CIP .
5. Enter values for the coordinates of the component.
6. As an option, enter the coordinates classifier and extension, the *Created Date*, and/or a *Comment*. The created date is initialized with the date of the youngest entry in the component to be claimed.
7. Click the *Claim* button, to officially stake your claim for the component.

On review of the existing report, as well as those in the future, there is now an indicator that information about the component has been edited. When hovered over, a tooltip is displayed identifying that the component has been claimed.

We refer to this as the edited component tick mark (a small red triangle) on all future scans for this application, as well as any application with a valid Application ID on the CLM Server.

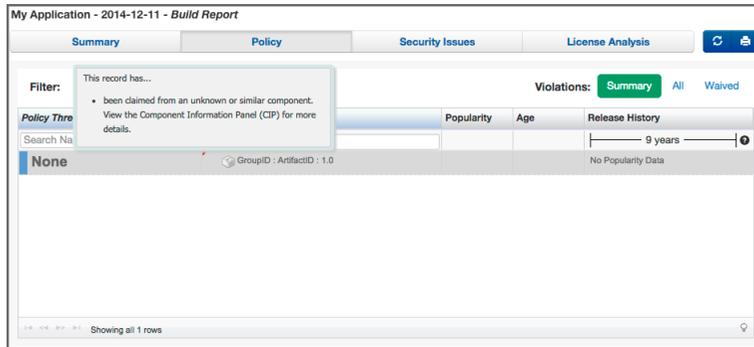


Figure 9.34: Claimed Component Indicator

In addition, the *Component Info* section for the claimed component will now have two new fields, one indicating the *Identification Source* is *Manual*, and the other, *Identification Comment* will include any comments that were entered. While any policy violations will be displayed, the component graph will not.

Finally, if you have made a mistake and wish to revoke the claim on the component or make an edit, click on the Claim Component tab. Then, use the *Revoke* or *Update* buttons respectively.

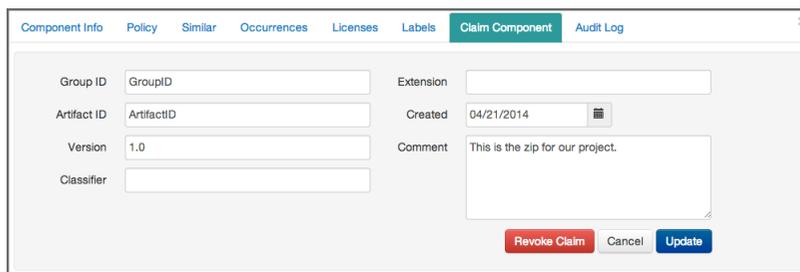


Figure 9.35: Update or Revoke Claimed Component Indicator

Tip

Use the cancel button to undo any changes you made but haven't saved.

9.8 Label Overview

Labels are one of the more powerful features of Sonatype CLM, though they operate similar to other label or tagging systems you have likely used. Basically you create a number of labels or tags as a set of available values and then assign them.

For example in a photo collection you could have labels for the content like *sunset*, *mountain*, *ocean*, *waterfall* and so on. An individual photo could then have multiple of these labels assigned. In a similar way you can use labels in Sonatype CLM to identify a particular type of component. An *approved* component or a component that *needs research*. Labels can be anything you desire and can have a number of contexts as well. Some labels might be architecture related, while others are related to legal and security properties and yet others are simply signaling ownership by a specific team. The flexibility of the label system allows you to design your own use cases and implement them.

Label creation and management is performed in the Sonatype CLM server organization and application configuration. Assigning those labels to a particular component is a function of the application composition report.



Figure 9.36: Labels at the CLM Server Level

9.8.1 Where do labels begin?

Initially, label assignment might seem like a task that deserves less process. It's actually the opposite.

The label process actually starts at the Sonatype CLM server, where each organization, and in many cases application, have a specific set of labels.

Creating labels at this level means that they are available to users of the application composition report. So, before you label something, a number of things need to be considered, which might mean you need to go back to label and policy management. Let's take a look at some questions that we can use to form a baseline to what labels we need and/or should have.

Do we have a process defining labels, as well as how and when they should be used? Every label should have a reason for using it. If it doesn't you don't need it. That's because people will naturally infer meaning to the label. For example, if you have a *needs review* label, you should have a process for reviewing components with that label. It shouldn't merely be a tag. You should also consider adding labels such as reviewed, so someone can tell if something was reviewed. The possibilities are endless, but this may mean you need to rethink the labels you have, or should create.

Should this label apply to only my applications, or all applications for this organization? We like to refer to this as the scope of a label. A good way to look at scope is the concept of macro and micro. At the macro level we have an organization that may have thousands of applications. If I apply this label to the component at the macro level (organization) it means the label will be seen by all applications under that organization. That's a pretty sweeping change, and it could be the right thing to do. However, you might actually be better served by considering the impact at the micro level. In this case, maybe this is a label that should only be applied to a particular application.

Note

Only labels that have been created at the organization level, can be assigned to the scope of organization.

If I apply this label, is it part of a policy, and does it escape certain violations? Conditions in a policy can include values for labels. In many cases, this is best used as a way to prevent a certain component from violating the condition. For example, I could have a policy that requires a component to be no older than three years. However, a safe, and commonly used component is four years old. If I have a process built around reviewing a case like this, where an exception would be valid, I could first place labels to identify the component to be reviewed for an exception, and then another label once that exception is approved (or disapproved). In this scenario, if I have built policies correctly, including allowing them to flow through certain stages, even with a violation, development can continue. This of course should occur simultaneously to a review/exception process. It's important to consider scenarios like this when creating labels, as well as policy.

There are more questions regarding labels that should be asked as well, but many of these you will discover as you develop your own processes. The key is that labels can be deceptively simple, given their implementation in other systems. The reality is, that in Sonatype CLM, they are much more than just a component tag. Now that you have the word of warning, let's take a look at how to assign a label.

9.8.2 Assigning a Label

When assigning a label, you will only have access to those labels created specifically for the application, or for that application's organization. Given this, if you don't see a label you need, speak with whomever is responsible for managing the labels for your application and the organization the application is associated to.

1. Access the Application Composition Report for your application.
2. Click on the *Policy* tab.
3. Click a component you wish to assign a label to. The Component Information Panel (CIP) displays.
4. Click the *Label* option from the CIP menu. Two boxes will be displayed:
 - a. The *Applied* box on the left represents labels that have been assigned to the component already.
 - b. The *Available* box on the right displays all labels.
5. Clicking on the button on the right side of a label will move to the opposite side. Hovering over a label displays the description.
6. Click on the + button on the right side of a label in the *Available* list to assign the label to the component.
7. Click on the - button on the right side of a label in the *Applied* list to remove the label from the component.

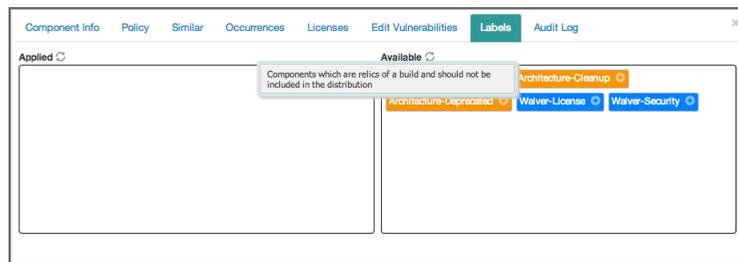


Figure 9.37: Assigning a Label

If the label was created at the organization level, you will be presented with two options:

- Apply the label to the current component in the current application
- Apply the label to the current component within the organization, so that all applications within the organization gain access to the label assigned.

9.9 Waivers

If you look at policy violations as a pain point preempting the flow of work, you are likely going about using Sonatype CLM the wrong way. In fact, if you saw the title of this section, hoping to find a way past policy violations, there may be a couple issues.

First, your policies should be designed to encourage workflow and communication. If development is being stopped regularly, you might want to revisit your policies, refining them so they present the possibilities for making better choices, not simply halting work altogether.

Second, and perhaps most importantly, Sonatype CLM does not present false positives. If you are looking for ways just to get past a violation, you've circumvented the goal of policy creation as well as Sonatype CLM. Again, this might be a problem with policy, or the perception of what should and should not be in your application.

OK, so excluding those possibilities, and working with the idea that you are here to find a way to accommodate the exceptions you may run across, waivers can help.

9.9.1 A Use Case for Waivers

Let's say, you have a component that's violating a policy, which has been created in an organization that houses your application. It's a great policy, in fact, one of many great policies. Unfortunately, one of your applications has a component that is violating a policy.

The problem is, that this policy just doesn't accurately line up to the implementation of this particular application. Your application has a security vulnerability that can be exploited when it connects to the internet. It's a pretty severe vulnerability, but benign given that your application is internal, and doesn't even have the ability to connect to the Internet.

What should you do?

You could...

- Change the *Security Status* to *Not Applicable*.
- Adjust the policy to be less stringent.
- Use labels in a conditions, providing an escape for exceptions.
- ... or, you could add a waiver.

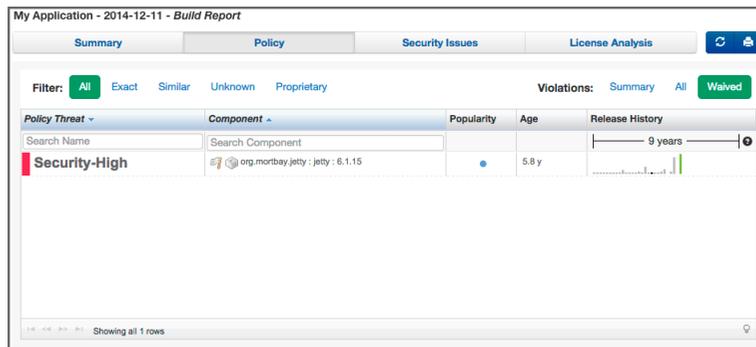


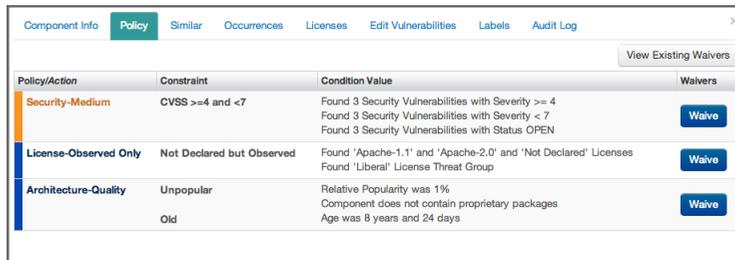
Figure 9.38: Waiver Visualization on Policy Tab

That is, by adding a waiver, you indicate that this particular component, either in the scope of this application (what we would do in our example), or all applications for the organization, is waived from this particular policy. In fact, if you desired you could even specify that you want to waive all components (within scope of an application or organization) from a policy.

Now the important thing to take note of here, is that while this waiver seems to be the answer to policy violations strife, you are actually waiving an entire policy. This means all constraints, and in turn, all conditions. It's no surprise why that should be something that's limited. For this reason, before waiving something, it's good practice to review some alternatives. A waiver very much does allow you to simply bypass all controls.

OK, we've harped enough on the warnings. The benefits of waivers can be just as numerous. This is because even with endless customizations, you will encounter situations where a policy just doesn't apply. It's important to take a look at the full range of waiver functionality, so let's look at how to add, view, and when necessary, remove a waiver.

9.9.2 Adding a Waiver



The screenshot shows a web application interface with a navigation bar at the top containing tabs: Component Info, Policy (selected), Similar, Occurrences, Licenses, Edit Vulnerabilities, Labels, and Audit Log. Below the navigation bar is a 'View Existing Waivers' button. The main content area displays a table with the following data:

Policy/Action	Constraint	Condition Value	Waivers
Security-Medium	CVSS >=4 and <7	Found 3 Security Vulnerabilities with Severity >= 4 Found 3 Security Vulnerabilities with Severity < 7 Found 3 Security Vulnerabilities with Status OPEN	Waive
License-Observed Only	Not Declared but Observed	Found 'Apache-1.1' and 'Apache-2.0' and 'Not Declared' Licenses Found 'Liberal' License Threat Group	Waive
Architecture-Quality	Unpopular	Relative Popularity was 1% Component does not contain proprietary packages	Waive
	Old	Age was 8 years and 24 days	Waive

Figure 9.39: Waiver Button

1. Access an application composition report.
2. Navigate to the *Policy* tab on the report, and click on a component that has policy violations. This will display the Component Information Panel (CIP).
3. Click the *Policy* tab. This will display the list of Policy Violations for the Component visible in Figure 9.39.
4. Click the *Waive* button next to the violation you wish to waive. A modal dialog similar to Figure 9.40 will display.
5. There are several options at this point, and each should be carefully considered:
 - a. The first option defines the scope for the waiver. This can be either the current application, or all applications for the organization.
 - b. The second option defines the target of the waiver. That is the currently selected component, or all components.
6. Enter an optional *Comment*, and then click the *Yes* button to process the waiver.



Warning

When processing a waiver, depending on the options that are chosen, you can effectively waive a policy for all components, for all applications in an organization. Since this will waive the entire policy, not just this violation, it may be a good idea to ensure adjusting the policy would not provide a solution that is more visible to all users.



Figure 9.40: Options to Apply Waiver to the Application or the Entire Organization

9.9.3 Viewing and Removing a Waiver

As we mentioned previously, component violations can be waived for a single component in a single application, all the way up to all components in all applications. This means, that a violation for a component in your application could have been waived elsewhere. A good practice when reviewing the Application Composition Report is to check and see what violations have been waived for components in your application. Here are a couple examples of why this is important:

- Scenario 1: A violation for a component has been waived, and the component has additional violations. Depending on the view selected, at least one of these additional violations will be displayed.
- Scenario 2: The only violation for a component has been waived. Given that the component has no additional violations, it will be moved into the **None** policy threat group (light blue) in the Summary view, while the other views will only show the waived violation.

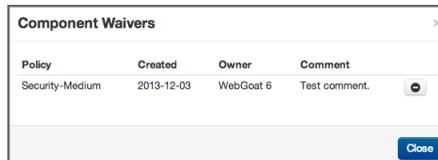
To view waived violations for your components, follow the instruction below.

1. First, access an application composition report.
2. Navigate to the *Policy* tab on the report. Just above the list of components, and to the right of the report, you will see three options in the Violations filter:
 - a. Summary - this is the default view of the Policy tab. It is important to note, that even though this view will display all components, only the highest threat violation per component is displayed. In this view, components with waived violations may have been moved to the **None** policy threat group (light blue).

- b. All - clicking this filter option will display every violation for all components in your application. This may result in the appearance of duplicates in the component list. Violations that have been waived will be indicated by a white flag icon.
 - c. Waived - clicking this filter option will display only the waived violations. In this view, you will only see those components where violations have been waived. Each component will have a white flag icon, and it is likely you will not see all components. This view may also produce the appearance of duplicated components.
3. Click on a component to display the Component Information Panel (CIP). For an example, see Figure 9.39.
 4. At the top of the of the component list, click on the *View Existing Waivers* button. A modal will be displayed showing all the waivers for the component, as well as the associated descriptions.
 5. Click the remove icon, which resembles a minus sign.
 6. A message will ask you to confirm this removal. Click the *Remove* button to continue.

Note

Because some waivers can be set for all applications, and even all components, it's important to understand the impact of removing a waiver. Be sure to verify with the application or organization owner, the intended scope of the waiver.



Policy	Created	Owner	Comment
Security-Medium	2013-12-03	WebGoat 6	Test comment.

Figure 9.41: View and Remove Waivers

9.10 Policy Reevaluation

You will likely find a number of consistent themes through the Sonatype CLM documentation. One of these is that regular policy review and refinement should be part of your companies approach to component lifecycle management.

Accomplishing this successfully could potentially mean regularly rebuilding applications or publishing them to repositories several times over. Not to mention that in the case of waiting for builds, you might wait hours before a scan is able to run.

This isn't an issue linked to Sonatype CLM, but rather the length of time it takes to build your application. No matter the reason, it really means access to the new results could be delayed, and the change you made to policy or statuses might not have even made a difference. You'll soon need to make another change, and then wait again. Luckily there is an alternative provided by Sonatype CLM. It allows you to reevaluate the results of a scan in the form of an application composition reports, which will use the existing component information from a scan and evaluate it against the current policies - which you might have changed since the last build and analysis.

To address this, you can use policy reevaluation to see how your changes affect the current policy. The policy reevaluation button, locate in the top right of the application composition report (to the left of the PDF Export/Printer icon). Simply click this button displayed in Figure 9.42, and any policy changes you've made will be considered against the data of the current report.



Figure 9.42: Application Composition Report Buttons For Printing and Reevaluation

Alternatively you can reevaluate policies right from the application configuration screen in the Sonatype CLM server. Simply find your application, and locate the stages report you want to re-evaluate under the application name beside the icon. Any stage that had a report processed will have a reevaluation icon right beside the stage name.

Of course, it's possible other data in the application could have changed, and that might not be realized until the next build. However, this will give you a good idea of how immediate policy changes impact any violations you currently have.

Note

Policy Reevaluation will not enact any actions you may have attached to your policies.

9.11 Sonatype CLM PDF Report

In some implementations of Sonatype CLM, not everyone will have access to the CLM Server or any of the integrated enforcement points, and in turn, any of the associated reports. However, certain individuals or teams would likely still benefit from the information the CLM Report provides. Even if that's not your particular situation, you may reach a point where you would like to produce an archive of a report for historical and audit purposes. Given this need, every report you produce with Sonatype CLM can be converted into a PDF.

Though the information presented in both the web application and the PDF are nearly identical, there are a few differences, mainly formed out of the contrasting visual and layout capabilities of a web application versus PDF. Below, we'll discuss how to create this PDF version as well as highlight some of the differences between the two.

9.11.1 Creating the PDF

If you've been working with Sonatype CLM for a while, you might have started to notice a set of blue icons in the top right of every report. While the first icon is related to reevaluating the report, the button on the right allows you to create a PDF version of the report. Simply, click on this button  and Sonatype CLM will prompt your browser to download a PDF version of the report.

Note

The report filename will be unique each time you use the button. However, in general the report will include the job name, build number, and the timestamp associated with the results. For example, `clm-0-20131017-140140.pdf`

9.11.2 Reviewing the PDF

The information provided by the PDF is identical to information that is provided within the application composition report in the application user interface. This includes the *Summary*, *Policy*, *Security Issues*, and *License Analysis* tabs. Within the PDF, the order of information is presented top to bottom, following the logic of the report tabs from left to right. With the exception of the first page, which provides the *Summary*, each section has a label to indicate the corresponding tab of the Application Composition Report:

Policy Violations

displays a list of all policy violations, ordered by threat level, for each component.

Security Issues

displays a list of all security issues, ordered by SV score, for each component.

License Analysis

displays a list of components, ordered by threat level of the associated license threat group for the license(s) for the component.

Components

lists all components identified during the scan, as well as a summary of the data provided in the other tabs.

Summary

The summary section is identical to the HTML version of the report and visible in [Figure 9.43](#)

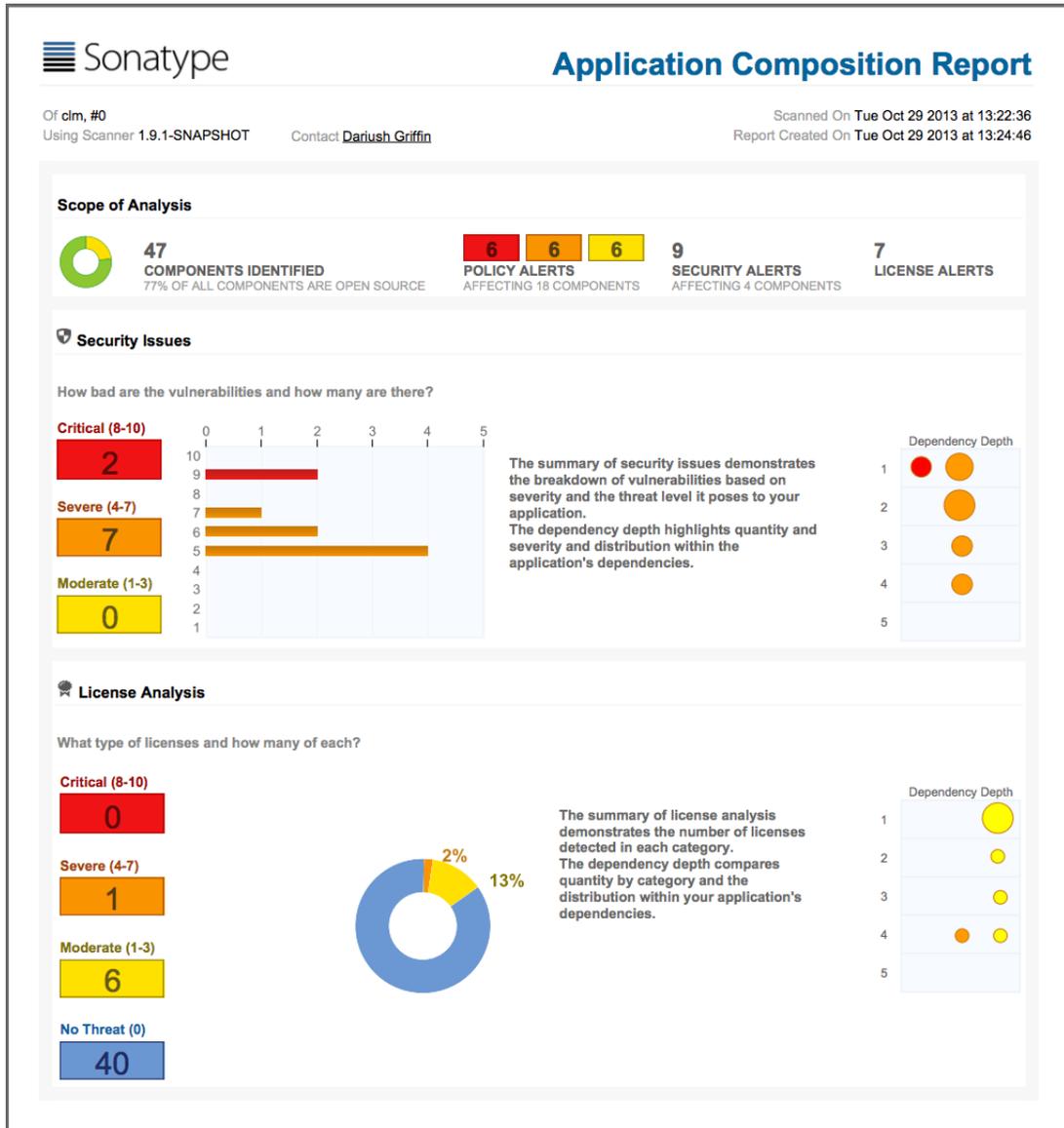


Figure 9.43: Summary Section of a Application Composition Report in PDF Format

Policy Violations

The *Policy Violations* section as visible in [Figure 9.44](#) displays the details for all scanned components. This matches the data displayed in the *Policy* tab of the *Component Information Panel (CIP)*. It should be noted, that depending on the number of violations in your application, this section could be very long.

Policy Violations				
Threat	Policy Name	Actions	Component	Conditions
9	Security-High		org.apache.geronimo.framework : geronimo-security : 2.1	Security Vulnerability Severity >= 7, Security Vulnerability Severity < 10, Security Vulnerability Status is not NOT_APPLICABLE
7	Security-Medium		commons-httpclient : commons- httpclient : 3.1	Security Vulnerability Severity >= 4, Security Vulnerability Severity < 7, Security Vulnerability Status is not NOT_APPLICABLE
			org.apache.geronimo.framework : geronimo-security : 2.1	Security Vulnerability Severity >= 4, Security Vulnerability Severity < 7, Security Vulnerability Status is not NOT_APPLICABLE
			org.mortbay.jetty : jetty : 6.1.15	Security Vulnerability Severity >= 4, Security Vulnerability Severity < 7, Security Vulnerability Status is not NOT_APPLICABLE
			org.openid4java : openid4java : 0.9.5	Security Vulnerability Severity >= 4, Security Vulnerability Severity < 7, Security Vulnerability Status is not NOT_APPLICABLE
			tomcat : catalina-host-manager : 5.5.23	Security Vulnerability Severity >= 4, Security Vulnerability Severity < 7, Security Vulnerability Status is not NOT_APPLICABLE
			tomcat : servlets-default : 5.5.4	Security Vulnerability Severity >= 4, Security Vulnerability Severity < 7, Security Vulnerability Status is not NOT_APPLICABLE
			tomcat : tomcat-util : 5.5.23	Security Vulnerability Severity >= 4, Security Vulnerability Severity < 7, Security Vulnerability Status is not NOT_APPLICABLE

Figure 9.44: Policy Violations Section of a Application Composition Report in PDF Format

Security Issues

The *Security Issues* section displays a breakdown of all security issues found in the scan of the application, matching what is displayed in the HTML version of the report. An example is available in [Figure 9.45](#)

Security Issues

Threat Level	Problem Code	Component
9	CVE-2008-5518	org.apache.geronimo.framework : geronimo
	53927	org.apache.geronimo.framework : geronimo
	53928	org.apache.geronimo.framework : geronimo
	53929	org.apache.geronimo.framework : geronimo
7	CVE-2009-4611	org.mortbay.jetty : jetty : 6.1.15
	75808	org.mortbay.jetty : jetty : 6.1.15
6	CVE-2009-0039	org.apache.geronimo.framework : geronimo
	53932	org.apache.geronimo.framework : geronimo
5	CVE-2012-5783	commons-httpclient : commons-httpclient :
	87160	commons-httpclient : commons-httpclient :
	CVE-2009-1523	org.mortbay.jetty : jetty : 6.1.15
	CVE-2011-4461	org.mortbay.jetty : jetty : 6.1.15
	54186	org.mortbay.jetty : jetty : 6.1.15
	78117	org.mortbay.jetty : jetty : 6.1.15
	CVE-2011-4314	org.openid4java : openid4java : 0.9.5
	73737	org.openid4java : openid4java : 0.9.5
	CVE-2007-5333	tomcat : tomcat-util : 5.5.23
	CVE-2012-0022	tomcat : tomcat-util : 5.5.23
	41435	tomcat : tomcat-util : 5.5.23
	78573	tomcat : tomcat-util : 5.5.23
	CVE-2010-0150	

License Analysis

The *License Analysis* section displays a breakdown of all license issues found in the scan of the application, matching what is displayed in the HTML version of the report. It should be noted that depending on your license threat groups, and license assignments, this section of the report could be very long. A short example is displayed in [Figure 9.46](#).

License Analysis

License Threat	Component
 Not Declared, GPL, GPL-2.0+	 cobertura : cobertura : 1.6
 MIT, Apache-2.0, CC-BY, GPL-2.0+, LGPL-2.1+, Non-Standard	 edu.ucar : unidataCommon : 4.2.20
 GPL-3.0, No Sources	 javancss : javancss : 29.50
 LGPL-2.1, BSD-3-Clause, LGPL-2.0+, Non-Standard	 edu.stanford.ejalbert : BrowserLauncher2 :
 Apache-2.0, Non-Standard	 org.mortbay.jetty : jetty : 6.1.15
 LGPL-3.0, LGPL	 ch.qos.logback : logback-access : 0.6
 EPL-1.0, No Sources	 org.eclipse.foundation : org.apache.lucene 2.9.1.v20100421-0704
 EPL-1.0, No Sources	 org.eclipse.foundation : org.slf4j.api : 1.6.1
 LGPL-2.1, No Sources	 org.opencms.modules : com.alkacon.open
 Apache-2.0	 commons-beanutils : commons-beanutils :
 Apache-2.0	 commons-dbcp : commons-dbcp : 1.4
 Apache-2.0, Apache	 commons-httpclient : commons-httpclient :
 Apache-2.0	 commons-pool : commons-pool : 1.4
 Not Declared, Apache-2.0	 geronimo : geronimo-tomcat : 1.0
 Not Declared, Apache-2.0	 geronimo : geronimo-tomcat-builder : 1.1
 BSD-3-Clause	 net.sf.xradar : xradar : 1.1.2
 Apache-2.0, BSD-3-Clause, MIT	 org.apache.avro : avro-ipc : 1.5.0
 Apache-2.0, No Sources	 org.apache.flume : flume-ng-core : 1.9.0

Components

As mentioned above, this section brings together information from all the others. It displays the highest security issue identified (and the associated CVS Score), any declared and/or observed licenses (and the highest threat level of the associated), the match state, age, and the policy violation counts for each threat level band (red, orange, yellow, and blue) for each component. An example is displayed in [Figure 9.47](#). In most cases this section can be used as a detailed bill of materials.

Components

Component / Declared and Observed License	Match State	Age
 ch.qos.logback : logback-access : 0.6 	exact	8 years
 LGPL-3.0; LGPL		NA
 cobertura : cobertura : 1.6	exact	9 years
 Not Declared; GPL, GPL-2.0+		NA
 commons-beanutils : commons-beanutils : 1.8.3	exact	4 years
 Apache-2.0		NA
 commons-dbcp : commons-dbcp : 1.4	exact	4 years
 Apache-2.0		NA
 commons-pool : commons-pool : 1.4	exact	7 years
 Apache-2.0		NA
 edu.stanford.ejalbert : BrowserLauncher2 : 1.3 	exact	3 years
 LGPL-2.1; BSD-3-Clause, LGPL-2.0+, Non-Standard		NA
 edu.ucar : unidataCommon : 4.2.20	exact	3 years
 MIT; Apache-2.0, CC-BY, GPL-2.0+, LGPL-2.1+, Non-Standard		NA
 geronimo : geronimo-tomcat : 1.0	exact	9 years
 Not Declared; Apache-2.0		NA
 geronimo : geronimo-tomcat-builder : 1.1	exact	8 years
 Not Declared; Apache-2.0		NA
 javancss : javancss : 29.50 	exact	6 years
 GPL-3.0; No Sources		NA
 net.sf.xradar : xradar : 1.1.2 	exact	4 years
 BSD-3-Clause		NA

Note

In some cases a URL for the project is provided. This is indicated by an information icon ⓘ.

Chapter 10

Sonatype CLM and Repository Management

Repository managers allow you to manage software components required for development, deployment, and provisioning and fulfill a central role for component lifecycle management. A repository manager greatly simplifies the maintenance of your own internal repositories and access to external repositories. Using a repository manager is a recommended best practice for development efforts using Apache Maven or other build systems with declarative and automated transitive dependency management.

By proxying external repositories as well as providing a deployment target for internal components, a repository manager becomes the central and authoritative storage platform for all components. You can completely control access to, and deployment of, every component in your organization from a single location. It allows you to control, which components get into your products from external sources as well as examine, and keep track of artifacts produced by your build systems.

In terms of the incoming components a repository manager allows you to secure the connection to an external repository and ensure that your component usage is not publicly exposed.

Just as Source Code Management (SCM) tools are designed to manage source artifacts, repository managers have been designed to manage and track external dependencies and components generated by your build. They are an essential part of any enterprise or open-source software development effort. They enable greater collaboration between developers and wider distribution of software bring increased build performance due to local component availability and reduced bandwidth needs by avoiding repeated downloads to your setup.

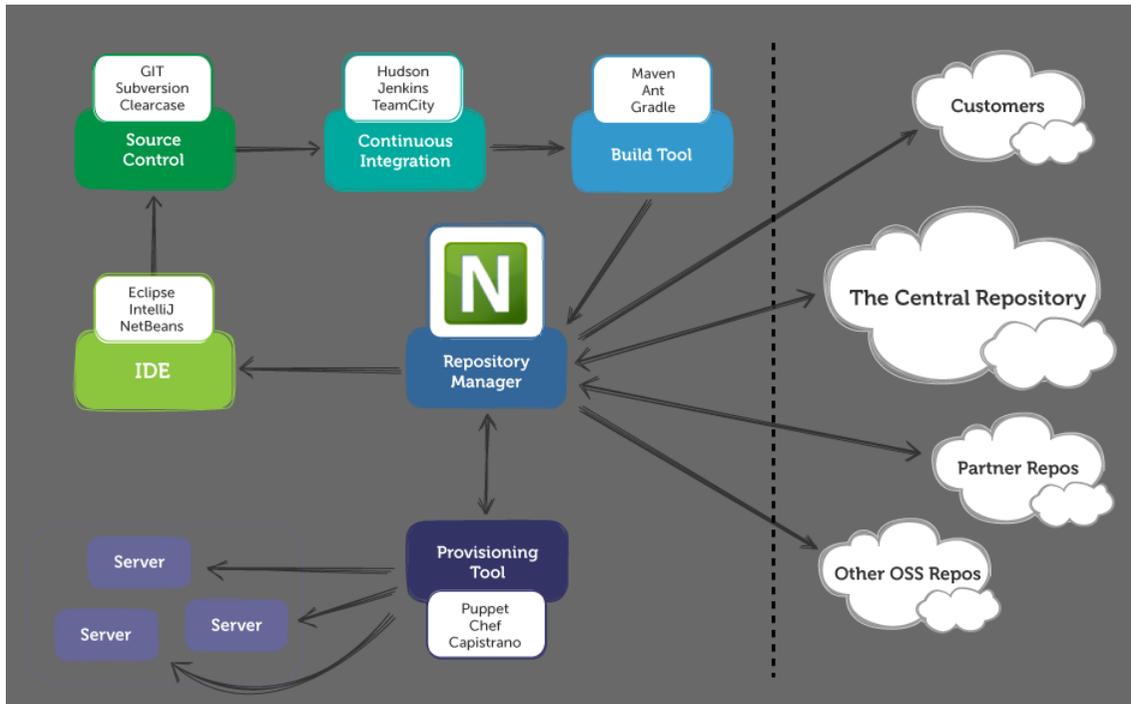


Figure 10.1: The Central Role of A Repository Manager in Your Infrastructure

For additional information, the book *Repository Management with Nexus* provides an extensive [introduction to repository management](#), its advantages, and stages of adoptions for further reference.

Chapter 11

Sonatype CLM for Nexus Pro

Sonatype Nexus is a powerful repository manager available as open source edition, [Nexus Open Source](#), as well as a fully supported, commercial edition, [Nexus Professional](#).

Both provide features such as

- Hosting repositories
- Proxying remote repositories
- Deployment target for third-party components
- Secure access to the Central Repository
- Grouping proxy and hosted repositories as well as snapshot and release repositories for simplified client configuration
- Hosting component documentation web sites
- Fine-grained security model
- LDAP integration
- Component search

Extensible documentation about Nexus features as well as instructions for installation, configuration and usage are available in the free book [Repository Management with Nexus](#) available at <http://www.sonatype.com/-/Support/Books/Repository-Management-with-Nexus>.

Nexus exposes all components via HTTP and can therefore be easily integrated with other tools. For more powerful integration than component retrieval, Nexus can be integrated by facilitating the REST API, used by the Nexus user interface and provided to other integrations. For more Nexus internal extension, the creation of additional plugins provides a wealth of possibilities.

TIP

The Nexus Professional Edition is using the same Nexus core as the open source edition and merely adds further functionality with additionally installed plugins.

The additional features provided by Nexus Professional include:

- Procurement Suite for component availability control
- Staging Suite for controlled release process and build promotion
- Enhanced LDAP support and integration with Atlassian Crowd
- Maven settings.xml management
- User account self serve
- Component validation and verification

11.1 Repository Health Check (RHC) vs. Sonatype CLM

It's likely, even as a user of Nexus Open Source, that you have seen some of the capabilities of Repository Health Check. For those that haven't, Repository Health Check (RHC) is a tool included within Nexus providing users with a quick glance at component properties in a repository. The results include a top level view of security vulnerabilities and license characteristics. Users of Nexus Professional are provided with security and license information as well as age and popularity data when searching for components. All this information is available in Nexus for manual searches and interaction with Nexus. There is however no automation available and no direct relationship to your software exists besides the fact that it's build accesses Nexus.

Sonatype CLM allows you to identify applications within your business. These applications can then be evaluated throughout the software development life cycle. This includes during development in your IDE, at build time in your CI server, and during the release phases in your repository manager.

With each evaluation of an application, components will be identified, and in the cases where components can be matched to those in the Central Repository, information similar to that in RHC will be provided.

An additional aspect of this evaluation is the ability to establish policy. Policy is simply a set of rules that allows you to validate the components used in your application based on the aspects available in CLM. When a component is found to break one of these rules, a violation occurs, and these results are provided through a number of reports, all available in the Sonatype CLM Server.

Taking a step back, looking at both RHC and Sonatype CLM at a high level, RHC is a static and limited view of specific data. This can help improve your component usage, but offers limited mitigation of risk. In contrast, the features of Sonatype CLM provide a robust set of features allowing you greatly expanded control over what components are used in your applications and take advantage of automation tools throughout the different phases of your software development lifecycle.

Note

Nexus Open Source and Nexus Professional both provide access to RHC, though the capabilities are expanded for Nexus Professional users. For more information on RHC and Nexus in general, please refer to the free book [Repository Management with Nexus](#).

11.2 Connecting Nexus to CLM Server

The first step to enabling the features associated with Sonatype CLM is connecting to an existing Sonatype CLM Server. The Sonatype CLM Server is a separate server application that Nexus integrates with via API calls.

If this is your first time working with Sonatype CLM, and you haven't already installed and configured your Sonatype CLM Server, you will want to do that before moving forward. Instruction can be found in the <<chapter-clm-server-setup, Sonatype CLM Server Setup chapter>>.

Once your Sonatype CLM Server is installed and configured, you are ready to connect Nexus to the CLM Server. From within Nexus Professional, click on the *CLM* menu item in *Administration* section on the left of the Nexus application window. This will open the tab visible in Figure [11.1](#).

Settings

[Make Nexus Staging even more powerful with Sonatype CLM](#)

CLM Server URL ★ ?

Username ★ ?

Password ★ ?

Request Timeout ?

Properties ?

Save Reset Test Connection

Figure 11.1: CLM configuration tab in Nexus

To configure the connection to the Sonatype CLM Server, follow these instructions:

1. Enter the URL location for your installation of the Sonatype CLM server.
2. Enter the Username and Password.

Tip

We recommend that you create a unique machine account that has access to the application(s) you wish to link to your Nexus repositories. . (Optional) Enter a *Request Timeout*. . (Optional) Enter information in the *Properties* input field using a key=value definition per line. An example is

```
procArch=false
ipAddresses=true
operatingSystem=false
```

These properties are passed to the CLM Server and can, for example, determine what properties are logged as part of a validation. Consult the CLM Server documentation for suitable parameters. In most use cases you will not need to configure any properties.

3. Click *Test Connection* to verify the information you have entered is correct and a connection to the CLM Server can be established.
 4. Click the *Save* button.
-

If successfully connected, a list of available applications will be displayed as seen in Figure 11.2. For more information of setting up organizations, applications, and policies, please review the chapter on [Sonatype CLM Policy Management](#).

Alternatively you can enable, disable, and/or configure the Sonatype CLM integration by adding the *CLM: Configuration* capability like any other capability as documented in the [Accessing and Configuring Capabilities](#) section of the Nexus book.

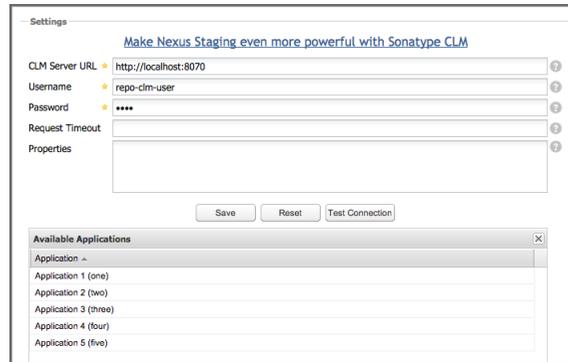


Figure 11.2: CLM configuration tab after Test Connection

Note

The features described here require licenses for Nexus Professional as well as Sonatype CLM Server that activate them. You can obtain them from our support team and will have to install them prior to the configuration.

With the connection between the CLM Server and Nexus established, you can configure any organizations, applications, and policies in the CLM server. Because Nexus will be accessing the CLM server using an application identifier (App ID), you will need to configure one application for each different application use case in Nexus.

11.3 Accessing CLM Component Information

As a native capability, Nexus provides robust search capability for returning components that exist in your repositories. When components are returned in your search results (see below), an option to see all

versions is displayed.

Group	Artifact	Version	Age	Popularity	Security Issues	License Threat	Download
struts:commons-struts2-core	struts:commons-struts2-core	2.1.4-1.2-2.4	6.1 yrs			Non-Standard	pom, jar, sources, jar
struts:commons-struts2-core	struts:commons-struts2-core	2.1.3-1.2-2.4	7.0 yrs			Non-Standard	pom, jar, sources, jar
struts:commons-struts2-core	struts:commons-struts2-core	2.1.3-1.2-2.3	7.7 yrs			LGPL-2.1+	pom, sources, jar
struts:commons-struts2-core	struts:commons-struts2-core	2.1.2-1.1-2.3	8.4 yrs			LGPL-2.1+	pom, jar, sources, jar
struts:commons-struts2-core	struts:commons-struts2-core	2.1-1.1-2.3	8.4 yrs			Not Provided	pom, jar
struts:commons-struts2-core	struts:commons-struts2-core	2.1-1.1-2.2	8.4 yrs			Not Provided	pom, jar
struts:commons-struts2-core	struts:commons-struts2-core	2.0-1.1-2.3	8.4 yrs			Not Provided	pom, jar
struts:commons-struts2-core	struts:commons-struts2-core	1.9-1.1-2.3	8.4 yrs			Not Provided	pom, jar

Figure 11.3: Typical Search Results in Nexus Pro

Tip

To get results that are not in the local Nexus cache, you will want to make sure the *Download Remote Index* option is enabled for the proxy repository. For guidance on this, checkout [section 6.2.4 \(specifically Fig 6.9\): Configuring Repositories](#) in the Nexus Book.

Clicking this link will display additional information in the search panel, as well as expand information available for each selected component. Depending on your Nexus license you will have one of the two options below.

RHC

Configuring an applicable repository to use RHC (Repository Health Check) will enable the repository to be analyzed by Sonatype directly, and will display (when available) security, license, age and popularity data. Details are provided in the Component Info tab located below the search panel.

Sonatype CLM

Configuring Nexus to connect to Sonatype CLM will provide the same information available for RHC, but will also provide additional general and policy violation information for each component.

Group	Artifact	Version -	Age	Popularity	Security Issues	License Threat	Download
org.apache.struts	struts2-core	2.3.8	1.3 yrs				pom, jar, sources, jar, javadoc, jar
org.apache.struts	struts2-core	2.3.7	1.4 yrs				pom, jar, sources, jar, javadoc, jar
org.apache.struts	struts2-core	2.3.4.1	1.6 yrs				pom, jar, sources, jar, javadoc, jar
org.apache.struts	struts2-core	2.3.4	1.8 yrs				pom, jar, sources, jar, javadoc, jar
org.apache.struts	struts2-core	2.3.3	1.9 yrs				pom, jar, sources, jar, javadoc, jar
org.apache.struts	struts2-core	2.3.16.1	17 d				pom, jar, sources, jar, javadoc, jar
org.apache.struts	struts2-core	2.3.16	108 d				pom, jar, sources, jar, javadoc, jar
org.apache.struts	struts2-core	2.3.15.3	154 d				pom, jar, sources, jar, javadoc, jar
org.apache.struts	struts2-core	2.3.15.2	181 d				pom, jar, sources, jar, javadoc, jar
org.apache.struts	struts2-core	2.3.15.1	247 d				pom, jar, sources, jar, javadoc, jar

Displaying Top 37 records [x](#) [Clear Results](#)

Figure 11.4: Nexus Search Showing All Versions

Note

Nexus search is only available for open source Java components.

For now, we'll focus on the additional information available through Sonatype CLM. To access this, you need to click on the Component Info tab. It is located just below the displayed search results, to the right of the directory tree for the selected component.

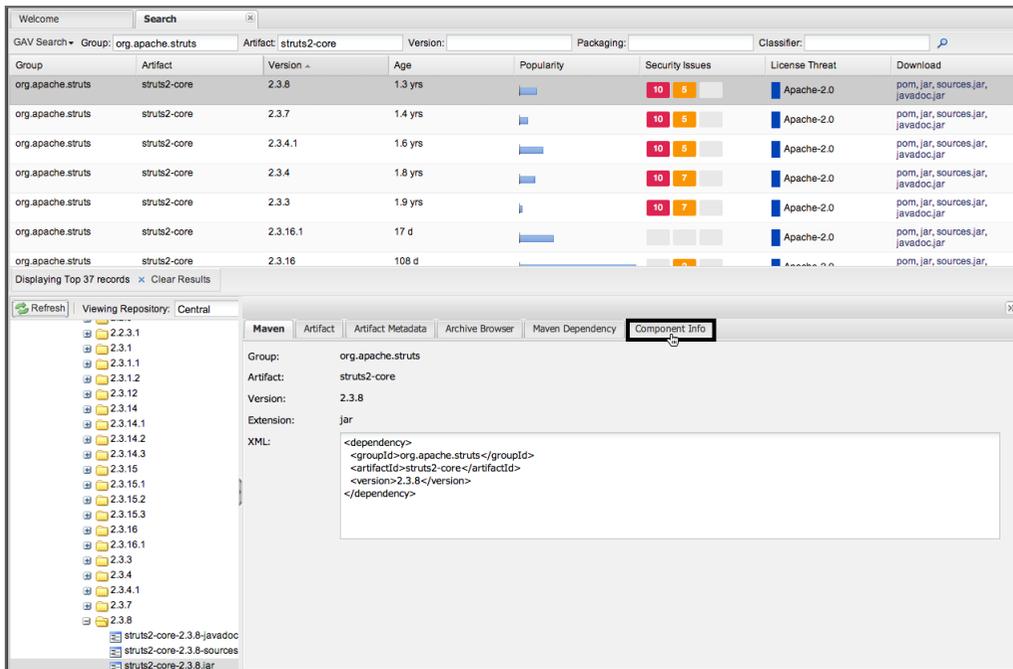


Figure 11.5: Accessing the Component Info Tab

Note

Only users that are logged in will be able to see the Component Info tab.

Clicking on the Component Info tab will display a drop down list of applications associated with your Sonatype CLM Server. Once you have selected an application, the Component Information Panel (CIP), similar to what is provided via the Application Composition Report and CLM for Eclipse, will be displayed.

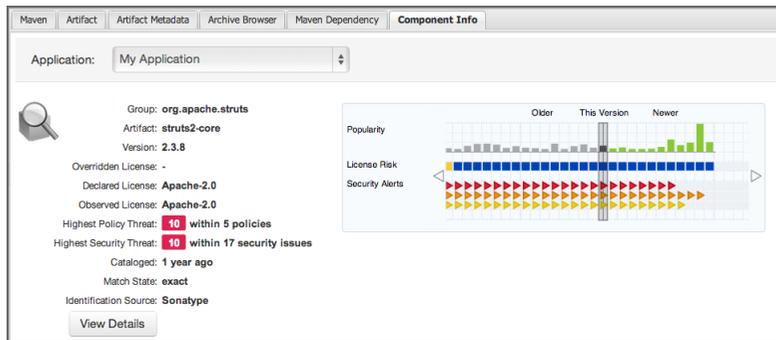


Figure 11.6: Component Information Panel

Note

Information on the Component Info tab requires a Sonatype CLM License. Nexus Pro Users will simply be provided with additional details regarding the security vulnerabilities and license issues. Those using Nexus Open Source will not have access to the Component Info tab.

11.4 The Component Information Panel (CIP)

As mentioned above, when the Component Information Panel is first displayed, you will need to select an application corresponding to your application on the CLM Server. This application will not change until you select a new one.

The Component Information Panel is divided into two areas. On the left side is component data, which includes information related to the component itself. To the right of the component information, a graphical display of any security or license issues, as well as popularity data for each version of the component is displayed. By default the current version of the component is selected. In the event there are more versions than can be displayed, arrows on the right and left allow for scrolling to newer or older versions. In addition, you can click on any of these versions (if available), which will change the information that is displayed on the left of the CIP.

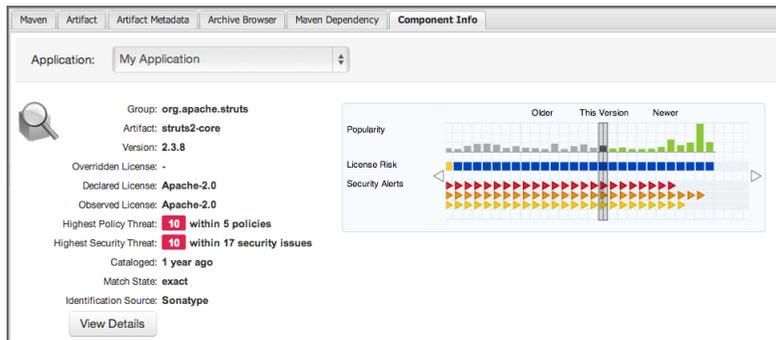


Figure 11.7: Component Information Panel Example

Note

In the screenshot above, we have sized the panels in Nexus to make all CIP information visible. By default the view will allow you to vertically scroll to view all information.

The textual information on the left includes:

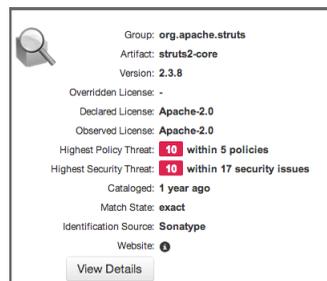


Figure 11.8: CIP Text

Coordinates

The identifying information for a component.

Overridden License

If you have chosen a different license for the component, it will be displayed here. This could e.g. be the case if you have purchased a license for a component allowing distribution, while the component is originally GPL.

Declared License

Any license that has been declared by the author.

Observed License

Any license(s) found during the scan of the component's source code.

Highest Policy Threat

The highest threat level policy that has been violated, as well as the total number of violations.

Highest Security Threat

The highest threat level security issue and the total number of security issues.

Cataloged

The age of the component based on when it first was uploaded to the Central Repository.

Match State

How the component was matched (exact, similar, or unknown).

Identification Source

Whether a component is identified by Sonatype, or claimed during your own process.

Website

If available, an information icon providing a link to the project is displayed.

The graph itself is laid out like a grid, with each vertical piece representing a particular version. The selected version being identified by a vertical line. The information displayed in the graph includes:

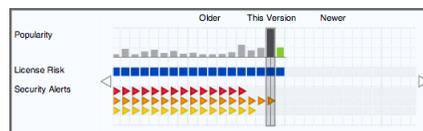


Figure 11.9: CIP Graph

Popularity

The popularity for each version is shown as a bar graph. The larger the graph the more popular the version.

License Risk

This will display the license risk based on the application that is selected, and the associated policy and/or license threat groups for that application. Use the application selector to change the application, and corresponding policies the component should be evaluated against.

Security Alerts

For each version, the highest security threat will be displayed by color, with the highest shown as red, and no marker indicating no threat.

11.5 Component Details (CLM)

In addition to the security vulnerability and license issue details provided, any particular policy violations for a component will be displayed as well. This can be helpful in determining if a component will meet the standards for component lifecycle management your company has established.

To view these details, click on the *View Details* button located below the Component Information.

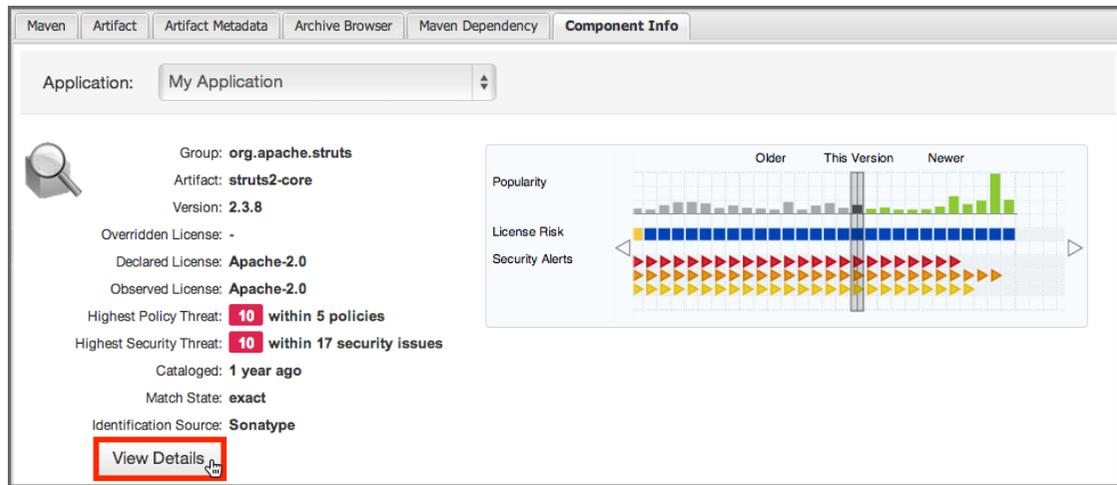


Figure 11.10: View Details Button

This will create a new tab in the main Nexus panel with the label CLM Detail.

CLM Details for org.apache.struts:struts2-core:2.3.16 in the context of CLM Application My Application

Policy Violations

Policy	Constraint	Summary
Security-Medium	CVSS >=4 and <7	Found 2 Security Vulnerabilities with Severity >= 4
		Found 2 Security Vulnerabilities with Severity < 7
		Found 2 Security Vulnerabilities with Status OPEN

License Analysis

Threat Level	Declared License(s)	Observed License(s)
Liberal	Apache-2.0	Not Provided

Security Issues

Threat Level	Problem Code	Status	Summary
5	CVE-2014-0094	Open	The ParametersInterceptor in Apache Struts before 2.3.16.1 allows remote attackers to "manipulate" the ClassLoader via the class parameter, which is passed to the getClass method.
	OSVDB-103918	Open	Apache Commons FileUpload ParametersInterceptor ClassLoader Manipulation Remote DoS

Figure 11.11: View Details

Note

In order to see the details for additional components, select another component from the search results, or select a different version in the CIP, and then click the *View Details* button.

11.6 Sonatype CLM for Nexus Staging

Sonatype CLM for staging in Nexus combines the powerful controls for your release process from Nexus with the rich information and validation available in the CLM Server. Using them together you can ensure that any releases you produce are actively and automatically validated against up to date information in terms of security vulnerabilities and license characteristics of all the components you use and any whitelists or blacklists you maintain as well as other policies you have defined are enforced.

You will need to have completed the following items before using CLM with Nexus Staging. This includes:

On the CLM Server

- Created an Organization
- Created an Application
- Created a Policy

In Nexus CLM

- Created a Staging Profile

Note

Before using CLM for staging you should be familiar with the general setup and usage patterns of the Nexus Staging Suite documented in [the chapter on staging](#), located in the Nexus book. There, you will be guided through the process to get Nexus prepared to handle your staging needs.

11.6.1 Staging Profile Configuration

As mentioned in the note above, you should already have your staging profile configured. This configuration can then be used for a staging profile or a build promotion profile by configuring the *CLM Application* field in the Staging Profile.

[The figure below](#) shows an example staging profile with a CLM application configured.

The screenshot shows the configuration for a staging profile named 'NXS301'. The configuration is as follows:

Property	Value
Profile ID	12aaba0a60ad0611
Deploy URL	http://localhost:8081/nexus/service/local/staging/deploy/maven2
Profile Name	NXS301
Profile Selection Strategy	Explicit or Implicit - selected by ID or Nexus will calculate match
Searchable Repositories	<input checked="" type="checkbox"/>
Staging Mode	Deploy and UI Upload
Template	Maven2 (hosted, release)
Repository Target	NXS-301
Release Repository	Releases
CLM Application	Example
	CLM Application Management
Content Type	maven2

Figure 11.12: Staging Profile with a CLM Application Configured

11.6.2 Policy Actions for Staging

While not a requirement for using CLM with Nexus staging, CLM does have the ability to Fail or Warn on staging closure. This is managed by setting the *Stage Release* and *Release* actions for each policy. These policy actions can be configured to warn, fail, or do nothing (default). The figure below provides an example policy that would warn for a staging deployment and fail a release.

Edit Policy

Name Threat Level 3

Constraints ⊖ ⊕ ✎

Actions

	Stage ✎	Fail	Warn	Do Nothing	Notify
Procure				✓	👤 0
Develop				✓	👤 0
Build				✓	👤 0
Stage Release			✓		👤 0
Release		✓			👤 0
Operate				✓	👤 0

Figure 11.13: Staging and Release Configuration for a Policy in the CLM Server

Configuration of Policy Actions is managed via the Sonatype CLM Server. While we'll cover the basic settings below, for instruction on setting these actions, please review the Policy Management chapter, specifically the section on managing policy actions.

The configuration of the *Stage Release* action of a policy in the CLM Server is used for closing the staging repository. Based on the action chosen, the staging repository will respond as follows:

- If the policy action is set to *Fail*, when a policy is violated, the staging repository closing fails.
- If the policy action is set to *Warn*, when a policy is violated, the staging repository closes successfully, but a warning will be produced.
- If the policy action is set to *Do Nothing*, the staging repository closes successfully regardless of any policy violations.

11.7 Policy Actions for Release Repositories

As with CLM and policy, Nexus also has actions specific to the *Release* feature, and these can be configured to fail, warn or do nothing and are used for releasing or promoting the staging repository.

Once the staging profile is configured with the CLM application identifier any deployment triggers a CLM policy evaluation, which will be visible as *Activity* for the staging repository. Any rule failures are provided with further information in the detail panel. Figure 11.14 displays a staging repository with CLM rule validations and a failure. The *View Full Report* buttons links back to the Sonatype CLM Server, which displays the detailed Application Composition Report.

The screenshot shows the Sonatype Nexus Staging Repository interface. At the top, there's a 'Welcome' tab and a 'Staging Repository' tab. Below the tabs are action buttons: Refresh, Close, Promote, Release, and Drop. A 'Filter by profile' dropdown is also present. A table lists the repositories:

Repository	Profile	Status	Updated	Description
catchall-009	CatchAll	open	2013-Apr-10 23:27:04	Implicitly ..
nxs301-008	NXS301	closed	2013-Apr-10 23:12:13	test

Below the table, the 'catchall-009' repository is selected, and the 'Activity' tab is active. The activity list shows:

- open
- close
- Evaluating rules: clm
- Evaluating rule: clm
- Failed: clm
- 1 rule failed: clm
- Evaluating rules: Default Always Run
- Evaluating rule: Repository Writable
- Passed: Repository Writable
- All rules passed: Default Always Run
- Close failed

On the right, the 'Event: Failed: clm' details are shown, dated Wednesday, April 10, 2013 23:27:04 PDT (GMT-0700). A red banner indicates 'Validation failed.' Below this, it states 'Found 28 violations in 28 components'. Two summary boxes show '7 CRITICAL' and '21 SEVERE' violations. A 'View Full Report' button is also present.

Figure 11.14: Staging Repository Activity with a CLM Evaluation Failure and Details

Chapter 12

Sonatype CLM and Continuous Integration

The idea of continuous integration is that software development efforts are much easier to manage when test failures and other bugs can be identified closer to the time they were introduced into a complex system. As a consequence the differences between the working and the failing system are smaller and therefore easier to detect.

The terms continuous integration was coined by Martin Fowler and Kent Beck in their book *Extreme Programming Explained* published 1999. They introduced the idea of creating a system that continuously builds your software and executes any tests against it on a regular base as well, all in response to any changes of the source code.

Since its introduction, usage of continuous integration servers became an established and well understood best practice across the entire software development industry.

A number of commercial as well as open source servers are now available for installation in your own infrastructure as well as a managed service running remotely. Typical CI installations are often comprised of a number of servers running the actual build and being orchestrated by one master and build running on the CI servers range from simple compile builds to running large integration test suites or regression tests in an automated fashion. In addition CI servers are increasingly used for continuous deployment, where a series of successful build and test runs results in actual production deployment of the software.

Sonatype CLM can analyze the components used in your software development for security and license

characteristics. When integrated with a continuous integration server it becomes a dynamic analysis performed on a regular basis occurring potentially with each build running on the server.

Depending on your application server software you can use the Sonatype CLM [Hudson and Jenkins](#), [Bamboo](#), [the CLI](#) or [Maven](#).

Every one of these tools allows you to perform a full security and license analysis of the artifacts produced by the configured build backed by your Sonatype CLM server. It will provide you access to the analysis report.

Chapter 13

Sonatype CLM for Bamboo

Getting Sonatype CLM for Bamboo up and running is not difficult. However, there are a few things we expect you to have completed prior to getting started.

- Install and configure the Sonatype CLM Server
- Create an organization, and at least one application
- Evaluated the application at least once

If anything in the list above looks completely new, or has not been completed, bring everything to a full stop. Even if you aren't responsible for those areas of Sonatype CLM, you will need to have them complete before configuring Sonatype CLM for Bamboo.

Don't feel bad, we know you just wanted to get started as quickly as possible. To get the most out of it however, you will want to start with the following chapters:

- [Sonatype CLM Installation and Configuration chapter](#)
- [Policy Management chapter](#).

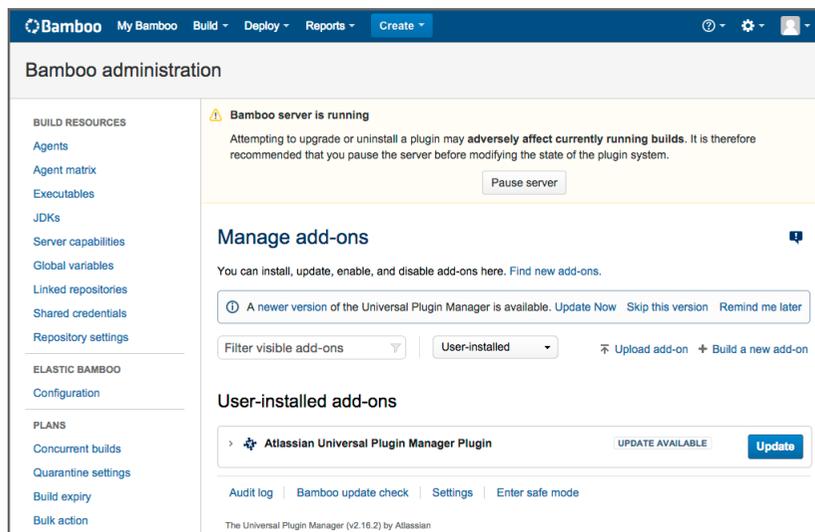
You may want to thumb through our [Application Composition Report chapter](#) as well.

Yeah, that's a lot of reading, but then again we wrote it just for users like you. Go ahead, we promise it will be worth it. If you do, we know you will be better prepared to extract the most value out of your Sonatype CLM for Bamboo installation.

Once you are ready, head on over and download Sonatype CLM for Bamboo from our [Sonatype CLM downloads page](#).

13.1 Install Sonatype CLM for Bamboo

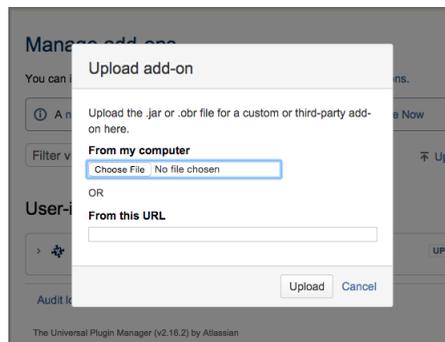
You should have the Sonatype CLM for Bamboo file downloaded. Now would be a good time to double-check the location you saved it to. That's something easy to forget. Got it? Good. Now, follow these steps:



1. First, access Bamboo's *Manage Add-ons* by clicking the *Gear* icon (within Bamboo) and then clicking *Add-ons* from the drop down list.
2. Next, click the *Upload Add-on* link. A modal will display, allowing you to specify a file, or a URL location. In this example, we'll be using the file you downloaded previously.
3. Choose the location of the Sonatype CLM for Bamboo file, click *Open*, and then the *Upload* button. The upload only takes a few seconds, but during this time, a progress modal will display.

4. Upon successful upload, a confirmation modal will display giving you a bit of information about the plugin. Click *Close*.

Awesome job! If everything went as planned, Sonatype CLM for Bamboo is now listed under *User-installed add-ons*. Let's head to the next step, configuration.



Tip

In most cases, pausing your Bamboo server is a good idea. Also, if you ever decide to the Uninstall Sonatype CLM for Bamboo, you can do it from the *Manage add-ons* area as well.

13.2 Configure Sonatype CLM for Bamboo

Ready to configure Sonatype CLM for Bamboo? Of course you are, but if you just joined us, we need to check and make sure you've installed it.

You have? Perfect!

Now you are ready to configure Sonatype CLM for Bamboo. You should be in the Bamboo administration area.

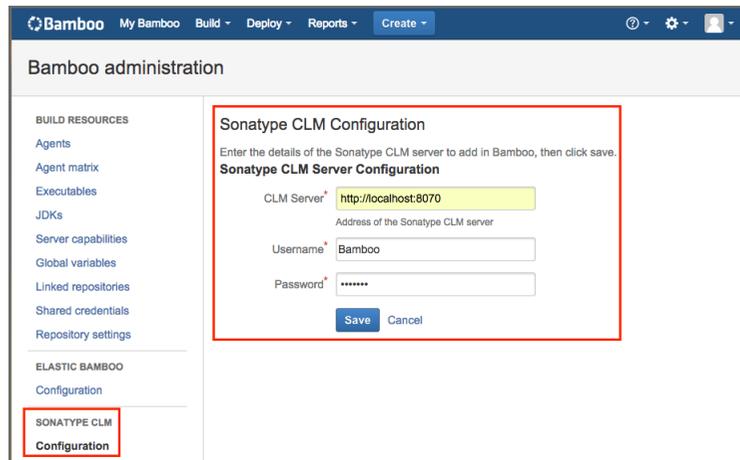
1. In the left hand navigation area / menu, locate a section titled *Sonatype CLM*. In that section, click on the *Configuration* link.
-

2. Now complete the following required fields:

- a. CLM Server - Enter the URL location for your installation of the Sonatype CLM server.
- b. User name and password.

Tip

We recommend that you create a unique machine account that has access to the application(s) you wish to link to your Bamboo Build(s)/Plans.



The screenshot shows the Bamboo administration interface. On the left, a navigation menu includes 'BUILD RESOURCES' (Agents, Agent matrix, Executables, JDKs, Server capabilities, Global variables, Linked repositories, Shared credentials, Repository settings), 'ELASTIC BAMBOO' (Configuration), and 'SONATYPE CLM' (Configuration). The main content area is titled 'Sonatype CLM Configuration' and contains the following form:

Sonatype CLM Configuration
Enter the details of the Sonatype CLM server to add in Bamboo, then click save.

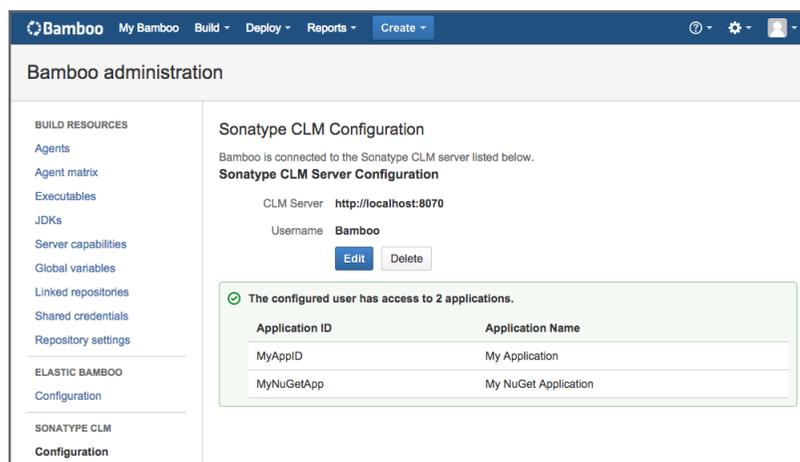
Sonatype CLM Server Configuration

CLM Server*
Address of the Sonatype CLM server

Username*

Password*

3. Click the *Save* button. Your configuration is saved, displaying the application(s) the user has access to.



The screenshot shows the Bamboo administration interface after saving the configuration. The 'SONATYPE CLM Configuration' section now displays the saved details:

Sonatype CLM Configuration
Bamboo is connected to the Sonatype CLM server listed below.

Sonatype CLM Server Configuration

CLM Server **http://localhost:8070**

Username **Bamboo**

The configured user has access to 2 applications.

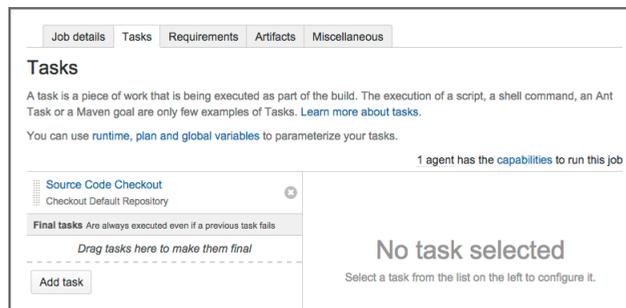
Application ID	Application Name
MyAppID	My Application
MyNuGetApp	My NuGet Application

13.3 Adding the Sonatype CLM Analysis Task

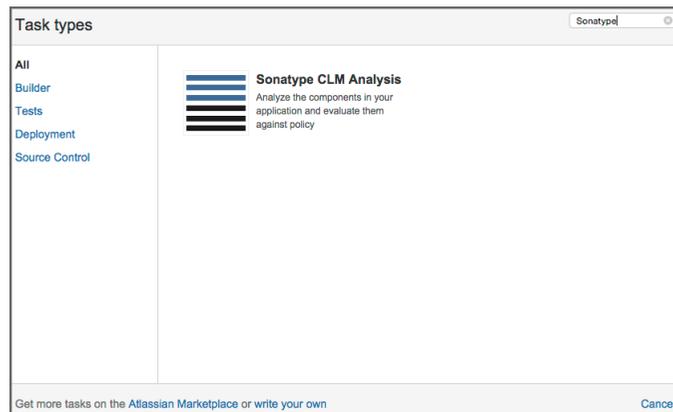
So now it's time to put everything you did to install and configure Sonatype CLM for Bamboo to good use, and add a Sonatype CLM Analysis Task.

The Sonatype CLM Analysis Task is available once you've installed and configured the Sonatype CLM Addon. The following steps will walk you through adding this new task to a job.

1. After navigating into a Bamboo Project > Plan > Stage > and then Job, click on the *Add task* button.



2. A modal will display offering a list of *Task Types*. The Sonatype CLM Analysis Task is listed in the *Test* type, or you can simply use search.



3. Enter the following information:

Task Description

A simple description to remember what the task does.

CLM Application

The list of CLM Applications corresponds to the account used during CLM for Bamboo configuration. Remember, this is the CLM Application containing the policies that components in the build will be evaluated against.

Fail build when CLM is unable to evaluate

Check this option if you want to fail the build when a CLM evaluation can't be performed. Once checked, if for any reason the evaluation is not generated, the build will be failed. An example of this might be if the CLM server is inaccessible. In the same example, but where the *Fail the build* option is left unchecked, the build would continue as it would have normally.

Tip

In any case where CLM is unable to evaluate an application, details are provided in the job/build-specific log.

CLM Stage

This corresponds to the stage you wish the policy evaluation of the application/project to be run against. Additionally, this will correspond to the stage location when viewing report information via the CLM Server. For example, if you chose the Build stage, summary and dashboard violation results will be displayed accordingly.

Scan Targets

The scan targets setting allows you to control which files should be examined with an Apache Ant styled pattern. The pattern is relative to the project workspace root directory and inherits the global configuration.

Module Excludes

If you are using the Sonatype CLM for Maven plugin, module files are created, and can contribute to results found during an evaluation. For information on how to exclude these files, please see the [Excluding Module Information Files in Continuous Integration Tools](#) of the Sonatype CLM for Maven chapter.

Sonatype CLM Analysis configuration [How to use the Sonatype CLM Analysis task](#)

Task description

Sonatype Evaluation

Disable this task

Select a CLM Application

Select a CLM Application from a list of available CLM Applications.

Specify a CLM Application

Specify the CLM Application ID, build variables can be used evaluate the ID at build time.

CLM Application

My Application (MyAppID) ▼

Fail build when CLM is unable to evaluate

CLM Stage

Build ▼

Controls the stage the policy evaluation is run against on the CLM server.

Scan Targets

A comma-separated list of Ant-style patterns relative to the workspace root that denote the files/archives to be scanned, e.g. `**/target/*.war,**/target/*.ear`
 If unspecified, the scan will default to the patterns `**/*.jar,**/*.war,**/*.ear,**/*.zip,**/*.tar.gz`

Module Excludes

A comma-separated list of Ant-style patterns relative to the workspace root that denote the module information files (`**/sonatype-clm/module.xml`) to be ignored, e.g. `**/my-module/target/**,**/another-module/target/**`
 If unspecified all modules will contribute dependency information (if any) to the scan.

Advanced Options

Options to be set on a case-by-case basis as advised by Sonatype Support.

4. Click the "Save" button.

The screenshot shows the Bamboo web interface for configuring a task. The breadcrumb trail is 'Build projects / My Project / My Application'. The page title is 'Configuration - My Application'. There are navigation buttons for 'Run' and 'Actions'. On the left, a sidebar shows 'Plan Configuration' with 'Stages & jobs' (1) and 'Default Stage' expanded to show 'Default Job' (0) and 'Branches' (0). The main content area has tabs for 'Job details', 'Tasks', 'Requirements', 'Artifacts', and 'Miscellaneous'. The 'Tasks' tab is selected, showing a list of tasks: 'Source Code Checkout' (Checkout Default Repository) and 'Sonatype CLM Analysis' (Sonatype Evaluation). Below the list, there is a section for 'Final tasks' and a dashed line with the text 'Drag tasks here to make them final'. An 'Add task' button is at the bottom left. On the right, a large message says 'No task selected' and 'Select a task from the list on the left to configure it.' Above this message, it says '1 agent has the capabilities to run this job'.

13.4 Reviewing CLM Policy Results

After your Bamboo job has completed, and your application has been successfully evaluated by the Sonatype CLM Server, a summary of the results will be provided on the Job Summary page. The summary results give a breakdown of the three threat level categories for policy:

- Critical (8-10)
- Severe (4-7)
- Moderate (2-3)

In addition to counts for each of these categories, a status for the success of the evaluation is provided, as well as a link to the Full report located on your Sonatype CLM Server is also provided. These are located just to the left of the summary results.

The screenshot displays the Bamboo web interface for a failed build. At the top, the navigation bar shows 'Bamboo My Bamboo Build Deploy Reports Create'. Below it, the breadcrumb trail is 'Build projects / My Project / My Application' and the build is identified as 'Build #4'. A red status bar indicates '#4 failed - Manual run by Admin'. The main content area has tabs for 'Build summary', 'Tests', 'Commits', 'Artifacts', 'Logs', and 'Metadata'. The 'Build result summary' section includes 'Details' (Completed: 23 Feb 2015, 3:37:09 PM - 1 second ago; Duration: 3 minutes; Labels: None) and a 'Responsible' section stating 'This build has been failing since #3' and 'No one has taken responsibility'. A 'CLM Policy Evaluation' section shows a bar chart with 3 Critical (red), 5 Severe (orange), and 2 Moderate (yellow) issues. A 'Full Report' link is provided at the bottom.

Tip

In the event CLM should encounter an issue during the evaluation related to the CLM Server itself, this will be indicated by one of three statuses: *Passed*, *Passed with Warnings*, or *Failed*.

Chapter 14

Sonatype CLM for Hudson and Jenkins

Eclipse Hudson and **Jenkins** are powerful and widely used open source continuous integration servers providing development teams with a reliable way to monitor changes in source control and trigger a variety of builds. They excel at integrating with almost every tool you can think of.

Historically the Hudson project and community split into two groups, with Jenkins as well as Hudson emerging as sibling products with a different focus going forward while sharing a common API for plugins. In general, with regard to the Sonatype CLM for CI functionality, the interaction will be near identical, with only a few differences, which are inherent to the CI, and not Sonatype CLM.

Sonatype CLM for Hudson and Jenkins evaluates the project workspace after a build for all supported component types, creates a summary file about all the components found and submits that to the Sonatype CLM service. The service uses that data to produce the analysis with the security and license information and send it back to the CI server. It will then use these results to render the analysis reports.

The file types supported for analysis are in tar/zip like format with the extensions tar, tar.bz2, tb2, tbz, tar.gz, tgz and zip or in Java archive formats of the type jar, ear, war, hpi, wsr, har, sar, rar, mar and nbm.

14.1 Installation

Sonatype CLM for Hudson and Jenkins is distributed as a Hudson plugin package (.hpi file) and is compatible with Jenkins and Hudson.

In order to install the plugin you have to log into Jenkins or Hudson as administrator and then select to *Manage Jenkins/Manage Hudson* to get to the global configuration menu displayed in Figure 14.1 in the Jenkins look. The Hudson look will be similar in content, yet different in colors and styling.

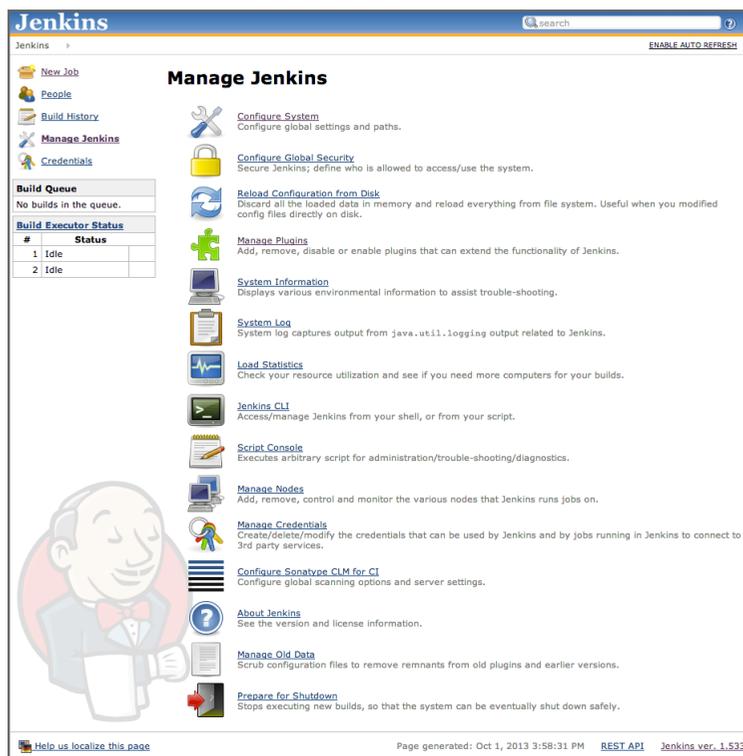


Figure 14.1: Jenkins Global Configuration Menu

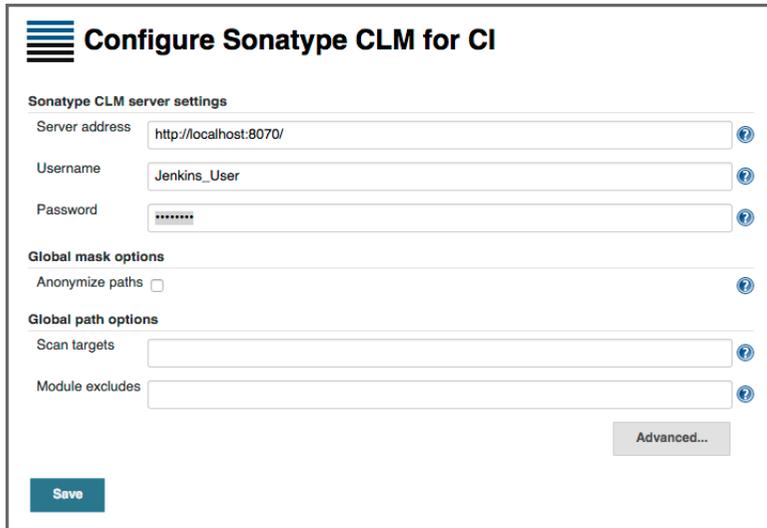
From the displayed configuration menu, select *Manage Plugins* and in the plugin management section, choose the *Advanced* tab.

The advanced plugin management allows you to upload a plugin distribution file (.hpi) in the section

entitled *Manual Plugin Installation* on Hudson and *Upload Plugin* on Jenkins. Click on *Choose File* and select Sonatype CLM for Hudson and Jenkins hpi file named `sonatype-clm-ci-x.y.z.hpi` with `x.y.z` representing a version number like `2.11.2` in the file selection dialog. Then press the *Upload* button. Once the plugin has been uploaded to the server, you need to restart your continuous integration server.

14.2 Global Configuration

After a successful installation of Sonatype CLM for Hudson and Jenkins, a new option will be available in the Jenkins/Hudson management area, *Configure Sonatype CLM for CI*. Follow these instructions to configure Jenkins or Hudson to connect to your Sonatype CLM Server.



Configure Sonatype CLM for CI

Sonatype CLM server settings

Server address

Username

Password

Global mask options

Anonymize paths

Global path options

Scan targets

Module excludes

Advanced...

Save

Figure 14.2: Global Configuration of Sonatype CLM for CI in Jenkins

Sonatype CLM server settings (required)

Server address

This is the address for the Sonatype CLM server as it can be reached from the Jenkins/Hudson server. By default, the Sonatype CLM Server address is `http://localhost:8070`.

If your Sonatype CLM server is behind a proxy server for serving HTTPS or other reasons, you have to use the public URL as it is reachable from the continuous integration server. Only

the master Jenkins/Hudson server connects to the CLM server and you therefore only need to ensure connectivity in terms of open firewall ports and proxy server settings between the master CI server and the CLM server.

Username

This is the username you wish to use to connect to the CLM Server. Since these settings will be used across all projects for your Jenkins/Hudson installation, we suggest creating a single account on the Sonatype CLM Server for Jenkins/Hudson, and then associating that account with the Application Evaluator role for the Organizations or Applications you will be linking to Sonatype CLM for Hudson/Jenkins.

Password

The password for the username entered above.

Tip

Username and password can also be configured [per job](#).

Global mask options

Anonymize paths

Enabling this feature will anonymize all paths before data is sent to the Sonatype CLM server. Ultimately, this prevents the CLM report from reporting the locations/occurrences of components.

Global path options

Scan targets

The scan targets setting allows you to control which files should be examined. The configuration uses an [Apache Ant styled pattern](#), is relative to each project's workspace root directory, and has a useful default setting that includes all `jar`, `war`, `ear`, `zip` and `tar.gz` files. The default value is therefore

```
**/*.jar, **/*.war, **/*.ear, **/*.zip, **/*.tar.gz
```

Note

This default only applies if, and only if, neither global nor job config specify scan targets. Adding to this, if you are using a private Maven repository, our default pattern will include your entire Maven repo. This could greatly increase the time necessary for your evaluation, as well as skew evaluation results. To avoid this, consider using a more specialized pattern like `**/target/*.jar`.

Module excludes

If you are using Sonatype CLM for Maven, you may have noticed the creation of module information files. The process for excluding modules is documented in the [Excluding Module Information Files in Continuous Integration Tools](#) section of the Sonatype CLM for Maven chapter.

Advanced options

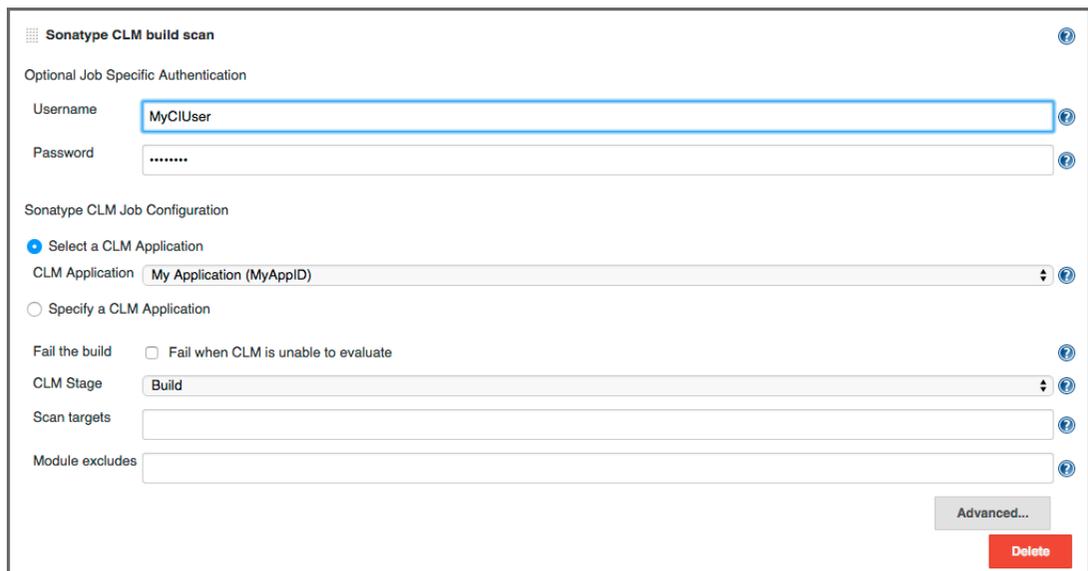
A number of additional parameters can be supplied to the plugin using this input field. Typically these parameters will be determined by Sonatype support.

14.3 Job Configuration

After a completed installation (see Section 14.1) and global configuration (see Section 14.2) of Sonatype CLM for CI, you are ready to configure an invocation as part of a specific job.

Depending on your job type it will be available as a pre and/or post-build step as well as an invocation as a main build step. A pre-build step or a main build step executed before your main build invocation step could be used to examine components existing in the workspace or being placed into the workspace by an earlier build step.

The typical invocation would be as main build step, after the package that should be examined has been created. An example configuration from Jenkins is displayed in Figure 14.3.



The screenshot shows the configuration page for the Sonatype CLM build scan plugin. The page is titled "Sonatype CLM build scan" and includes several sections:

- Optional Job Specific Authentication:** Contains fields for "Username" (set to "MyCIUser") and "Password" (masked with dots).
- Sonatype CLM Job Configuration:** Includes a radio button for "Select a CLM Application" (selected) and a dropdown menu for "CLM Application" (set to "My Application (MyAppID)"). There is also an option for "Specify a CLM Application" which is unselected.
- Fail the build:** A checkbox labeled "Fail when CLM is unable to evaluate" is currently unchecked.
- CLM Stage:** A dropdown menu is set to "Build".
- Scan targets:** An empty text input field.
- Module excludes:** An empty text input field.

At the bottom right of the configuration area, there are two buttons: "Advanced..." and "Delete".

Figure 14.3: Sonatype CLM Build Scan Configuration for a Build Step

The configuration options for Sonatype CLM for CI invocations mimic the parameters from the global configuration described in Section 14.2 and are appended to the global parameters. The configuration parameters are:

Optional Job Specific Authentication

While username and password can be [configured globally](#), in some cases you may want a certain job to be associated with a user who has permissions to specific Organization and/or Applications. Job Specific Authentication allows you to configure a user for this job and use the associated permissions to select the Application for the evaluation.

Username

the Sonatype CLM Server username you wish to use for this job.

Password

the password for the username above.

Sonatype CLM Job Configuration

Depending on what application is used, the policies associated to the application will be used for the analysis of this build job output. There are two options for choosing what CLM application to associate with the build:

Select a CLM Application

The CLM Application dropdown will be populated with the names of applications based on the permissions for the configured user name and password.

Specify a CLM Application

If you want to use a build variable to provide the CLM Application ID, you can enter it in the field displayed after selecting this option. Click on the help icon to the right of the field for information on using build variables (e.g. `_${THE_APPLICATION_ID}`) to evaluate the application at build time.

Fail the build

Check this option if you want to fail the build when a CLM evaluation can't be performed. Once checked, if for any reason the evaluation is not generated, the build will be failed.

An example of this might be if the CLM server is inaccessible. In this scenario, the build would fail. In the same example, but where the *Fail the build* option is left unchecked, the build would be marked unstable.

CLM Stage

This corresponds to the stage you wish the policy evaluation of the application/project to be run against. Additionally, this will correspond to the stage location when viewing report information via the CLM Server (e.g. if you chose the Build stage, summary and dashboard violation results will be displayed accordingly).

Note

Depending on how your policies are configured, this may impact warning and fail actions.

Scan targets

The scan targets setting allows you to control which files should be examined with an Apache Ant styled pattern. The pattern is relative to the project workspace root directory and inherits the global configuration.

Module excludes

You can exclude modules from being scanned with module information files configured in this setting. The default value is inherited from the global configuration.

Advanced options

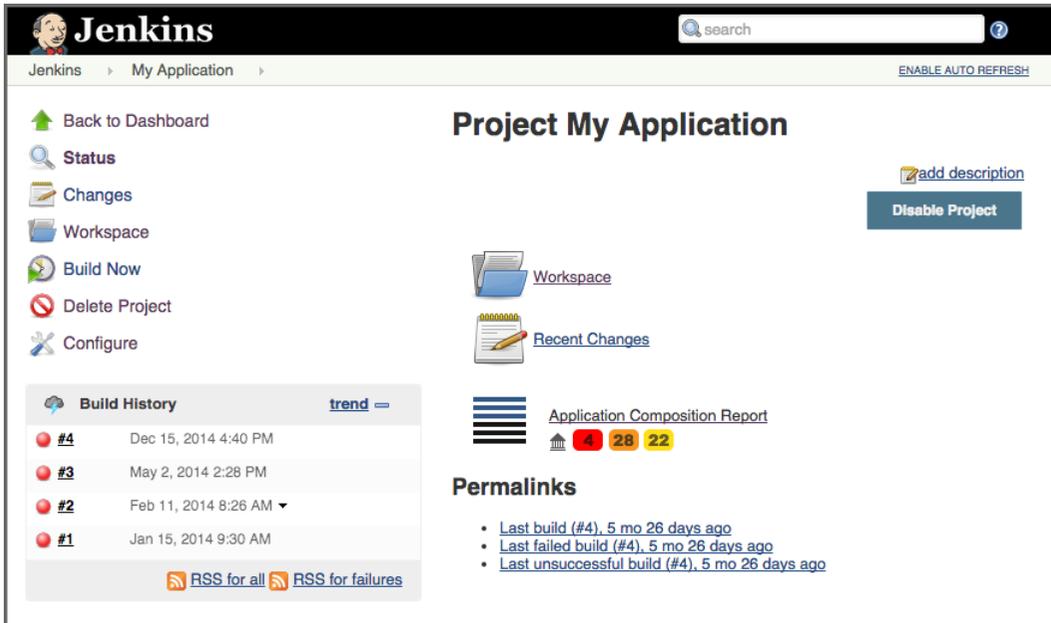
A number of additional parameters can be supplied to the plugin using this input field. Typically these parameters will be recommended to you by the Sonatype support team.

14.4 Inspecting Results

Once a specific build has successfully completed, Sonatype CLM for CI provides a link to the application composition report in the job list in the *Policy Violations* column as well as the project specific overview page. Clicking on the link *Application Composition Report*, will direct you to the display of the report within the Sonatype CLM Server. The three boxes (red, orange, and yellow) located below the link, give you counts for policy violations, and are based on the associated severities (high, medium, and low).

Note

Accessing this information may require a login. Also, if you are using a version of Sonatype CLM for Hudson and Jenkins prior to version 2.11, and Sonatype CLM Server 1.7, a message will display indicating your report has been moved. Following this link will take you to the report on the Sonatype CLM Server.



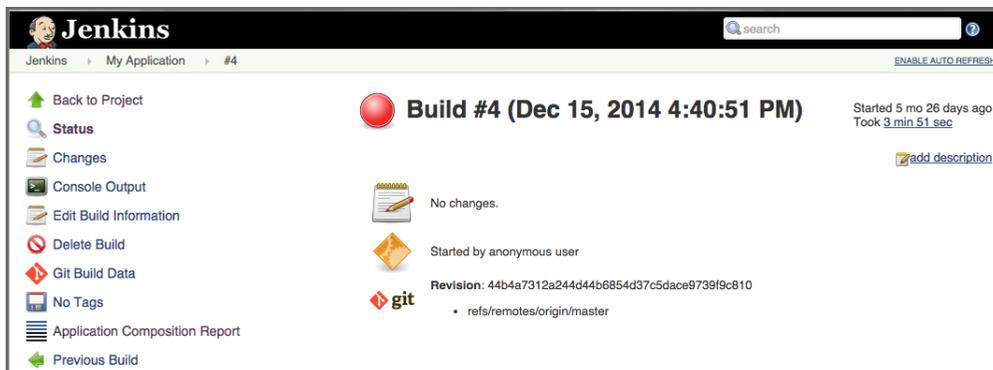
The screenshot shows the Jenkins interface for a job named "Project My Application". The page is divided into several sections:

- Left Navigation Menu:** Includes links for "Back to Dashboard", "Status", "Changes", "Workspace", "Build Now", "Delete Project", and "Configure".
- Build History:** A table showing the last four builds:

Build #	Timestamp
#4	Dec 15, 2014 4:40 PM
#3	May 2, 2014 2:28 PM
#2	Feb 11, 2014 8:26 AM
#1	Jan 15, 2014 9:30 AM
- Project Overview:** Displays "Workspace", "Recent Changes", and "Application Composition Report" (with counts: 4, 28, 22).
- Permalinks:** A list of links for "Last build (#4)", "Last failed build (#4)", and "Last unsuccessful build (#4)".

Figure 14.4: Job Overview Page with a Link to the Application Composition Report

If you are looking for previous report results, simply navigate to a specific build in the *Build History*. If you have previously scanned the application during that specific build, you will see a new item in the left menu, *Application Composition Report*. As with the report link above, you will be taken to the Sonatype CLM Server to review the results. An example is shown in Figure 14.5 below.



The screenshot shows the Jenkins interface for a specific build, "Build #4 (Dec 15, 2014 4:40:51 PM)". The page is divided into several sections:

- Left Navigation Menu:** Includes links for "Back to Project", "Status", "Changes", "Console Output", "Edit Build Information", "Delete Build", "Git Build Data", "No Tags", "Application Composition Report", and "Previous Build".
- Build Information:** Shows "Started 5 mo 26 days ago" and "Took 3 min 51 sec".
- Changes:** Displays "No changes.".
- Started by:** Shows "Started by anonymous user".
- Revision:** Shows "Revision: 44b4a7312a244d44b6854d37c5dace9739f9c810" and "refs/remotes/origin/master".

Figure 14.5: Left Menu with Link to the Application Composition Report

Chapter 15

Sonatype CLM and IDEs

An Integrated Development Environment, or IDE for short, is a software developer's workbench. It typically provides features to assist a developer in all their activities writing and maintaining software.

These include:

Powerful code editing features

with syntax highlighting, code completion and others helpful tools for efficient editing.

Integration of development tools

like compilers, debuggers, and profilers.

Support for version control systems

including tasks like checkout, commit, update and others.

Integration with language specific libraries

for accessing help, browsing libraries, completing code.

Build automation

within the tool e.g. for incremental compile as well as support for external build tools.

Testing support

for unit and integration test authoring and execution.

Integration with other software development tools

like code review systems, continuous integration servers, or repository managers.

The aim of an IDE is to maximize the productivity of a software developer during the entire software development process. Thus, an increasingly important part of these tasks is understanding, controlling, and managing the components that are part of your software product. This also includes carrying out the component lifecycle management related tasks like assessing security and license issues, as well as ensuring your component is using the best version possible.

Note

At this time, Sonatype CLM only supports the Eclipse IDE as a direct add on. However, other IDEs can be integrated using [Sonatype CLM for Maven](#).

Chapter 16

Sonatype CLM for Eclipse

Often only called Eclipse, the **Eclipse IDE** is a very powerful, open source IDE written mostly in Java and managed by the **Eclipse Foundation**. It can be used for development in a number of languages, and is the most widely used IDE for Java development. It features a powerful plug-in system that allows you to customize the IDE, with features that support a large number software development-related tasks including localization options, version control systems, and myriad of other tasks.

16.1 Installing Sonatype CLM for Eclipse

Sonatype CLM for Eclipse can be installed by adding a new software repository. Navigate to the *Help* menu and select *Install New Software*. Press the *Add* button in the dialog displayed in [Figure 16.1](#) and create a new repository with the *Location* set to the URL for Sonatype CLM for Eclipse releases from [URL for the Sonatype CLM for Eclipse repository](#) and a *Name* of your choice. Once you press *OK* a list of available releases is downloaded and an entry for the latest version of Sonatype CLM for Eclipse is displayed. Uncheck the item *Show only the latest versions of available software*, if you need to install an older release. [Figure 16.1](#) shows a list of releases available.

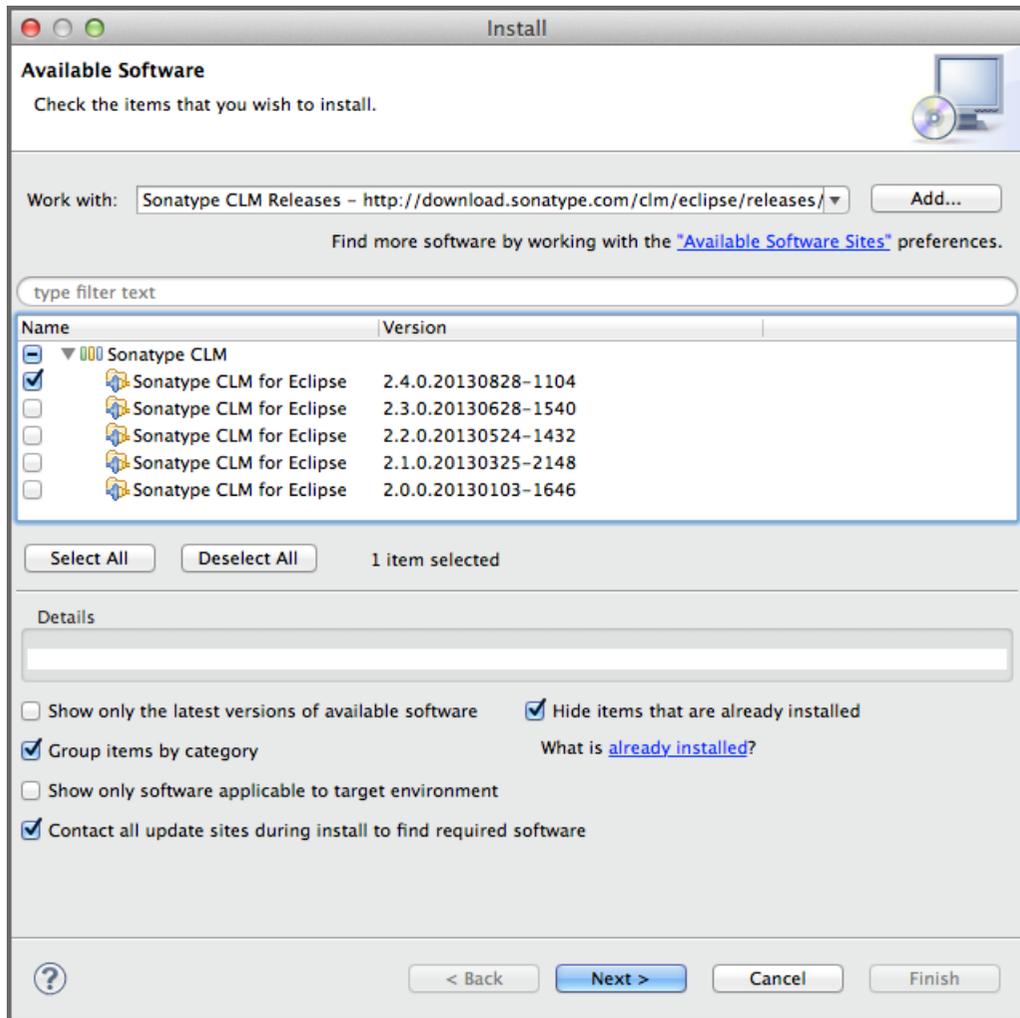


Figure 16.1: Eclipse Dialog to Install New Software with Sonatype CLM for Eclipse

URL for the Sonatype CLM for Eclipse repository

<http://download.sonatype.com/clm/eclipse/releases/>

Select the version of Sonatype CLM for Eclipse you would like to install and press *Next>*, proceed through accepting the end user license agreement and restart Eclipse to complete the installation.

Tip

Be sure to check the Section [3.5](#) before attempting to install.

16.2 Configuring Sonatype CLM for Eclipse

After successful installation of Sonatype CLM for Eclipse, you will be able to choose to show the Sonatype CLM view displayed in [Figure 16.2](#).

To access this view:

1. Choose the *Window* menu and select *Other* in the *Show View* submenu.
2. Locate the *Sonatype CLM* section with *Component Info* as shown in [Figure 16.2](#).
3. Select it and press *OK* and the view will appear in your IDE.

Tip

By typing "Compo" in the filter input, *Component Info* is automatically highlighted.

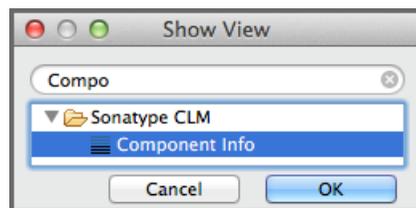


Figure 16.2: Activating the Component Info View of Sonatype CLM for Eclipse

Once the view is displayed, a warning will appear. This is because the you need to point Eclipse at your Sonatype CLM Server.

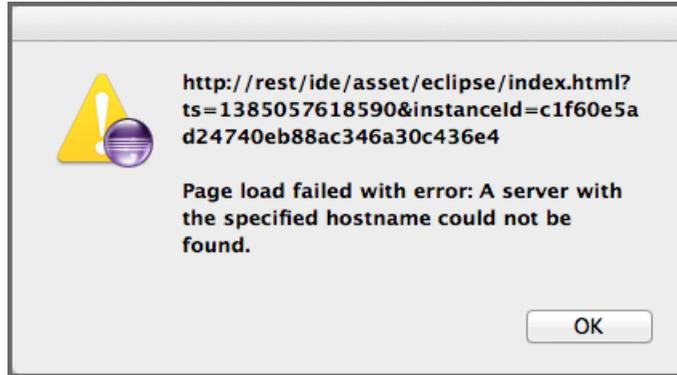


Figure 16.3: Warning after initial installation

To configure the Sonatype CLM for Eclipse plugin, simply press the  *Configure* button in the top right-hand side of the component view.

Once in Sonatype CLM for Eclipse Configuration area, there are a number of parameters you will need to complete before you can review data from Sonatype CLM. These are covered below.

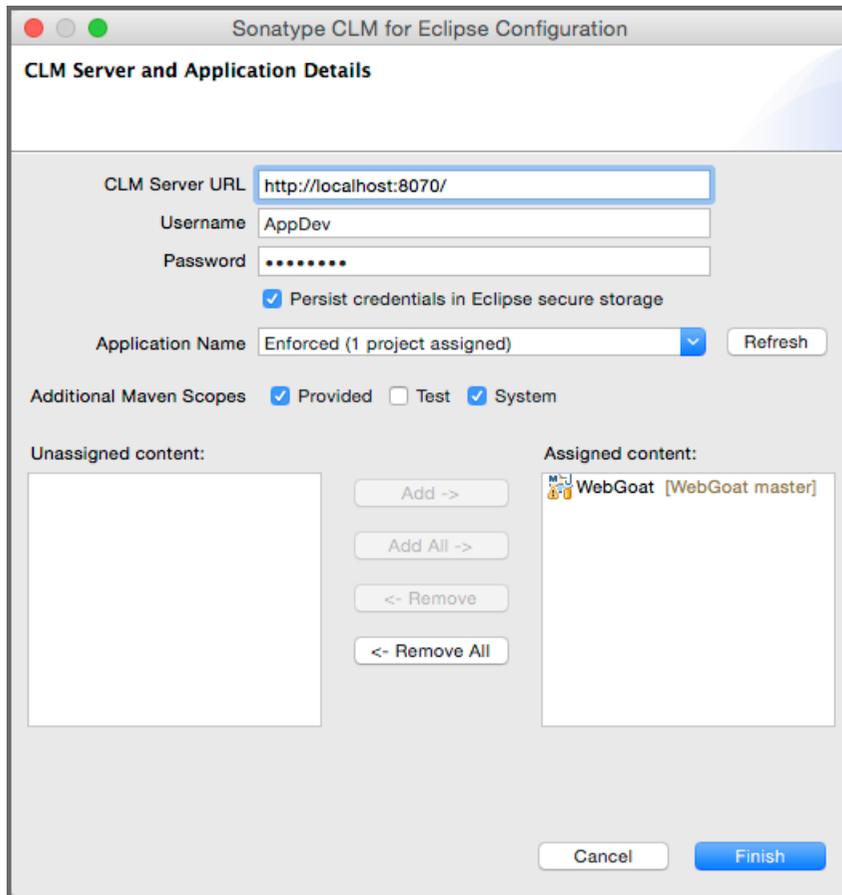


Figure 16.4: Sonatype CLM for Eclipse Configuration Dialog

CLM Server URL

The *CLM Server URL* input field has to be configured with the URL of your Sonatype CLM server.

Username and Password

This is the username and password your Sonatype CLM Administrator has assigned you. In many cases this will simply be your single sign on credentials (e.g. LDAP), though it may also be a unique username. Again, your administrator will advise you of this login information.

Note

Selecting the option to persist credentials in Eclipse secure storage will reuse your credentials after a restart. If this is not selected you will need to reenter your credentials after a restart.

Application Name

The *Application Name* is the application which has been configured in the CLM server for you. This should match the common name you associate with the application. If you don't see a name you recognize, contact your Sonatype CLM Administrator.

Note

The drop down will display a list of all available applications after pressing the *Refresh* button.

Additional Maven Scopes

The compile and runtime scopes will always be considered. Additional scopes (provided, test, and system) you would like CLM to include can also be selected.

Assigned vs. Unassigned Content

After selecting an application name that represents a collection of policies configured in your CLM server, you can determine the Eclipse projects that should be analyzed. The list on the left titled *Unassigned content* contains all projects in your current Eclipse workspace that have not been assigned to a Sonatype CLM Application. Select a project from that list and add it to the *Assigned content* list on the right by clicking the *Add* button. This will add the project to the component analysis via the CLM server. In order to perform an analysis, the project needs to be open. To select multiple projects use the Shift and Control keys, and then click the *Add* button. The *Add All*, *Remove* and *Remove All* buttons help you to control the projects to analyze for different analysis sessions.

Note

Projects can, at most, be assigned to a single application.

With a finished selection of the projects you want to analyze, press the *Finish* button and wait for the component list to be displayed in the view. Section 16.3 documents how to inspect the results of the analysis and further features available from this information.

Tip

Only open projects will be taken into account as part of the component analysis.

16.3 Using the Component Info View

Once configured and the component analysis is completed a component view will look similar to the example displayed in Figure 16.5. The list of components will reflect an analysis of the build path.

Note

For Maven projects we include the compile and runtime scopes in the CLM evaluation. If you wish to include additional dependencies found in provided, test, and system scope, these can be [configured](#).

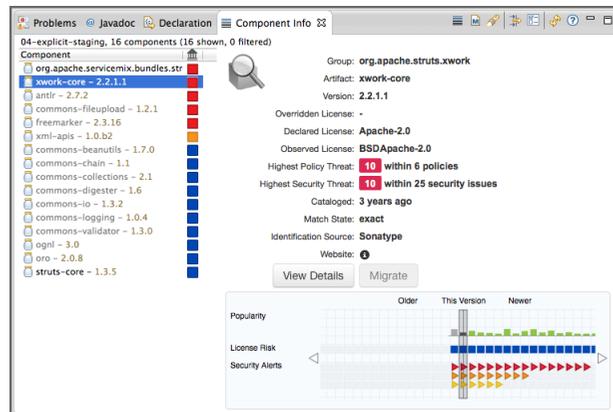


Figure 16.5: Example Component Info View

The top left-hand corner of the Sonatype CLM for Eclipse *Component Info* view displays either the number of projects currently being examined in the view, or the name of the specific project. Next to that, the number of components found, and the number of components shown in the list is displayed.

The top right-hand corner provides a number of buttons to access the following features of Sonatype CLM for Eclipse:



Open Component Details::

Opens another window with more details about the selected component including policy violations, license analysis and security issues.



Open POM

Opens the Maven pom.xml file of the selected component from the list in the Maven POM Editor.



Locate Declarations

Starts a search, that displays all usages of a selected component in the projects currently examined as documented in Section [16.5](#).



Filter

Brings up the filter selection, that lets you narrow down the number of components visible in the view as documented in. Section [16.4](#).



Configure

Activates the configuration dialog for the component analysis.



Refresh

Refreshes the component list and analysis results.



Show information about the plugin

Displays the Sonatype CLM for Eclipse support pages in an external browser.



Minimize

Minimize the view.



Maximize

Maximize the view.

The left-hand side of the view contains the list of components found in the project and identified by their artifact identifier and version number. A color indicator beside the components signals potential policy violations. The right-hand side of the view displays the details of the selected component from the list on the left.

Tip

You may notice some components are black or gray. This indicates components you have included (black) in your application, versus components that are included via a transitive dependency (gray).

By clicking on the list header on the left, the list can be ordered by the threat level of the policy a component has violated. In cases where there is no violation, the threat is simply light blue.

When you select a specific component in the list, the details, various properties, and a visualization of the different versions is displayed to the right of the list.

Tip

Depending on your screen size, the visual display may be shown below the component list. Try adjusting your screen size, or adjusting the panel.



Figure 16.6: Details for a Component in the Component Info View

The details of a specific component as displayed in Figure 16.6 include properties about the component and provide access to further features:

Group

The Maven `groupId` the component was published with. In many cases this is equivalent with the reverse domain name of the organization responsible for the deployment or running the project.

Artifact

The Maven `artifactId` of the component acts as a short and ideally descriptive name.

Version

The Maven `version` of the component. A version string ending in `-SNAPSHOT` signifies a transient, in development version, any other version is a release version.

Overridden License

The value of a license override configured in your Sonatype CLM server.

Declared License

The software license declared by the developer of the project, which in some cases, is identified during research by Sonatype, or directly from the Maven POM file.

Observed License

The licenses found by the Sonatype CLM server in a source code analysis.

Highest Policy Threat

The highest threat level policy that has been violated, as well as the total number of violations.

Highest Security Threat

The highest security threat level as well as the number of issues found with the respective level.

Patch Available

This is a future feature that will provide details in instances where a patch is available. Patches will be provided and verified by Sonatype.

Cataloged

The age of the component in the Central Repository.

Identification Source

The catalog in which a component identification match was found. This includes either a match made by Sonatype (e.g. the catalog of the Central Repository), or a match made manually (i.e. through the Sonatype CLM claiming process).

Website

If available, an information icon providing a link to the project is displayed.

View Details

Press this button to display the details view for the selected component as detailed in [Section 16.6](#).

Migrate

Press this button to start a project refactoring that allows you to change all usages of the current component to a different version as documented in [Section 16.7](#).

Custom Metadata

This is a future feature that will allow you to display all custom metadata tags assigned to the component.

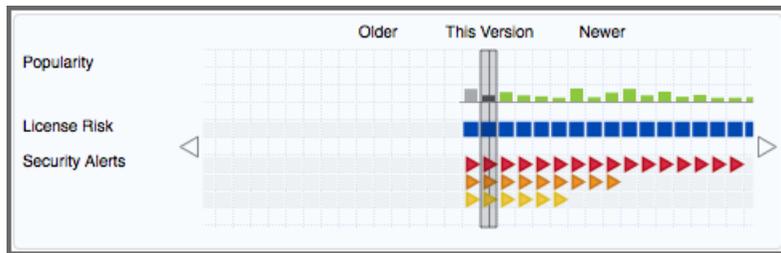


Figure 16.7: Properties of a Component for a Version Range

The visualization chart displayed in Figure 16.6 shows a number of properties for different, available versions of the selected component. Older versions are displayed on the left and newer versions on the right. Click on any section in the visualization, and all information for that particular version will be highlighted, with the specific version number at the bottom. In addition, the details for that version of the component will display in the left-hand list of properties. Arrows to the left and right of the visualization allow you to view the full range of available versions.

The properties displayed include:

Popularity

the relative popularity of a version as compared to all other component versions.

License Conflict

displays an indicator, if the observed licenses in the component are creating a legal conflict, e.g. GPL V2 and Apache V2 are not compatible for distribution of one component.

License Risk

the risk posed based on what has been set within the license threat groups. While defaults are available, these are configurable via the Sonatype CLM Server.

Security Alerts

indicators for the severity of security alerts affecting the component version.

You will likely notice a number of colors within the visualization chart. The value for each of these colors is as follows:

For Popularity

- Grey for any versions older than the current version.

- Green for newer, but within the same major version of the component.
- Blue for newer component versions, but with a greater major version than the current component.

For License and Security

- Blue - no security or license risk
- Yellow - minor security or license risk
- Orange - medium security or license risk
- Red - severe security or license risk

16.4 Filtering the Component List

The list of components found in the analysis and displayed in the component info view can be configured by pressing the  *Filter* button. The filter dialog, displayed in Figure 16.8, allows you to narrow down the components shown.

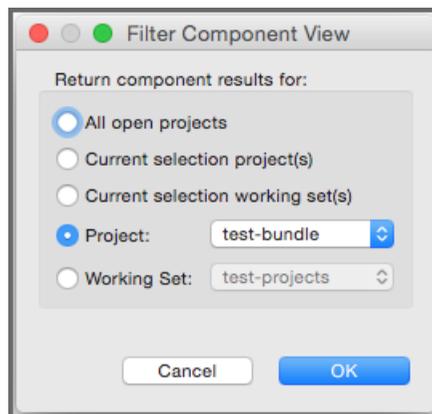


Figure 16.8: Filter Dialog for the Component Info View

The *Scope* setting determines, which projects' components are displayed in the list:

All open projects

include all the components, from all open projects.

Current selection project(s)

include the components from the project currently selected in the package explorer.

Current selection working set(s)

include the components from all the projects in the working set currently selected in the package explorer.

Project

include the components from the project selected in the drop down.

Working Set

include the components from all the projects in the working set selected in the drop down.

16.5 Searching for Component Usages

Once you have selected a specific component in the list on the left of the component info view, Sonatype CLM can determine in which projects the component is used. After pressing the  *Locate Declarations* button, and once the search has completed, you will see the results in a tree view of projects and project pom.xml files, all displayed in the *Search* window.

Inspecting this list can help you assess the impact of a potential upgrade to a new component version. Further detail is documented in Section 16.7. Looking at the found projects, you can potentially remove wrong declarations, determine side effects from transitive dependencies, or find out why this component shows up as dependency at all.

16.6 Inspecting Component Details

Press the  *Open Component Details* button in order to access the details about policy violations, license analysis and security issues for a specific component selected in the list. Figure 16.9 displays an example details view.

Policy Violations		
Policy	Constraint	Summary
Security-High	CVSS >=7 and <10	Found 2 Security Vulnerabilities with Severity >= 7
		Found 2 Security Vulnerabilities with Severity < 10
		Found 2 Security Vulnerabilities with Status OPEN
Architecture-Quality	Old	Age was 7 years, 5 months and 24 days

License Analysis		
Threat Level	Declared License(s)	Observed License(s)
Liberal	BSD-3-Clause	BSD

Security Issues			
Threat Level	Problem Code	Status	Summary
9	CVE-2007-4575	Open	HSQLDB before 1.8.0.9, as used in OpenOffice.org (OOo) 2 before 2.3.1, allows user-assisted remote attackers to execute arbitrary Java code via crafted database documents, related to "exposing static java methods."
	OSVDB-40548	Open	OpenOffice.org (OOo) HSQLDB Database Document Handling Unspecified Arbitrary Java Code Execution

Figure 16.9: Example Component Details Display

The *Policy Violations* section in the top contains a list of all the policies that have been violated by the component.

The *License Analysis* section contains the *Threat Levels* posed by the licenses declared for each component, as well as those that have been observed in the source code.

The *Security Issues* section below contains the list of issues found. They are sorted from higher to lower risk, with each issue detailed by a *Threat Level* ranging from 0 to 10, or potentially with the value *Unscored* and a descriptive text in the *Summary* column. In addition, links to the security vulnerability database entry are added as links in the *Problem Code* column.

16.7 Migrating to Different Component Versions

If you determine that a component upgrade is required to avoid a security or license issue or a policy violation, after reviewing your component usage, Sonatype CLM for Eclipse can be used to assist you in the necessary refactoring.

NOTE

This feature relies on the project being a Maven project.

The first step to start the migration is to select a newer version for the component in the visualization chart. An example is displayed in Figure 16.10.

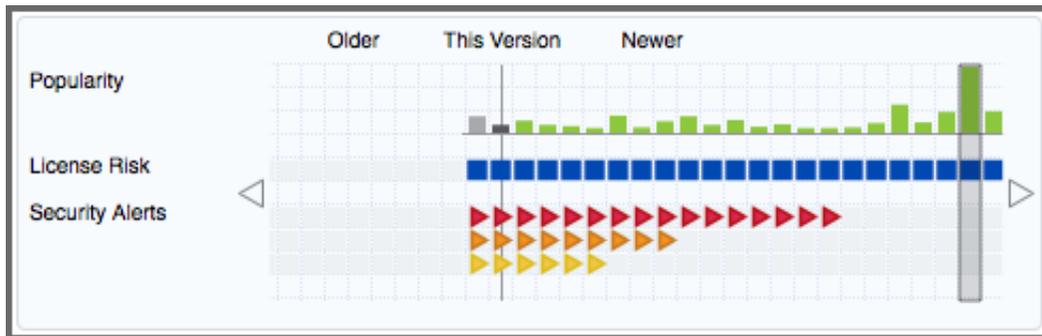


Figure 16.10: Migrating to a Newer Component Version

Once you have selected a different version than the one currently used, the *Migrate* button will become active. Pressing the button opens a dialog that assists you in the migration to the newer component. The complexity of this task varies considerably from project setup to project setup. The migration wizard is able to detect circumstances such as the component being a transitive dependency or versions managed in a property.

The simplest flow is when a dependency version can be applied, and the result is a single dialog like the one displayed in Figure 16.11.

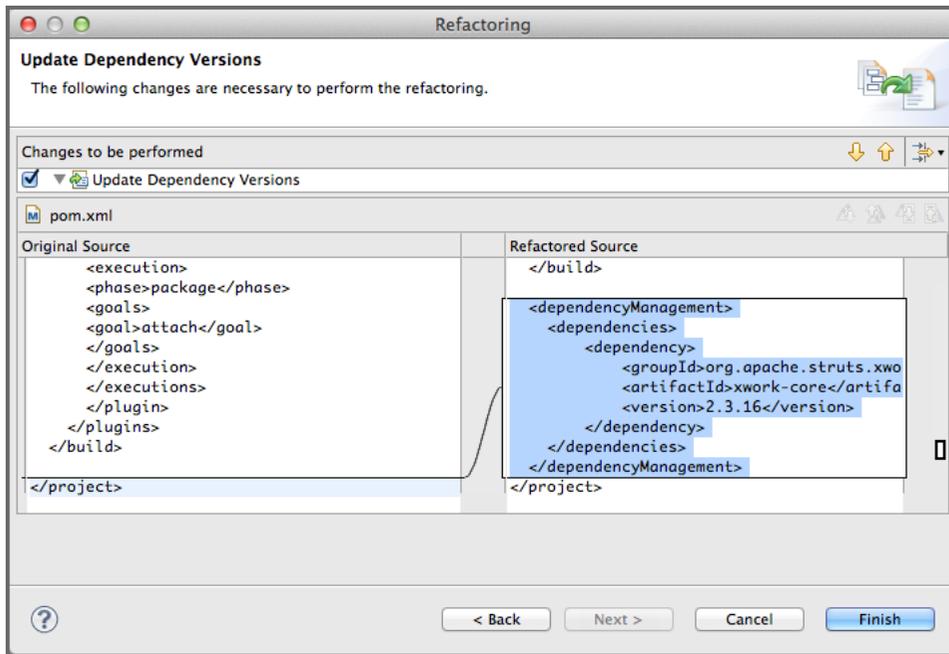


Figure 16.11: Applying a Dependency Version Upgrade

If the version is managed in a property, the initial screen from Figure 16.12 allows you to select if you want to continue with a property upgrade, or perform a replacing version upgrade.

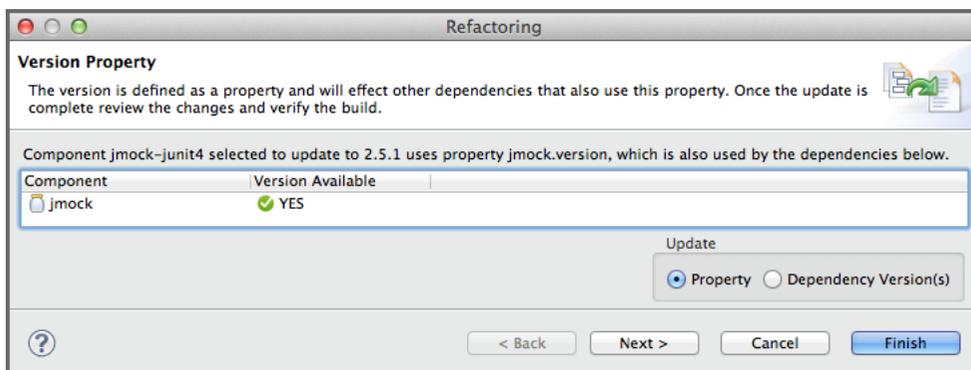


Figure 16.12: Selecting Dependency Version or Property Upgrade

Once you have selected to perform a property upgrade, you will be able to apply it in the next screen, *Refactoring*, visible in Figure 16.13.

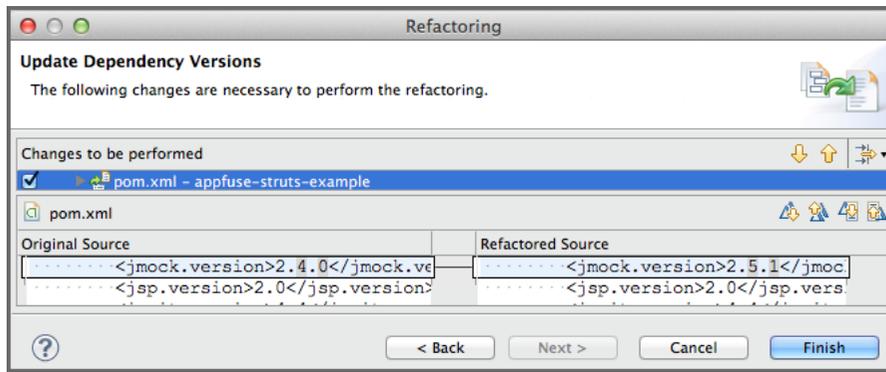


Figure 16.13: Applying a Property Upgrade

The *Refactoring* screen features navigation tools allowing you to view all potential changes in the dialog, and step through them one by one before deciding to continue.

After you have completed the refactoring of your project files, you should perform a full build, as well as a thorough test, to determine that you can proceed with the new version in your development.

Typically smaller version changes will have a higher chance of working without any major refactorings, or adaptations, of your code base and projects, while larger version changes potentially give you more new features or bug fixes.

Your release cycle, customer demands, production issues and other influencing factors, will determine your version upgrade choices. You might decide a multi-step approach, where you do a small version upgrade immediately to resolve current issues and then work on the larger upgrade subsequently to get the benefits of using a newer version. Or, you might be okay with doing an upgrade to the latest available version straight away. Potentially, a combination of approaches in different branches of your source code management system is used to figure out the best way of going forward with the upgrade.

Sonatype CLM for Eclipse and other tools of the Sonatype CLM suite can assist you through the process of upgrading, as well as monitoring, the applications after upgrade completion.

Chapter 17

Sonatype CLM for SonarQube

Sonatype CLM integrates with a wide range of external enforcement points that include [continuous integration servers](#) ([Hudson/Jenkins](#)), the [IDEs](#) ([Eclipse](#)), and [repository management](#) ([Nexus](#)). This is in addition to the Sonatype CLM [stand-alone/command line scanner](#) and [Maven plugin](#).

The enforcement points are a common aspect of the development lifecycle, and in Sonatype CLM, each represents a unique stage. This creates an invaluable integration of Sonatype CLM with industry standard tools that already make the lives of your business and development process even better. This also means, your team has greater overall control in identifying and reducing open source component risk.

Better component usage doesn't just lead to risk reduction though, it also leads to better applications. This is something that ties closely with code analysis, and tools such as [SonarQube](#).

As a user of SonarQube, you know first hand the impact that principles such as the 7 Axes of Code Quality can have on the applications and projects your teams create. Paralleling this, as a user of Sonatype CLM you also know how policy management is a critical and essential part of open source component usage.

Sonatype CLM for SonarQube brings both of these together, and in this chapter we'll cover everything you need to get going as quickly as possible. This includes:

- Download, installation, and configuration
 - Application Composition Report access
-

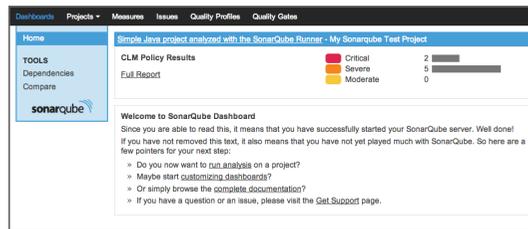


Figure 17.1: SonarQube Overview

Note

You should have installed and are running the [Sonatype CLM Server](#). You must have at least one [organization and an application](#) created, as well as at least [one policy](#) the application can be evaluated against.

17.1 Installation

As mentioned in the introduction there are a few things that must be done prior to getting the Sonatype CLM for SonarQube functional. This includes:

- Installed and configured the Sonatype CLM Server
- Created an organization, and at least one application
- Evaluated the application at least once
- Have an existing SonarQube project

With these items completed, you can download Sonatype CLM for SonarQube from our [Sonatype CLM downloads page](#).

Note

Sonatype CLM for SonarQube supports the [latest and LTS \(long-term-support\) versions](#).

Once downloaded, find the `extensions>plugins` directory in your installation of SonarQube. Next, copy the Sonatype CLM for SonarQube JAR file (the one just downloaded from the link above) into this directory. Finally, start your SonarQube instance, and log in with your administrator account.

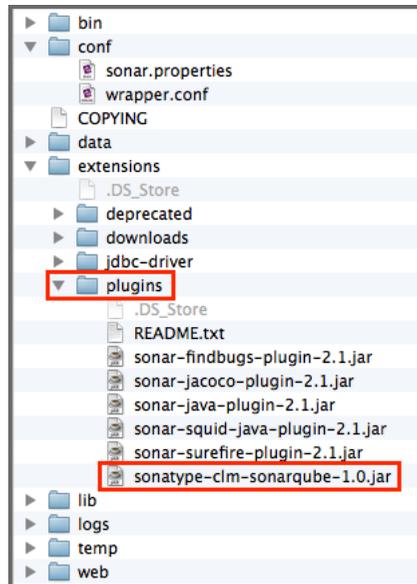


Figure 17.2: SonarQube Plugin Directory

Note

If your installation of SonarQube is running, stop it before adding the plugin.

17.2 Configuration

The Sonatype CLM Server settings allow you to specify the location of your CLM server. In the example below, basic defaults for configuration have been used. Yours will likely be different.

1. From the main SonarQube interface, click on the *Settings* menu item. This is located in the upper right corner of *SonarQube Home Page*.
-



Figure 17.3: SonarQube Settings Menu

2. On the left hand side, in the *Configuration menu*, click on *CLM Settings*.

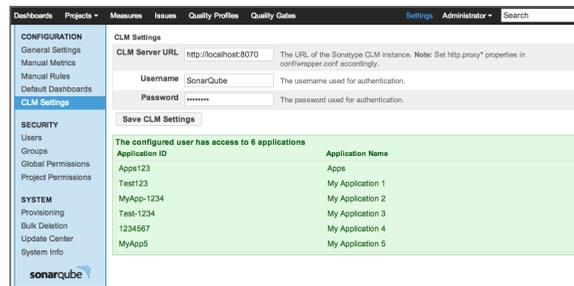


Figure 17.4: SonarQube CLM Server Settings

3. Enter the details for your Sonatype CLM Server location, as well as the user name and password that is at least a member of the developer role for the application you wish to associate with this SonarQube project.

Note

These settings will be used across all projects for your SonarQube installation. Because of this we suggest creating a single account in Sonatype CLM for SonarQube, and then associating that account with the Developer role for the applications you will be linking to SonarQube.

4. Click the *Save CLM Settings* button to save your Sonatype CLM settings, and then return to the SonarQube Dashboard home.

17.3 Proxy Configuration

In some instances, your CLM server may be setup behind a proxy. If this is the case, be sure to have your SonarQube admin configure your proxy via `wrapper.conf` file located within the `conf` directory of your SonarQube installation.

Within this file you will need to add several parameters to the *Java Additional Parameters* section. Here's an example of how this configuration might look.

```
wrapper.java.additional.7=-Dhttp.nonProxyHosts=disable-default- ↵  
    nonproxyhosts  
wrapper.java.additional.8=-Dhttp.proxyHost=192.168.2.97  
wrapper.java.additional.9=-Dhttp.proxyPort=8888
```

Tip

For more information on adding additional java properties, we suggest an internet search for *additional parameters wrapper tanukisoftware*.

17.4 Select the CLM Application

Now, you are ready to select the application that is associated with your SonarQube project.

1. Open the project you want to associate the application with, and then click the *Configuration* menu located just below the SonarQube Search field. A menu will drop down; Choose *Sonatype CLM*.

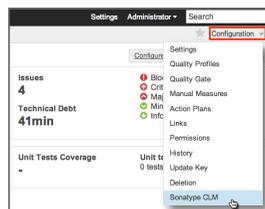


Figure 17.5: SonarQube Sonatype CLM Configuration Menu

2. You will now see the Sonatype CLM Configuration Area. This consists of a drop down menu allowing you to see all Sonatype CLM applications that the Sonatype CLM user account you entered earlier has access to. Choose the application that should be associated to your SonarQube Project.

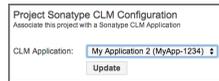


Figure 17.6: SonarQube Sonatype CLM Application Selection

3. Click the *Update* button.

17.5 Add and Configure the Sonatype CLM Widget

The final step is to add the Sonatype CLM Widget to your SonarQube project. This is done from the *SonarQube Widget Configuration area*.

1. Click on the *Configure widgets* link located just below and to the left of the SonarQube main search field (upper right of SonarQube page).



Figure 17.7: SonarQube Configure Widgets Menu

2. The easiest way to find the Sonatype CLM Report Summary widget is by using the SonarQube widget search (just enter Sonatype). Click on the *Add widget* button to add the widget to your Home page.

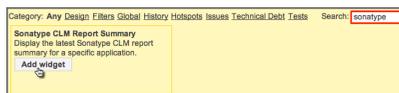


Figure 17.8: SonarQube Search for CLM Widget

3. Next, click on the *Edit* link in the top right of the Sonatype CLM Report Summary widget box. Several options will display.
 - a. Select the *Project* you want to see Sonatype CLM data in.

Tip

The option to select a project is only available when adding the widget from a non-project-specific dashboard.

- b. Enter a *Title*. This will appear above the summary information for the Sonatype CLM data.
- c. Choose the *CLM Stage*. The Sonatype CLM stage selected affects which Application Composition Report will be used to display summary-level data. Be sure to pick the stage that best represents the state of your application when it is scanned by SonarQube. *Default* will use the *Build stage*.

Note

Due to technical constraints, the dropdown option also includes stages that might not be available for your Sonatype CLM license. Selecting any of those will however yield an error when accessing the Sonatype CLM server.

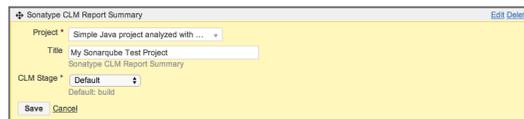


Figure 17.9: SonarQube Configure Sonatype CLM Widget options

4. Click the *Save* button to save your selections.

17.6 Accessing the Application Composition Report

Within SonarQube, you will be provided with basic summary information for the Application, as well as a link to the associated Application Composition Report. To access the detailed information provided by this report, click on the *Full Report* link displayed in the Sonatype CLM Report Summary widget in your project.

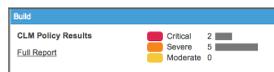


Figure 17.10: SonarQube Sonatype CLM Widget Example

Chapter 18

Sonatype CLM for CLI

While the best integration between Sonatype CLM is through the fully supported tools (e.g. Nexus, Eclipse, and Hudson/Jenkins), any application can be evaluated against your Sonatype CLM policies simply by using Sonatype CLM for CLI (aka Stand-alone).

Before you get started, there are a few things you should make sure you have, including:

- A general familiarity with the CLI (you don't need to be an expert, but basic knowledge helps).
- Installed and set up the Sonatype CLM Server
- Set up an organization and application
- Created or imported a policy for your application, or the organization that contains your application.

Note

With regard to integrating Sonatype CLM for CLI with CI servers, you should also have a familiarity with the way your particular CI utilizes build steps to launch a simple script.

18.1 Downloading Sonatype CLM for CLI

To use Sonatype CLM for CLI, first download the jar file named similar to `sonatype-clm-scanner-1.7.0-02.jar` from the [Sonatype Support website](#) and place the file in its own directory.

Note

Be sure to remember where you placed the file you downloaded. As a recommendation, it's best to have Sonatype CLM for CLI in its own directory, and not shared with the Sonatype CLM Server.

18.2 Locating Your Application Identifier

Before you start an evaluation, you need to have your Application ID. To get this...

1. Log into your Sonatype CLM server with a user account with at least a developer role for the application you plan on evaluating.
2. Click the Manage Organizations and Applications icon .
3. Click on *Applications* in the menu on the left to display a list of applications, and then click on the application you want to find the Application ID for. You will see a screen similar to Figure 18.1
4. Locate the text underneath the application name. The application identifier is the text between *Application* and *in [organization name]*

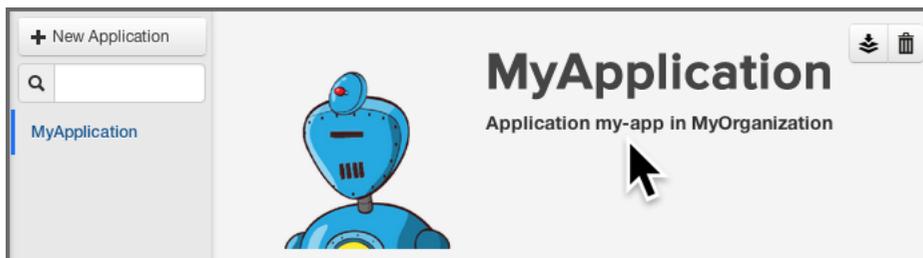


Figure 18.1: Application Overview and Application Identifier

18.3 Evaluating an Application

Now that you have Sonatype CLM for CLI set up, you are ready to evaluate an application. As a Java application, it can be started using the `java` command, and adding the necessary parameters. The syntax below represents the minimum set of options required to evaluate an application:

```
java -jar [scanner jar] -i [application id] -s [server URL] [target]
```

scanner jar

This is the path to Sonatype CLM for CLI (scanner) jar file e.g. `./sonatype-clm-scanner.jar`.

--authentication

Using the switch `-a`, enter the user name:password (e.g. `MyUserName:MyUserPassword`).

Note

Authentication will permit (or prevent) the ability to submit an application for evaluation, as well as retrieve the summary results and URL. At this time, it is not required.

--application-id

Using the switch `-i`, enter the application id for your application (see instructions above).

--server-url

Using the switch `-s` enter the location of your CLM server (e.g. `http://localhost:8070`).

Target

This is the path to a specific application archive file or a directory containing such archives. A number of formats are supported, including `jar`, `war`, `ear`, `tar`, `tar.gz`, `zip` and many others.

Tip

Listed in the options below, you can specify the specific CLM stage. However, if you do not include this option the system will default to the Build stage.

18.3.1 Additional Options

There are several additional options that can be used in the construction of the syntax for evaluating an application with Sonatype CLM for CLI.

--fail-on-policy-warnings

using the switch `-w` will cause a failure of the evaluation if any warnings are encountered. By default, this is set to false.

--ignore-system-errors

Using the switch `-e`, allows you to ignore any system errors (e.g. IO, Network, server, etc.). This is most helpful when using Sonatype CLM for CLI with continuous integration servers, as these errors can cause the unintentional failure of a build.

--proxy

Using the switch `-p`, you can specify a proxy to use in connecting to the CLM Server. The format is `<host[:port]>`.

--proxy-user

Using the switch `-U`, you can specify credentials for the proxy. The format is `<username:password>`.

--result-file

Using the switch `-r`, you can specify the name and location of a JSON file that will store the results of the policy evaluation in a machine-readable format.

--stage

Using the switch `-t`, you can specify the Sonatype CLM stage you wish the report to be associated with. This is an optional parameter, and if it is not specified, the report will be associated with the Build stage by default.

Note

At this time only the Build, Stage Release, and Release stages will display a report in the CLM Reports Dashboard. For a full list of stages, use the CLI help provided with the tool.

18.4 Example Evaluation

In an example scenario, let's say you have copied the `sonatype-clm-scanner.jar` as well as the application you want to examine to a specific directory e.g. `~/clm-test`. The application's filename is `sample-application.zip`.

To evaluate this application you have to identify the Sonatype CLM Application ID and supply it with the `-i` switch as well as supply the URL of your CLM server with `-s`. As an option, and what is demonstrated below, you can also specify a particular stage.

The full command line for an Application ID `Test123` and a URL of `http://localhost:8070` is

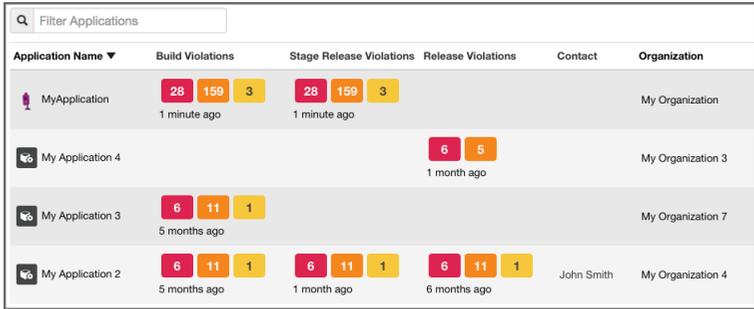
```
java -jar ./sonatype-clm-scanner.jar -i Test123 -s http://localhost:8070
-t release sample-application.zip
```

To access help content for Sonatype CLM for CLI, run it without supplying parameters:

```
java -jar ./sonatype-clm-scanner.jar
```

Go ahead and try an evaluation yourself. Sonatype CLM for CLI will accept a number of file types, including jar, war, and zip files. If your evaluation is successful, the log output of the command execution will provide a summary as well as a link to the produced results similar to:

```
[INFO] Policy Action: Warning
[INFO] Summary of policy violations: 4 critical, 85 severe, 46 moderate
[INFO] The detailed report can be viewed online
at http://localhost:8070/ui/links/application/my-app/report/95c4c14e
```



Application Name	Build Violations	Stage Release Violations	Release Violations	Contact	Organization
MyApplication	28 159 3 1 minute ago	28 159 3 1 minute ago			My Organization
My Application 4			6 5 1 month ago		My Organization 3
My Application 3	6 11 1 5 months ago				My Organization 7
My Application 2	6 11 1 5 months ago	6 11 1 1 month ago	6 11 1 6 months ago	John Smith	My Organization 4

Figure 18.2: Violations Report After an Evaluation

Note

If using Sonatype CLM for CLI, and you kept our defaults, the report will be listed under Build Violations. You should see something similar to the results displayed in Figure 18.2.

18.5 Using Sonatype CLM for CLI with a CI Server

We won't be covering a specific CI here, but in general, all you need to identify (in your CI), is the location for adding a build step that includes processing a simple shell script during the building of your application.

Once you are there, make sure your script calls Sonatype CLM for CLI using the following syntax:

```
java -jar [ScannerJar] -i [AppID] -e [IgnoreSystemErrors] -w [ ←  
FailOnPolicyWarning] -s [ServerURL] [Target]
```

Each of the areas in the syntax above have been described in the previous section [Evaluating an Application](#).

Given a typical setup, your syntax, including all available options will likely look similar to this:

```
java -jar /scanner/sonatype-clm-scanner.jar -i tester123 -s  
http://localhost:8070 ./target/sample-app.war
```

Now, when your application is built, the build step you have added will call Sonatype CLM for CLI, evaluate your application, and upload results of the evaluation to Sonatype CLM Server. By default this will be placed below the build column in the Reports and Application area on the Sonatype CLM Server, for your application.

Note

We advise you to use a separate application identifier for each of your unique applications. Using the same application identifier will result in report results being overwritten each time an application is built. While this is always the case, matching the latest evaluation to the right application can prove difficult.

Chapter 19

Sonatype CLM for Maven

Sonatype CLM for Maven allows users to evaluate any Maven-based software projects, in the same way our integrated tools (e.g. Nexus Pro, Eclipse, Hudson/Jenkins) do.

This means you can access the same robust, reporting features no matter what toolset you use. It can be run on a command line interface and can therefore be executed on any continuous integration server, as well as a number of popular IDEs.

How you use the Sonatype CLM for Maven plugin widely depends on how you enforce policy. However, in general, when using the plugin on a multi-module project in most cases you will only configure an execution for the modules that produce components that will be deployed as an application.

Typically these are `ear` files or `war` files for deployment on application servers or `tar.gz` or other archives that are used for production deployments or distribution to users. That said, you can also analyze all modules of a project. Again, this will largely depend on what your CLM policy is enforcing and what you want to validate.

In the following sections, we'll provide details on these goals and their usage:

Index

The `index` goal of the plugin allows you to prepare data for analysis with Sonatype CLM for CI.

Attach

The `attach` goal aids your integration with Sonatype Nexus CLM Edition and the release process

using the staging tools of Nexus.

Evaluate

The `evaluate` goal can trigger an evaluation directly against a Sonatype CLM server.

Note

The `help` goal provides documentation for all the goals and parameters and you can invoke it with an execution like `mvn com.sonatype.clm:clm-maven-plugin:help`.

19.1 Evaluating Project Components with Sonatype CLM Server

The `evaluate` goal scans the dependencies and build artifacts of a project and directly submits the information to a Sonatype CLM Server for policy evaluation.

If a policy violation is found and the CLM stage is configured to `Fail`, the Maven build will fail. If invoked for an aggregator project, dependencies of all child modules will be considered.

The `evaluate` goal requires the Sonatype CLM Server URL as well as the application identifier to be configured. Optionally a CLM stage can be configured.

The command line arguments are:

`clm.serverUrl`

the URL for the CLM server, this parameter is required.

`clm.serverId`

used for authentication and must match the id given to the CLM Server [specified in your Maven settings](#).

`clm.username`

the username used to authenticate access to the CLM server.

Note

This is not required when using `clm.serverId`, but can be used to overwrite those settings.

clm.password

the password for the username indicated above.

Note

This is not required when using `clm.serverId`, but can be used to overwrite those settings.

clm.applicationId

the application identifier for the application to run policy against, this parameter is required

clm.resultFile

the path for specifying the location of a JSON file where the following information will be stored:

- `applicationId` : Application ID
- `scanId` : Organization ID
- `reportHtmlUrl` : URL to the HTML version of the report
- `reportPdfUrl` : URL to the PDF version of the report
- `reportDataUrl` : URL to the Data version of the report (for use via CURL, or similar tool)

clm.stage

the stage to run policy against with the possible values of `develop`, `build`, `stage-release`, `release` and `operate` with a default value of `build`.

clm.additionalScopes

the additional scopes you would like CLM to include components from during the evaluation. Values include `test`, `provided`, and `system`. In cases where you want to include more than one of these, separate the list using a comma (see examples below).

An example invocation is:

```
mvn com.sonatype.clm:clm-maven-plugin:evaluate -Dclm.additionalScopes=test ↵  
    ,provided,system -Dclm.applicationId=test -Dclm.serverUrl=http:// ↵  
    localhost:8070
```

You can avoid specifying the parameters on the command line by adding them to your `settings.xml` or `pom.xml` as `properties`.

```
<properties>  
  <clm.serverUrl>http://localhost:8070</clm.serverUrl>  
  <clm.applicationId>test</clm.applicationId>  
</properties>
```

Sonatype CLM for Maven can be executed against an aggregator project. When executed in an aggregator project, it calculates the dependencies and transitive dependencies of all child modules and takes all of them into account for the policy evaluation.

The evaluate goal logs its activity and provides the location of the generated report.

```
[INFO] --- clm-maven-plugin:2.1.1:evaluate (default) @ test-app ---
[INFO] Starting scan...
[INFO] Scanning ../repository/org/codehaus/plexus/plexus-utils/3.0/plexus- ←
      utils-3.0.jar...
[INFO] Scanning ../repository/org/apache/maven/maven-settings/3.0/maven- ←
      settings-3.0.jar...
[INFO] Scanning target/test-app-1.0-SNAPSHOT.jar...
[INFO] Saved module scan to /opt/test-app/target/sonatype-clm/scan.xml.gz
[INFO] Uploading scan to http://localhost:8070 ...
[INFO] Evaluating policies... (ETA 5s)
[INFO] Policy Action: None
Summary of policy violations: 0 critical, 0 severe, 0 moderate
The detailed report can be viewed online at
http://localhost:8070/ui/links/application/test/report/f4582a1570634dc2ac8
```

Note

The evaluate goal cannot be bound to a lifecycle phase.

After a successful build the report can be accessed in the Sonatype CLM server under the application that was configured. A direct link is provided on the log.

19.1.1 Authentication

To configure authentication to the CLM Server, you will need to add your Sonatype CLM Server information to your Maven `settings.xml` file:

```
<settings>
  ...
  <servers>
    <server>
      <id>clm_server</id>
      <username>my__clm_login</username>
      <password>my_clm_password</password>
    </server>
```

```
...
</servers>
...
</setting>
```

Note

In our example we have not encrypted our password. This is generally recommended. The Apache Maven project provides [instructions](#) for password encryption. Additionally, username and password can still be specified at the command line, and will be used in place of these settings.

19.1.2 Simplifying Command Line Invocations

If you happen to use the plugin frequently by running it manually on the command line and want to shorten the command line even more, you can add a plugin group entry to your Maven `settings.xml` file:

```
<settings>
...
<pluginGroups>
  <pluginGroup>com.sonatype.clm</pluginGroup>
...
</pluginGroups>
...
</settings>
```

This enables you to invoke the plugin using its shorthand prefix form:

```
mvn ... clm:index
```

19.1.3 Skipping Executions

The `clm.skip` parameter can be used, when a CLM plugin execution is configured in your project's `pom.xml` file, but you want to avoid the execution for a particular build. An example execution is:

```
mvn clean install -Dclm.skip=true
```

The parameter can also be set in your IDE configuration for Maven build executions or as a property in your `settings.xml` or `pom.xml`:

```
<properties>
  <clm.skip>true</clm.serverUrl>
</properties>
```

19.2 Creating a Component Index

When evaluating a Maven-based software project, Sonatype CLM for Maven can take advantage of the dependency information contained in the project's `pom.xml` files and the information about transitive dependencies available to Maven.

The `index` goal of Sonatype CLM for Maven allows you to identify component dependencies and makes this information available to Sonatype CLM CI tools (e.g. Sonatype CLM for Hudson/Jenkins or Bamboo). You can invoke an execution of the `index` goal manually as part of your command line invocation by executing the `index` goal after the `package` phase:

```
mvn clean install com.sonatype.clm:clm-maven-plugin:index
```

Alternatively you can configure the execution in the `pom.xml` files `build` section or in a profile's `build` section.

```
<build>
  <plugins>
    <plugin>
      <groupId>com.sonatype.clm</groupId>
      <artifactId>clm-maven-plugin</artifactId>
      <version>2.1.1</version>
      <executions>
        <execution>
          <goals>
            <goal>index</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

With the above configuration a normal Maven build execution with e.g. `mvn clean install` will trigger the CLM plugin to be executed in the `package` phase and result in a log output similar to

```
[INFO] --- clm-maven-plugin:2.1.1:index (default) @ test-app ---  
[INFO] Saved module information to /opt/test-app/target/sonatype-clm/ ↵  
    module.xml
```

If you want to manually configure the lifecycle phase to execute the plugin, you have to choose a phase after `package`.

The generated `module.xml` file contains the information that will be picked up by Sonatype CLM for CI and incorporated into the CLM evaluation. This improves the analysis since Sonatype CLM for Maven is able to create a complete dependency list rather than relying on binary build artifacts.

Note

By default only dependencies in the `compile` and `runtime` scopes will be considered, since this reflects what other Maven packaging plugins typically include. Dependencies with the scopes `test`, `provided` and `system` must be manually added, and are described in the [Evaluating Project Components with Sonatype CLM Server](#) section.

19.2.1 Excluding Module Information Files in Continuous Integration Tools

When using the Sonatype CLM Maven plugin and the `index` goal, module information files are created. If desired, you can exclude some of the modules from being evaluated. For example, you may want to exclude modules that support your tests, and don't contribute to the distributed application binary.

The default location where the module information files are stored is `${project.build.directory}/sonatype-clm/module.xml`.

In the supported CI tool, you will see a section labeled *Module Excludes*. On this area, use a comma-separated list of [Apache Ant styled patterns](#) relative to the workspace root that denote the module information files (`**/sonatype-clm/module.xml`) to be ignored.

Here's an example of the pattern described above:

```
**/my-module/target/**, **/another-module/target/**
```

If unspecified, all modules will contribute dependency information (if any) to the evaluation.

19.3 Creating a Component Info Archive for Nexus Pro CLM Edition

The `attach` goal scans the dependencies and build artifacts of a project and attaches the results to the project as another artifact in the form of a `scan.xml.gz` file. It contains all the checksums for the dependencies and their classes and further meta information and can be found in the `target/sonatype-clm` directory. A separate `scan.xml.gz` file is generated for each maven module in an aggregator project in which the plugin is executed.

This attachment causes the file to be part of any Maven `install` and `deploy` invocation. When the deployment is executed against a Sonatype Nexus CLM Edition server the artifact is used to evaluate policies against the components included in the evaluation.

To use this goal, add an execution for it in the POM, e.g. as part of a profile used during releases:

```
<build>
  <plugins>
    <plugin>
      <groupId>com.sonatype.clm</groupId>
      <artifactId>clm-maven-plugin</artifactId>
      <version>2.1.1</version>
      <executions>
        <execution>
          <goals>
            <goal>attach</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

Once configured in your project, the build log will contain messages similar to

```
[INFO] --- clm-maven-plugin:2.1.1:attach (default) @ test-app ---
[INFO] Starting scan...
[INFO] Scanning ...plexus-utils-3.0.jar
[INFO] Scanning ...maven-settings-3.0.jar...
[INFO] Scanning target/test-app-1.0-SNAPSHOT.jar...
[INFO] Saved module scan to /opt/test-app/target/sonatype-clm/scan.xml.gz
```

The attachment of the `scan.xml.gz` file as a build artifact causes it to be stored in the local repository as well as the deployment repository manager or the Nexus staging repository ending with `-sonatype-clm-scan.xml.gz`. This file will be picked up by Sonatype Nexus CLM Edition and used in the policy analysis during the staging process. It improves the analysis since Sonatype CLM for Maven is able to create a complete dependency list rather than relying on binary build artifacts.

19.4 Using Sonatype CLM for Maven with Other IDEs

While the integration with Eclipse offered by Sonatype CLM for IDE is the most powerful tooling for developers available, user of other popular integrated development environments are not left without support. All common Java IDEs have powerful integration with Apache Maven and therefore can be used together with Sonatype CLM for Maven to evaluate projects against your Sonatype CLM server.

This chapter showcases the integration with [IntelliJ IDEA from JetBrains](#) and [NetBeans IDE from Oracle](#).

19.4.1 Maven Plugin Setup

In our example setup for the usage with other IDE's we are going to add a plugin configuration for Sonatype CLM for Maven into the `pom.xml` file of the project we want to analyze as documented in [Example Configuration of Sonatype CLM for Maven](#). This configuration defines the `serverUrl` of the CLM server to be contacted for the evaluation, the `applicationId` used to identify the application in the CLM server to evaluate against and the `stage` configuration to use for the evaluation.

Example Configuration of Sonatype CLM for Maven

```
<build>
  <pluginManagement>
    <plugins>
      <plugin>
        <groupId>com.sonatype.clm</groupId>
        <artifactId>clm-maven-plugin</artifactId>
        <version>2.1.1</version>
        <configuration>
          <serverUrl>http://localhost:8070</serverUrl>
          <applicationId>test</applicationId>
          <stage>develop</stage>
        </configuration>
      </plugin>
    </plugins>
  </build>
```

```
</pluginManagement>  
</build>
```

With this configuration in place a user can kick off an evaluation with the command line `mvn package clm:evaluate`.

This will result in an output detailing the components to be analyzed, any policy violations and a link to the resulting report in the Sonatype CLM server.

Note

To speed the build up you can skip the test compilation and execution by passing `-Dmaven.test.skip=true` on the command line invocation, since it is not needed for the CLM evaluation.

19.4.2 IntelliJ IDEA

IntelliJ IDEA supports Maven projects natively and you can simply open a project in the IDE by opening the `pom.xml` file.

Once your project is opened and you have added the plugin configuration for Sonatype CLM for Maven from [Example Configuration of Sonatype CLM for Maven](#), you can create a configuration to run the desired Maven command.

Select *Edit Configurations* from the *Run* menu, press the + button and select *Maven* to add a new configuration. Enter the command line and other desired details as displayed in [Figure 19.1](#).

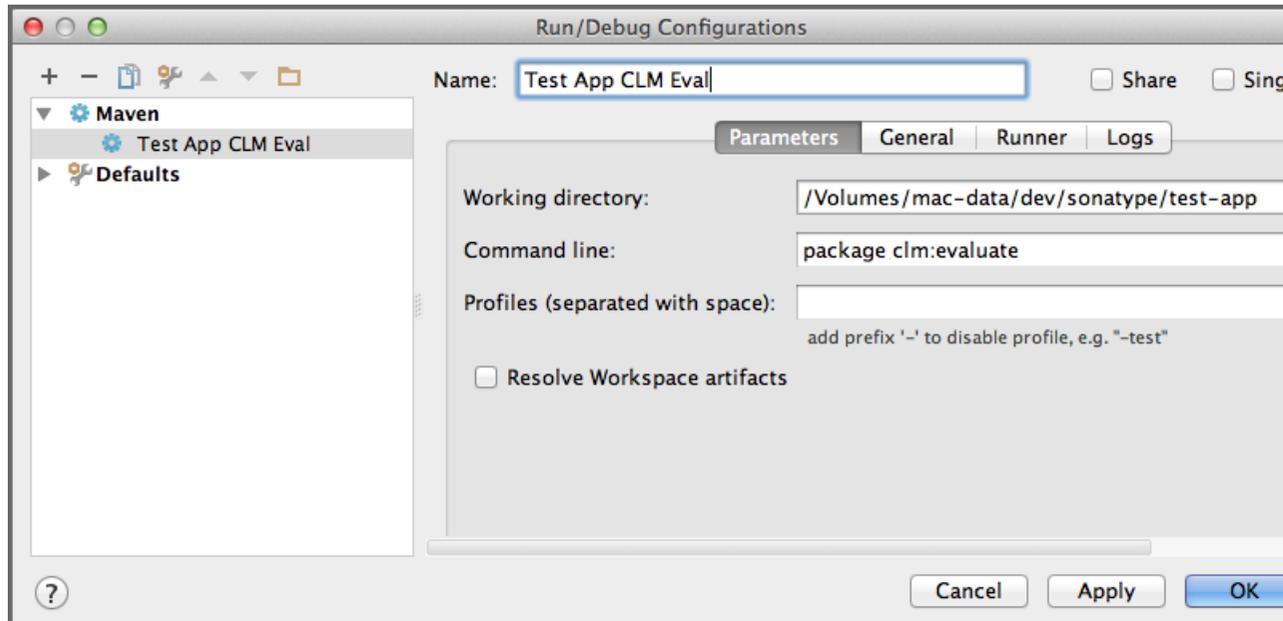


Figure 19.1: Creating a Maven Run Configuration for a CLM Evaluation in IntelliJ

After pressing *OK* in the dialog the new configuration will be available in the run configuration drop down as well the *Maven Projects* view. You can open the view using the *View* menu, selecting *Tools Window* and pressing *Maven Projects*. You will see the window appear in the IDE looking similar to Figure 19.2. It displays the run configuration selector with the green play button on the top as well as the Maven project with the CLM evaluation run configuration.

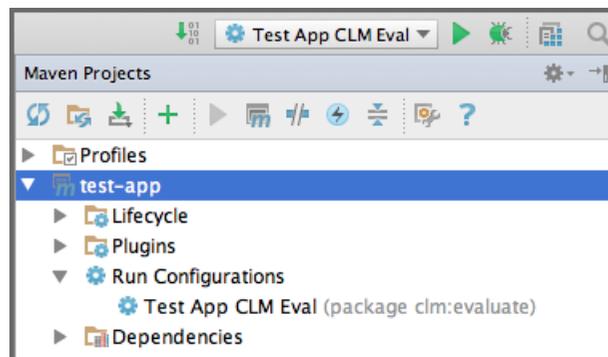
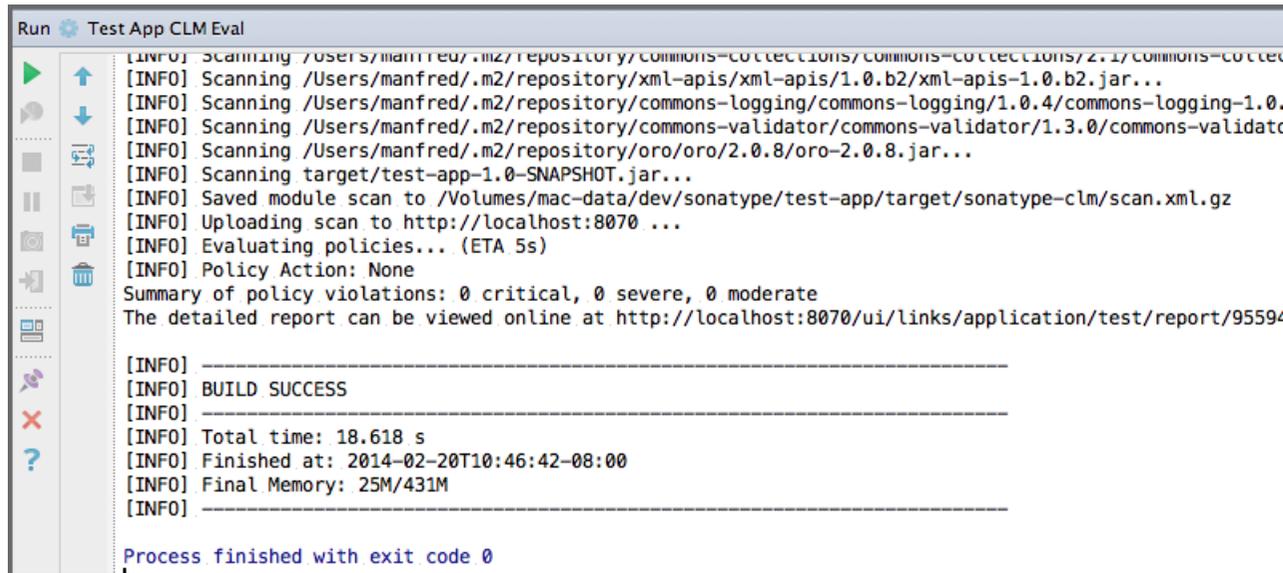


Figure 19.2: Maven Projects View with the CLM Evaluation Run Configuration in IntelliJ

You can press the green play button beside the run configuration, or the configuration entry itself in the *Maven Projects* window, to start a build. The build will run in an embedded console window in the IDE as displayed in Figure 19.3, and show all the output from the Maven build including any error messages and the link to the produced report in the Sonatype CLM server. Policy violations can be configured to result in a build failure.

The image shows a screenshot of the IntelliJ IDE's Run Console window. The window title is "Run Test App CLM Eval". On the left side, there is a vertical toolbar with various icons, including a green play button at the top. The main area of the console displays the following text:

```
[INFO] Scanning /Users/manfred/.m2/repository/commons-collections/commons-collections/2.1/commons-collections-2.1.jar...
[INFO] Scanning /Users/manfred/.m2/repository/xml-apis/xml-apis/1.0.b2/xml-apis-1.0.b2.jar...
[INFO] Scanning /Users/manfred/.m2/repository/commons-logging/commons-logging/1.0.4/commons-logging-1.0.4.jar...
[INFO] Scanning /Users/manfred/.m2/repository/commons-validator/commons-validator/1.3.0/commons-validator-1.3.0.jar...
[INFO] Scanning /Users/manfred/.m2/repository/oro/oro/2.0.8/oro-2.0.8.jar...
[INFO] Scanning target/test-app-1.0-SNAPSHOT.jar...
[INFO] Saved module scan to /Volumes/mac-data/dev/sonatype/test-app/target/sonatype-clm/scan.xml.gz
[INFO] Uploading scan to http://localhost:8070 ...
[INFO] Evaluating policies... (ETA 5s)
[INFO] Policy Action: None
Summary of policy violations: 0 critical, 0 severe, 0 moderate
The detailed report can be viewed online at http://localhost:8070/ui/links/application/test/report/95594...

[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 18.618 s
[INFO] Finished at: 2014-02-20T10:46:42-08:00
[INFO] Final Memory: 25M/431M
[INFO] -----

Process finished with exit code 0
```

Figure 19.3: CLM for Maven Output in the Run Console in IntelliJ

19.4.3 NetBeans IDE

NetBeans IDE supports Maven projects natively and you can simply open a project in the IDE by choosing *Open Project* from the *File* menu and navigating to the directory that contains your project.

Once your project is opened, you can expand the *Project Files* section in the *Projects* window as displayed in Figure 19.4. Double-click on the `pom.xml` file and add the plugin configuration for Sonatype CLM for Maven from [Example Configuration of Sonatype CLM for Maven](#).

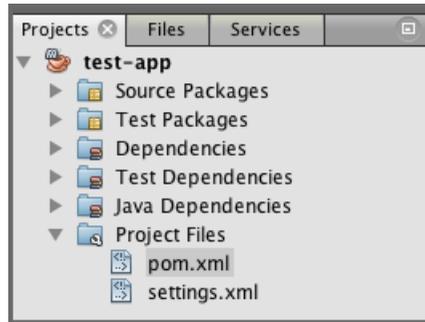


Figure 19.4: Project View with the `pom.xml` in NetBeans

If you right-click on the `pom.xml` file, you can choose *Run Maven* and *Goals*, to display the dialog displayed in Figure 19.5. Enter the configuration as displayed and don't forget to select *Remember as:* providing a name. This will allow you to simply start this defined configuration from the *Run Maven* context menu of the `pom.xml` file.

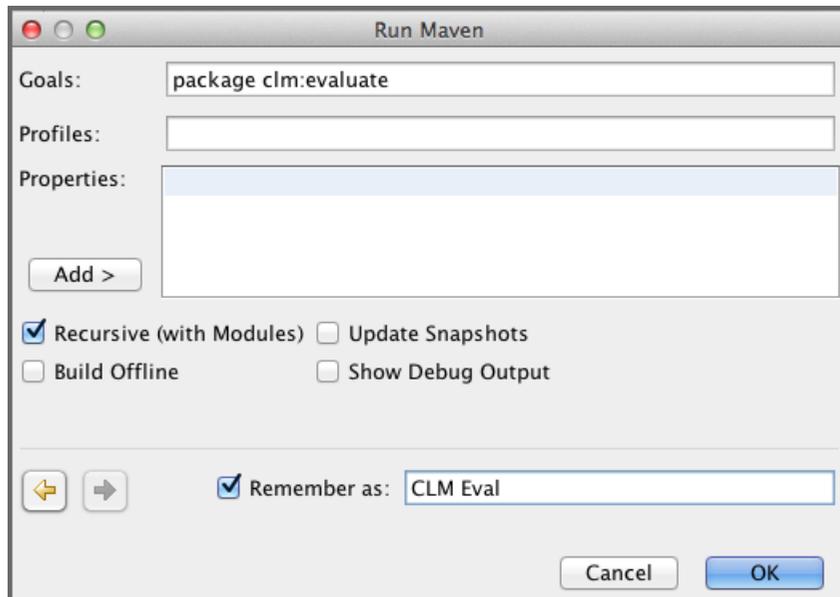
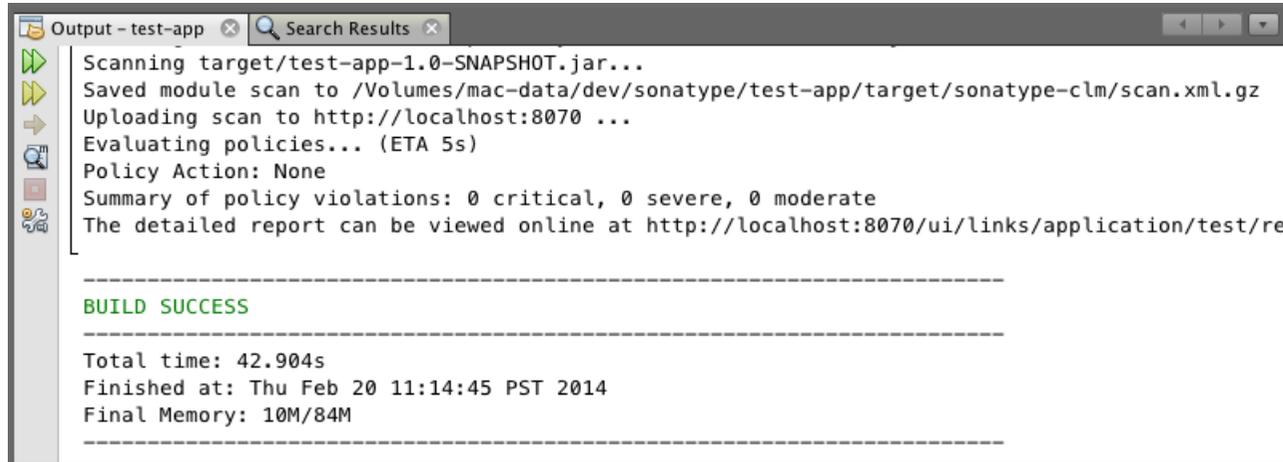


Figure 19.5: Maven Goal Setup for a CLM Evaluation in NetBeans

After pressing *OK* the defined Maven execution will start and display the output including any error messages and the link to the produced report in the Sonatype CLM server in the Output window displayed in Figure 19.6. Policy violations can be configured to result in a build failure.

The image shows a screenshot of the NetBeans IDE's Output Window. The window title is "Output - test-app" and it contains a search bar labeled "Search Results". The output text is as follows:

```
Scanning target/test-app-1.0-SNAPSHOT.jar...
Saved module scan to /Volumes/mac-data/dev/sonatype/test-app/target/sonatype-clm/scan.xml.gz
Uploading scan to http://localhost:8070 ...
Evaluating policies... (ETA 5s)
Policy Action: None
Summary of policy violations: 0 critical, 0 severe, 0 moderate
The detailed report can be viewed online at http://localhost:8070/ui/links/application/test/re

-----
BUILD SUCCESS
-----

Total time: 42.904s
Finished at: Thu Feb 20 11:14:45 PST 2014
Final Memory: 10M/84M
-----
```

Figure 19.6: CLM for Maven Output in the Output Window in NetBeans

Chapter 20

Sonatype CLM REST APIs

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.

**Warning**

Sonatype CLM APIs are versioned. This document represents a collection of both deprecated version (v1) and current (v2) REST APIS. We highly recommend updating to the latest version of Sonatype CLM and using the latest REST API. This warning appears in each section as well.

20.1 Component Search REST APIs (v1)



Warning

Sonatype CLM APIs are versioned. This document represents a deprecated version (v1) and we highly recommend updating to the latest version of Sonatype CLM and using the [latest version of this API](#).

You may reach a point where you would like to search for a particular component that has been identified during a CLM evaluation. In other words, you have already completed an evaluation (more likely you have done several or many evaluations), and either the CLM server has identified, or you have claimed, at least one component in that application.

The likely scenario is that CLM has identified many components across all of your applications. Even more likely, a few of the same components show up in several (maybe all) of your applications. In these instances, the Sonatype CLM Component Search API allows you to both find a particular component, as well as get information back about that component.

This API consists of GET requests, which allow you to retrieve component information such as application ID, application name, report URL, component hash, component coordinates, and perhaps most importantly, it provides the highest threat level of the policy violations (for the found component).

Below, we've provided an example of the GET request. We've done this using the HTTP client cURL. Of course, you could use any HTTP client tool. Additionally, to help demonstrate use of the API, we've broken out the various pieces for this request and provided an example of data that is retrieved.

Note

Compared to the other APIs, Component Search is fairly simple. However, you should have some basic information about the component coordinates, as well as the Sonatype CLM stage (e.g. build) where the component was analyzed.

Searching for Components

First, make sure your Sonatype CLM server is started. Also, as we mentioned, you will need to have evaluated at least one application. With those two things completed, let's take a look at the GET API call.

```
GET /api/v1/search/component
```

Now, in addition to this, you will need to set the stage, and then add your search parameters. Let's take a look at stage first.

Stage

Typically the stage represents the development lifecycle of your product. There are four stages that are currently supported. These include:

- build
- stage
- stage-release
- operate

Entering any of these for the stage ID will pull from that specific stage's evaluation data.

Next up, you need to set the component search parameters using any combination of these options:

Parameters

- hash - the component hash (e.g. hash=1234567890)
- groupId - the Group ID as part of the GAV (e.g. groupId=com.mycompany)
- artifactId - the Artifact ID as part of the GAV (e.g. artifactId=example-library)
- version - the Version as part of the GAV (e.g. version=1.6.1)

Now, let's consider an example where we wanted to find all components within the group ID "tomcat", for any applications evaluated during the build stage. Using the above, as well as cURL, we should have something like this...

```
curl -u admin:admin123 -X GET "http://localhost:8070/api/v1/search/ ↵  
component?stageId=build&groupId=tomcat"
```

Note

Included in our cURL example is the default CLM server location (localhost:8070) as well as the default username and password for the admin account. Your account credentials may vary, but are

necessary in order for the request to be processed. The results provided will be filtered based on the permissions of the credentials you use. For example, if you are not included in at least the developer role for an application, no results will be returned for that application. Given this, some results may be obscured.

Alright, so in our case, the API above produced the following results. If you have any tomcat component, you could expect something similar...

```
{
  "results": [
    {
      "applicationId": "bom1-12345678",
      "applicationName": "dev",
      "reportUrl": "http://localhost:8070/ui/links/application/bom1 ↵
        -12345678/report/75492b2ce44e4cfc94594eedf9d30895",
      "hash": "1249e25aebb15358bedd",
      "groupId": "tomcat",
      "artifactId": "tomcat-util",
      "version": "5.5.23",
      "threatLevel": 8
    },
    {
      "applicationId": "demo-app-1",
      "applicationName": "CLM",
      "reportUrl": "http://localhost:8070/ui/links/application/demo- ↵
        app-1/report/ee5d569f75864acc98321c68ed3ad76a",
      "hash": "1249e25aebb15358bedd",
      "groupId": "tomcat",
      "artifactId": "tomcat-util",
      "version": "5.5.23",
      "threatLevel": 8
    },
    {
      "applicationId": "demo-app-2",
      "applicationName": "Nexus",
      "reportUrl": "http://localhost:8070/ui/links/application/demo ↵
        -app-2/report/7ab76c4f15da4bdab82c37c09b125078",
      "hash": "1249e25aebb15358bedd",
      "groupId": "tomcat",
      "artifactId": "tomcat-util",
      "version": "5.5.23",
      "threatLevel": 8
    }
  ],
  "criteria": {
    "stageId": "build",
    "hash": null,
    "groupId": "tomcat",
```

```
    "artifactId": null,  
    "version": null  
  }  
}
```

Tip

These results should be self explanatory, though it is important to note that the criteria specified as part of the call is provided as well.

Using Wildcards

Of course, you may come across instances where you want to produce more results with less specific component details. If this is the case, the Component Search API does support the use of wildcards when searching using the GAV (coordinates).

If you are familiar with the coordinates policy condition, it follows the exact same logic. You can read more on the Coordinates condition in [Step Five in Policy Management](#) chapter.

20.2 Component Information API (v1)

**Warning**

Sonatype CLM APIs are versioned. This document represents a deprecated version (v1) and we highly recommend updating to the latest version of Sonatype CLM and using the [latest version of this API](#).

When Sonatype CLM evaluates an application, information found during the evaluation is provided via the [Application Composition Report](#). While this is the easiest way to review this information, it can also be exported to a JSON file using the Component Information API.

The API works by sending a REST request to the Sonatype CLM Server. This request involves using a specially formatted URL and any HTTP client. In the example we have provided we make use of cURL and the command line. In addition, we format the JSON to make it a bit more readable.

Sending the Request

First, let's take a look at the GET API we'll be using:

```
GET http://localhost:8070/api/v1/applications/[applicationId]/reports/[reportId]
```

As you may have noticed, this API uses a URL specific to the location of the report. There are two pieces of information you will need in order to retrieve the results.

- applicationId - The Application ID for the specific application.
- reportId - The ID of the specific report.

There are a variety of ways to retrieve this information, including gathering it by using the <api-v1-comp-search-intro,Component Search API>. However, in our example, we're just going to pull this information from the log output of an [evaluation](#) using Sonatype CLM for CLI.

Here's an example of the log output produced using Sonatype CLM for CLI:

```
...
```

```
*****  
[INFO] Policy Action: None [INFO] CLM stage: build [INFO] Summary of policy  
violations: 6 critical, 11 severe, 1 moderate [INFO] The detailed report can be  
viewed online at  
http://localhost:8070/ui/links/application/MyApp-1234/report/612247-fa6bf64ae98e472054b492fbfb  
[INFO]  
*****
```

Note

This section has been truncated, showing only the last section of the log output.

We'll use the link for the report to gather the information we need. Here is link specifically:

```
http://localhost:8070/ui/links/application/MyApp-1234/report/68 ↵  
b6bdb1573a40eeb4205d890b602525
```

In the example above, the segment of the URL following "application" and before the segment "report", is the Application ID. Similarly, the segment after "report" is the report ID.

Note

The report is also presented in a full GUI. To access this, simply paste the link in your browser. However, you will need to be at least a member of the developer group for the application that has been evaluated, or you will not be able to access the report.

Now, we can download the data using our HTTP request tool.

Downloading Component Information

Again, in our example we are going to use cURL, though any HTTP client could be used. Here is what our request looks like:

```
curl -u admin:admin123 -X GET "http://localhost:8070/api/v1/applications/ ↵  
MyApp-1234/reports/68b6bdb1573a40eeb4205d890b602525"
```

Note

Included in our cURL example is the default username and password for the admin account. Your account credentials may vary, but are necessary in order for the request to be processed. If the username and password provided are not at least within the developer role for the application, an error will be returned.

Looking at the result through a JSON Viewer, you should see something like this:

```
{  
  "components": [  
    {  
      "hash": "1249e25aebb15358bedd",  
      "mavenCoordinates": {
```

```
    "groupId":"tomcat",
    "artifactId":"tomcat-util",
    "version":"5.5.23"
  },
  "matchState":"exact",
  "proprietary":false,
  "pathnames":[
    "all-your-dependencies.zip/all-your-dependencies/sample-clm- ↵
      application.zip/tomcat-util-5.5.23.jar",
    "all-your-dependencies.zip/all-your-dependencies/tomcat-util ↵
      -5.5.23.jar"
  ],
  "licenseData":{
    "declaredLicenses":[
      {
        "licenseId":"Apache-2.0",
        "licenseName":"Apache-2.0"
      }
    ],
    "observedLicenses":[
      {
        "licenseId":"No-Sources",
        "licenseName":"No Sources"
      }
    ],
    "overriddenLicenses":[

    ],
    "status":"Open"
  },
  "securityData":{
    "securityIssues":[
      {
        "source":"cve",
        "reference":"CVE-2007-3385",
        "severity":4.3,
        "status":"Open"
      }
    ],
  }
}
]
```

Now that you have access to this data, it can be packaged in a variety of ways. For example, you may want to use this as a bill of materials to include associated license data.

20.3 Application REST APIs (v1)



Warning

Sonatype CLM APIs are versioned. This document represents a deprecated version (v1) and we highly recommend updating to the latest version of Sonatype CLM and using the [latest version of this API](#).

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, <update-app,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 `&publicID={SomeOtherPublicId}` to the command above.

Placing your application's `publicId` where `{YourPublicID}` is, the following 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.

20.4 Violation REST API (v1)



Warning

Sonatype CLM APIs are versioned. This document represents a deprecated version (v1) and we highly recommend updating to the latest version of Sonatype CLM and using the [latest version of this 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 examples 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.

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. 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 the Sonatype 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. For example, in the case of Java components, this would be matched to Central. If you have proprietary components configured, it would be matched against your list of proprietary components.

groupId, artifactId, version

This is the G : A : V (Group, Artifact Id, Version) for the component.

20.5 Supported Component Identifiers

In the v1 API a Maven coordinate structure (GAV) is used when referring to a component and as input to search. A breaking change was introduced in v2 to use an agnostic structure when referring to a component's coordinates as well as the format. This new structure (component identifier) provides flexibility across different packaging systems and how a component is referenced in those systems.

Support for component identifiers includes the following packaging systems:

- Maven
- NuGet

As an example, the skeleton of this agnostic structure is:

```
{
  "componentIdentifier":{
    "format":"supported-format",
    "coordinates":{

    }
  }
}
```

Maven Component Identifier

Maven components are indicated with the format of *maven* and specify attributes for the complete Maven GAVEC:

```
{
  "componentIdentifier":{
    "format":"maven",
    "coordinates":{
      "artifactId":"tomcat-util",
      "classifier":"",
      "extension":"jar",
      "groupId":"tomcat",
      "version":"5.5.23"
    }
  }
}
```

NuGet Component Identifier

NuGet components are indicated with the format of *nuget*:

```
{
  "componentIdentifier":{
    "format":"nuget",
    "coordinates":{
      "packageId":"Newtonsoft.Json",
      "version":"6.0.8"
    }
  }
}
```

20.6 Component Search REST APIs (v2)

You may reach a point where you would like to search for a particular component that has been identified during a CLM evaluation. In other words, you have already completed an evaluation (more likely you have done several or many evaluations), and either the CLM server has identified, or you have claimed, at least one component in that application.

The likely scenario is that CLM has identified many components across all of your applications. Even more likely, a few of the same components show up in several (maybe all) of your applications. In these instances, the Sonatype CLM Component Search API allows you to both find a particular component, as well as get information back about that component.

This API consists of GET requests, which allow you to retrieve component information such as application ID, application name, report URL, component hash, component coordinates, and perhaps most importantly, it provides the highest threat level of the policy violations (for the found component).

Below, we've provided an example of the GET request. We've done this using the HTTP client cURL. Of course, you could use any HTTP client tool. Additionally, to help demonstrate use of the API, we've broken out the various pieces for this request and provided an example of data that is retrieved.

Note

Compared to the other APIs, Component Search is fairly simple. However, you should have some basic information about the component coordinates, as well as the Sonatype CLM stage (e.g. build) where the component was analyzed.

Searching for Components

First, make sure your Sonatype CLM server is started. Also, as we mentioned, you will need to have evaluated at least one application. With those two things completed, let's take a look at the GET API.

```
GET /api/v2/search/component
```

Now, in addition to this, you will need to set the stage, and then add your search parameters. Let's take a look at stage first.

Stage

Typically the stage represents the development lifecycle of your product. There are four stages that are currently supported. These include:

- build
- stage
- stage-release
- operate

Entering any of these for the stage ID will pull from that specific stage's evaluation data.

Next up, you need to set the component search parameters using any combination of these options:

Parameters

- hash - the component hash (e.g. hash=1234567890)
- componentIdentifier - this is an object that contains the the cordinates of the component and its format.
- format - this is the format of the component (e.g. "maven").
- coordinates - this object contains the name/value pairs for identifying coordinates (e.g. artifactId, groupId, and version).

Now, let's look at an example. Consider a case where we wanted to find all components within the group ID "tomcat", for any applications evaluated during the build stage.. Using the information above, as well as cURL and an encoded URL, here is what we would have...

```
curl -u admin:admin123 -X GET "http://localhost:8070/api/v2/search/ ↵
component?stageId=build&componentIdentifier=%7B%22format%22%3A%22maven ↵
%22%2C%22coordinates%22%3A%7B%22groupId%22%3A%22tomcat%22%7D%7D"
```

Of course the above is an encoded URL, so just for a bit of help, here is what a non-encoded URL would look like. This should help you identify the JSON in the example above.

```
"http://localhost:8070/api/v2/search/component?stageId=build& ↵
componentIdentifier={"format":"maven","coordinates":{"groupId":"tomcat ↵
"}}"
```

Note

Included in our cURL example is the default CLM server location (localhost:8070) as well as the default username and password for the admin account. Your account credentials may vary, but are necessary in order for the request to be processed. The results provided will be filtered based on the permissions of the credentials you use. For example, if you are not included in at least the developer role for an application, no results will be returned for that application. Given this, some results may be obscured.

Alright, so in our case, the API above produced the following results. If you have any tomcat components, you could expect something similar...

```
{
  "criteria":{
    "stageId":"build",
    "hash":null,
    "componentIdentifier":{
      "format":"maven",
      "coordinates":{
        "groupId":"tomcat"
      }
    }
  },
  "results":[
    {
      "applicationId":"MyApp-1234",
      "applicationName":"My Application 2",
      "reportUrl":"http://localhost:8070/ui/links/application/MyApp ↵
-1234/report/c81991938f304f30bc139ea13cf93cd5",
      "hash":"1249e25aebb15358bedd",
      "componentIdentifier":{
        "format":"maven",
        "coordinates":{
```

```
        "artifactId": "tomcat-util",
        "classifier": "",
        "extension": "jar",
        "groupId": "tomcat",
        "version": "5.5.23"
    }
}
```

Note

The data above was formatted to make it a bit more readable.

Using Wildcards

Of course, you may come across instances where you want to produce more results with less specific component details. If this is the case, the Component Search API does support the use of wildcards when searching using the GAV (coordinates).

If you are familiar with the coordinates policy condition, it follows the exact same logic. You can read more on the Coordinates condition in [Step Five in the Policy Management chapter](#).

20.7 Component Information API (v2)

When Sonatype CLM evaluates an application, information found during the evaluation is provided via the [Application Composition Report](#). While this is the easiest way to review this information, it can also be exported to a JSON file using the Component Information API.

The API works by sending a REST request to the Sonatype CLM Server. This request involves using a specially formatted URL and any HTTP client. In the example we have provided we make use of cURL and the command line. In addition, we format the JSON to make it a bit more readable.

Sending the Request

First, let's take a look at the GET API we'll be using:

```
GET http://localhost:8070/api/v2/applications/[applicationId]/reports/[reportId]
```

As you may have noticed, this API uses a URL specific to the location of the report. There are two pieces of information you will need in order to retrieve the results.

- applicationId - The Application ID for the specific application.
- reportId - The ID of the specific report.

There are a variety of ways to retrieve this information, including gathering it by using the [Component Search API](#). However, in our example, we're just going to pull this information from the log output of an [evaluation](#) using Sonatype CLM for CLI.

Here's an excerpt of the log output produced using Sonatype CLM for CLI.

```
...
*****
[INFO] Policy Action: None [INFO] CLM stage: build [INFO] Summary of policy
violations: 6 critical, 11 severe, 1 moderate [INFO] The detailed report can be
viewed online at
http://localhost:8070/ui/links/application/MyApp-1234/report/68
b6bdb1573a40eeb4205d890b602525
[INFO]
*****
```

NOTE:

We'll use the link for the report to gather the information we need:

```
http://localhost:8070/ui/links/application/MyApp-1234/report/68
b6bdb1573a40eeb4205d890b602525
```

In the example above, the section of the URL following "application" and before the section "report", is the Application ID. Similarly, the section after "report" is the report ID.

Note

The report is also presented in a full GUI. To access this, simply paste the link in your browser. However, you will need to be at least a member of the developer group for the application that has been evaluated, or you will not be able to access the report.

Now, we can download the data using our HTTP request tool.

Downloading Component Information

Again, in our example we are going to use cURL, though any HTTP client could be used. Here is what our request looks like:

```
curl -u admin:admin123 -X GET "http://localhost:8070/api/v2/applications/ ↵  
MyApp-1234/reports/68b6bdb1573a40eeb4205d890b602525"
```

Note

Included in our cURL example is the default username and password for the admin account. Your account credentials may vary, but are necessary in order for the request to be processed. If the username and password provided are not at least within the developer role for the application, an error will be returned.

Looking at the result through a JSON Viewer, you should see something like this:

```
{  
  "components": [  
    {  
      "hash": "1249e25aebb15358bedd",  
      "componentIdentifier": {  
        "format": "maven",  
        "coordinates": {  
          "artifactId": "tomcat-util",  
          "groupId": "tomcat",  
          "version": "5.5.23"  
        }  
      }  
    },  
    "proprietary": false,  
    "matchState": "exact",  
    "pathnames": [  
      ]  
    ]  
  ]  
}
```

```
    "sample-application.zip/tomcat-util-5.5.23.jar"
  ],
  "licenseData":{
    "declaredLicenses":[
      {
        "licenseId":"Apache-2.0",
        "licenseName":"Apache-2.0"
      }
    ],
    "observedLicenses":[
      {
        "licenseId":"No-Sources",
        "licenseName":"No Sources"
      }
    ],
    "overriddenLicenses":[

    ],
    "status":"Open"
  },
  "securityData":{
    "securityIssues":[
      {
        "source":"cve",
        "reference":"CVE-2007-3385",
        "severity":4.3,
        "status":"Open",
        "url":"http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE ↵
          -2007-3385",
        "threatCategory":"severe"
      }
    ]
  }
}
]
```

Now that you have access to this data, it can be packaged in a variety of ways. For example, you may want to use this as a bill of materials to include associated license data.

20.8 Component Evaluation REST APIs (v2)

The Component Evaluation REST API allows for a single component, or multiple components to be evaluated against a specific application, and the associated policies.

This API uses the following REST resources:

- POST - to submit a component or list of components, as well as the application containing the policies the component(s) will be evaluated against.
- GET - to check the status of the evaluation and retrieve the results when completed.

For the API we provide a step-by-step example using the HTTP client CURL, though any HTTP client tool could be used. In addition we'll reference other APIs such as the one required to obtain an application's internal ID.

Step 1 - Get Component Information

First, you need to decide which components you want to evaluate. You'll need a bit of component information in the form of the GAVE (Group, Artifact, Version, and Extension) or the hash for each component you wish to evaluate.

There are a variety of ways to get the GAVE, the most common is to get the information from the repository that houses the component. For example, [The Central Repository](#) has the GAVE for all components stored there.

If you wish to use the hash, Sonatype uses sha1 for all components it processes.

Tip

Only components known to Sonatype will provide security, license, and popularity data related to policies. However, depending on the policies for the application you choose to evaluate the component against, other policy conditions, such as match state, will help in determining if the component is known to your instance of CLM.

Step 2 - Get the Application ID

Next, you will need pick the application you want to evaluate your component against. This begins with obtaining the Application's Public ID, which is used to retrieve the internal Application ID.

The public Id can be found via the Sonatype CLM GUI by navigating to the respective application and copying the Application ID located just below the Application Name.

This is done using the following GET REST resource from our application API:

```
GET /api/v2/applications?publicId={YourPublicId}
```

Using the cURL command, it would look like this:

```
curl -u admin:admin123 -X GET 'http://localhost:8070/api/v2/applications? ←  
publicId=MyApplicationID'
```

If you like, you can also return multiple applications by appending additional `&publicId={SomeOtherPublicId}` to the command above. Alternatively, if desired, all applications can always be returned by omitting the reference to the public id. This will give the Public ID and internal Application ID for all applications.

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

```
{  
  "applications": [  
    {  
      "id": "4537e6fe68c24dd5ac83efd97d4fc2f4",  
      "publicId": "MyApplicationID",  
      "name": "MyApplication",  
      "organizationId": "bb41817bd3e2403a8a52fe8bcd8fe25a",  
      "contactUserName": "NewAppContact",  
      "applicationTags": [  
        {  
          "id": "9beee80c6fc148dfa51e8b0359ee4d4e",  
          "tagId": "cfea8fa79df64283bd64e5b6b624ba48",  
          "applicationId": "4bb67dcfc86344e3a483832f8c496419"  
        }  
      ]  
    }  
  ]  
}
```

From the information returned above, make note of the `id`. This is the internal Application ID.

Step 3 Submit Component for Evaluation

Alright, by now you should have either the GAVE or hash for your component, as well as the internal Application ID. We'll put this information together using the POST REST resource:

```
POST /api/v2/evaluation/applications/{applicationId}
```

Added to this will be a JSON formatted body:

```
{
  "hash": "null",
  "componentIdentifier": {
    "format": "maven",
    "coordinates": {
      "artifactId": "commons-fileupload",
      "groupId": "commons-fileupload",
      "version": "1.2.2",
      "extension": "jar"
    }
  }
}
```

Tip

If desired, multiple components can be included, but remember, the extension is required when entering components by GAVE.

Together, this should look like this:

```
curl -u admin:admin123 -X POST -H "Content-Type: application/json" -d'{"components": [{"hash": "null", "componentIdentifier": {"format": "maven", "coordinates": {"artifactId": "commons-fileupload", "groupId": "commons-fileupload", "version": "1.2.2", "extension": "jar"}}}]}' 'http://localhost:8070/api/v2/evaluation/applications/529b7f71bb714eca8955e5d66687ae2c'
```

A successful POST will result in JSON formatted data providing a confirmation that the evaluation was submitted.

```
{
  "resultId": "917e0c5e92a646a5b8879a40890078bc",
  "submittedDate": "2015-02-02T10:43:22.975-05:00",
  "applicationId": "529b7f71bb714eca8955e5d66687ae2c",
  "resultsUrl": "api/v2/evaluation/applications/529 ←
    b7f71bb714eca8955e5d66687ae2c/results/917 ←
    e0c5e92a646a5b8879a40890078bc"
}
```

Be sure to make note of *resultID* and "applicationID". These will be used to check and retrieve results. If you are following along with CURL, you can simply copy the *resultsURL*.

Step 4 Get Evaluation Results

Now, depending on how many components you evaluated, the CLM Server may need some time to process them. You can check the status and obtain results using the following GET REST Resource:

```
GET /api/v2/evaluation/applications/{applicationId}/{resultId}
```

In our example, we're going to use the *resultsURL* and as before, CURL:

```
curl -u admin:admin123 -X GET 'http://localhost:8070/api/v2/evaluation/ ←
  applications/529b7f71bb714eca8955e5d66687ae2c/results/917 ←
  e0c5e92a646a5b8879a40890078bc'
```

If the report is ready, you will receive results similar to these below:

Note

If the report is not ready, you will receive a (404) error.

```
{
  "submittedDate": "2015-02-13T18:01:42.082-05:00",
  "evaluationDate": "2015-02-13T18:01:42.084-05:00",
  "applicationId": "529b7f71bb714eca8955e5d66687ae2c",
  "results": [
    {
      "component": {
        "hash": null,

```

```
    "componentIdentifier":{
      "format":"maven",
      "coordinates":{
        "artifactId":"commons-fileupload",
        "extension":"jar",
        "groupId":"commons-fileupload",
        "version":"1.2.2"
      }
    },
    "proprietary":false
  },
  "matchState":"exact",
  "catalogDate":"2010-07-29T16:41:12.000-04:00",
  "licenseData":{
    "declaredLicenses":[
      {
        "licenseId":"Apache-2.0",
        "licenseName":"Apache-2.0"
      }
    ],
    "observedLicenses":[
      {
        "licenseId":"Apache-2.0",
        "licenseName":"Apache-2.0"
      }
    ],
    "overriddenLicenses":[

    ],
    "status":"Open"
  },
  "securityData":{
    "securityIssues":[
      {
        "source":"cve",
        "reference":"CVE-2013-2186",
        "severity":7.5,
        "status":"Open",
        "url":"http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE ↔
          -2013-2186",
        "threatCategory":"critical"
      },
      {
        "source":"osvdb",
        "reference":"98703",
        "severity":7.5,
        "status":"Open",
        "url":"http://osvdb.org/98703",
        "threatCategory":"critical"
      }
    ]
  }
}
```

```
    },
    {
      "source": "cve",
      "reference": "CVE-2014-0050",
      "severity": 5.0,
      "status": "Open",
      "url": "http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE ←
        -2014-0050",
      "threatCategory": "severe"
    },
    {
      "source": "osvdb",
      "reference": "102945",
      "severity": 5.0,
      "status": "Open",
      "url": "http://osvdb.org/102945",
      "threatCategory": "severe"
    }
  ]
},
"policyData": {
  "policyViolations": [
    {
      "policyId": "0ccc3321662d491c86b32554ea84e4e9",
      "policyName": "Security-High",
      "threatLevel": 9,
      "constraintViolations": [
        {
          "constraintId": "d2dfc91f3bf4460ba0da3f7201e4adb7",
          "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"
            }
          ]
        }
      ]
    }
  ]
},
{
  "policyId": "0f1b57e6788c4e9bb0f65cde3b6e9f1c",
```

```
"policyName": "Security-Medium",
"threatLevel": 7,
"constraintViolations": [
  {
    "constraintId": "12a71811de694876b51e4c06c10bad76",
    "constraintName": "CVSS >=4 and <7",
    "reasons": [
      {
        "reason": "Found 4 Security Vulnerabilities ↔
                  with Severity >= 4"
      },
      {
        "reason": "Found 2 Security Vulnerabilities ↔
                  with Severity < 7"
      },
      {
        "reason": "Found 4 Security Vulnerabilities ↔
                  with Status OPEN"
      }
    ]
  }
]
}
]
}
]
}
],
"isError": false,
"errorMessage": null
}
```

Note

The last bit of information indicates if there were any errors encountered during the evaluation. In this example, there were no errors.

20.9 Application REST APIs (v2)

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/v2/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/v2/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/v2/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/v2/ ←
  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/v2/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/v2/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/v2/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/v2/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/v2/applications? ↵  
publicId=MyApplicationID'
```

If you like, you can also return multiple applications by appending additional `&publicId={SomeOtherPublicId}` to the command above.

Placing your application's `publicId` where `{YourPublicId}` is, the following 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/v2/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.

20.10 Violation REST API (v2)

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 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 examples 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.

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/v2/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/v2/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/v2/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/v2/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/v2/ ↵  
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",
  "threatLevel":9,
  "constraintViolations":[
    {
      "constraintId":"19011de290b147a38c820ad7bd5c653d",
      "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"
        }
      ]
    }
  ],
  "component":{
    "hash":"384faa82e193d4e4b054",
    "componentIdentifier":{
      "format":"maven",
      "coordinates":{
        "artifactId":"tomcat-util",
        "classifier":"",
        "extension":"jar",
        "groupId":"tomcat",
        "version":"5.5.23"
      }
    },
    "proprietary":false
  }
}
```

And there you have it, you've just retrieved policy violations. 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 the Sonatype 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.

stageId

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.

threatLevel

This the threat level of the policy that was violated.

constraintViolations

This is a subcategory of 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.

component

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

Note

The component field only refers to a single component. If another component violates the same policy, another entry for that in "policyViolations" would be present.

hash

This is the hash value for the component. For example, in the case of Java components, this would be matched to a Java repository (e.g. Central). If you have proprietary components configured, it would be matched against your list of proprietary components.

componentIdentifier

This is simply a container for the component information. It will always include the format and the coordinates.

format

This is the format the component is in, and will determine what type of coordinate information is displayed.

coordinates

This will depend on the format. An example would be Maven, which uses a G : A : E : C : V (Group, Artifact Id, Extension, Classifier, and Version) for the component. In this example, the fields provided are: groupId, artifactId, and version.

20.11 Reports REST API (v2)

The Reports API provides a summary of an application's most recent reports across the various CLM stages (e.g. build, stage release, and release). The API consists of using the GET REST resource that is part of both the applications and reports APIs.

To assist in learning to use this API we will provide both the API as well as an example using the HTTP client cURL. We've approached this in a step-by-step manner that will start with gathering the internal Application ID and then retrieving the report information.

Note

In our example below, we have isolated retrieving a report for a single application. However, if desired the reports for all applications can also be retrieved. In this case, skip to Step 2, where a corresponding tip has been included to assist.

Step 1 - Get the Application ID

First, you will use the Application's Public ID to retrieve the internal Application ID. This is done using the following GET REST resource from our application API. . .

```
GET /api/v2/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/v2/applications? ↵  
publicId=MyApplicationID'
```

If you like, you can also return multiple applications by appending additional `&publicId={SomeOtherPublicId}` to the command above.

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

```
{
  "applications": [
    {
      "id": "4537e6fe68c24dd5ac83efd97d4fc2f4",
      "publicId": "MyApplicationID",
      "name": "MyApplication",
      "organizationId": "bb41817bd3e2403a8a52fe8bcd8fe25a",
      "contactUserName": "NewAppContact",
      "applicationTags": [
        {
          "id": "9beee80c6fc148dfa51e8b0359ee4d4e",
          "tagId": "cfea8fa79df64283bd64e5b6b624ba48",
          "applicationId": "4bb67dcfc86344e3a483832f8c496419"
        }
      ]
    }
  ]
}
```

From the information returned above, make note of the `id`. This is the internal Application ID.

Note

The public Id can be found via the Sonatype CLM GUI by navigating to the respective application and copying the Application ID located just below the Application Name.

Step 2

Next we will take the internal Application ID from Step 1 and use that with the GET REST resource for the Reports API. This is done with our GET REST resource associated with our Reports API. . .

```
GET /api/v2/reports/applications/{id}
```

Tip

If you would like to obtain the report information for all applications, simply leave off the "id" and all report information for all applications will be provided.

Using cURL, it will look like this. . .

```
curl -u admin:admin123 -X GET 'http://localhost:8070/api/v2/reports/
applications/4bb67dcfc86344e3a483832f8c496419'
```

If any report information is found for the application, you will receive JSON formatted data. An example is included below.

```
[
  {
    "stage": "build",
    "applicationId": "4537e6fe68c24dd5ac83efd97d4fc2f4",
    "evaluationDate": "2015-01-16T13:14:32.139-05:00",
    "reportHtmlUrl": "ui/links/application/Test123/report/474 ←
      ca07881554f8fbec168ec25d9616a",
    "reportPdfUrl": "ui/links/application/Test123/report/474 ←
      ca07881554f8fbec168ec25d9616a/pdf",
    "reportDataUrl": "api/v2/applications/Test123/reports/474 ←
      ca07881554f8fbec168ec25d9616a"
  },
  {
    "stage": "stage-release",
    "applicationId": "4537e6fe68c24dd5ac83efd97d4fc2f4",
    "evaluationDate": "2014-07-14T15:08:09.501-04:00",
    "reportHtmlUrl": "ui/links/application/Test123/report/2 ←
      dbfd514e5ad497598086c3e4c89c413",
    "reportPdfUrl": "ui/links/application/Test123/report/2 ←
      dbfd514e5ad497598086c3e4c89c413/pdf",
    "reportDataUrl": "api/v2/applications/Test123/reports/2 ←
      dbfd514e5ad497598086c3e4c89c413"
  },
  {
    "stage": "release",
    "applicationId": "4537e6fe68c24dd5ac83efd97d4fc2f4",
    "evaluationDate": "2014-08-31T12:01:46.179-04:00",
    "reportHtmlUrl": "ui/links/application/Test123/report/97 ←
      cccbeeb58d4aac83264993ef6bf4d8",
    "reportPdfUrl": "ui/links/application/Test123/report/97 ←
      cccbeeb58d4aac83264993ef6bf4d8/pdf",
    "reportDataUrl": "api/v2/applications/Test123/reports/97 ←
      cccbeeb58d4aac83264993ef6bf4d8"
  }
]
```

Additionally, CLM will return the standard HTTP response codes.

Appendix A

Copyright

Copyright © 2011-2013 Sonatype, Inc. All rights reserved.

Online version published by Sonatype, Inc,

Nexus™, Nexus Professional™, and all Nexus-related logos as well as Sonatype CLM are trademarks or registered trademarks of Sonatype, Inc., in the United States and other countries.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle, Inc., in the United States and other countries.

IBM® and WebSphere® are trademarks or registered trademarks of International Business Machines, Inc., in the United States and other countries.

Eclipse™ is a trademark of the Eclipse Foundation, Inc., in the United States and other countries.

Apache and the Apache feather logo are trademarks of The Apache Software Foundation.

Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and Sonatype, Inc. was aware of a trademark

claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and authors assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.