# Sonatype CLM - Repository Manager User Guide

# Contents

# List of Figures

# Chapter 1

# Introduction

This guide is designed to help you better understand how Sonatype CLM can be integrated with repository managers, such as Nexus Professional CLM Edition. It covers a brief background on repository management, as well as an in depth look at configuration and usage of the CLM-specific elements for repository managers such as Sonatype Nexus Professional CLM Edition.

# Chapter 2

# Sonatype CLM for Repository Managers

Repository managers allow you to manage repositories filled with software components required for development, deployment, and provisioning. You can publish your own components to these repositories as well as automatically proxy external repositories like the Central Repository to provide efficient component access to your organization. In this role they fulfill a central part for component lifecycle management.

A repository manager greatly simplifies the maintenance of your own internal repositories, as well as 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 manage, which components get into your products from external sources as well as examine and keep track of components 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 code, 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, enabling greater collaboration between developers and wider distribution of all components. You benefit from increased build performance due to local component availability and reduced bandwidth needs by avoiding repeated downloads to your setup.

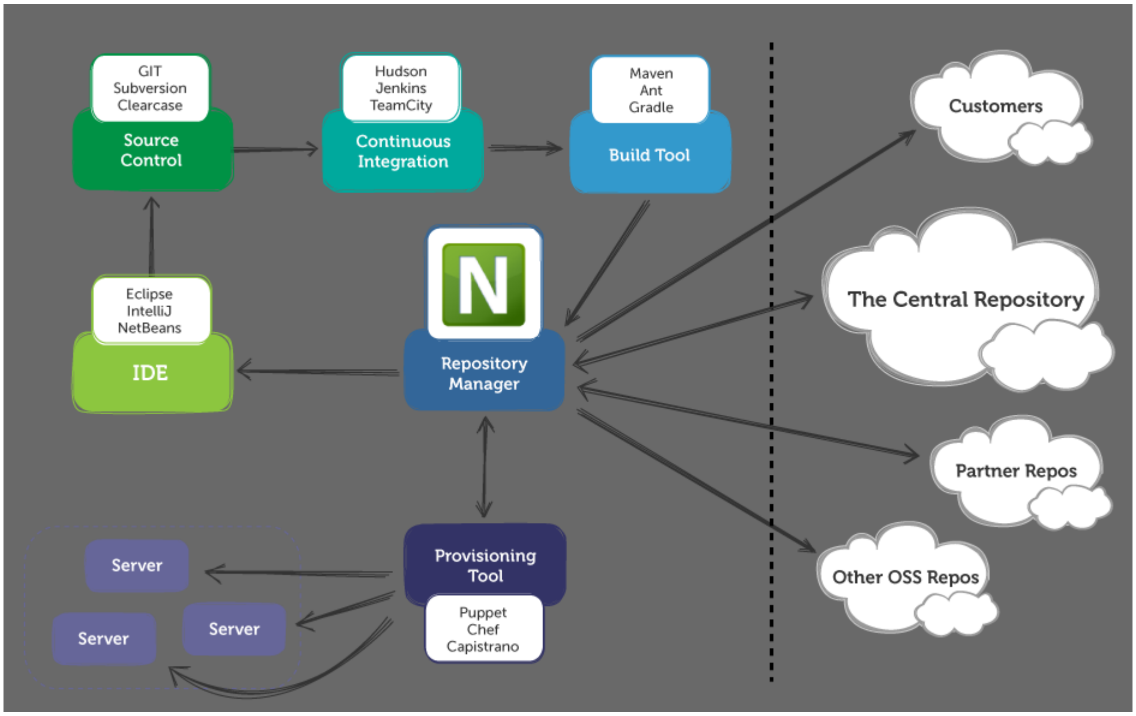


Figure 2.1: The Central Role of A Repository Manager in Your Infrastructure

---

**Note**

The book Repository Management with Nexus provides an extensive introduction to repository management, its advantages and stages of adoptions for further reference. If this is your first introduction to repository management, there is a wealth of information, that expands beyond what we have provided here.

---

# Chapter 3

# Nexus Pro and Sonatype CLM Integration

## 3.1 Introduction

Nexus comes in two forms, the popular Nexus Open Source , as well as industry-leading Nexus Professional. In addition, users of Nexus Professional can add the Nexus CLM License to expand functionality to include use of Sonatype CLM as part of Nexus Professional staging capabilities. This allows you to enjoy a robust repository manager coupled with the advanced policy and risk management features provided by Sonatype CLM.

In this section we'll discuss all the capabilities provided by the integration of Nexus Professional and Sonatype CLM. When necessary, we will indicate if a feature is exclusive to a Nexus Professional CLM Edition.

**Note**
If you are unsure of which Nexus License you have, please contact our Support Team at support@sonatype.com.

## 3.2 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.

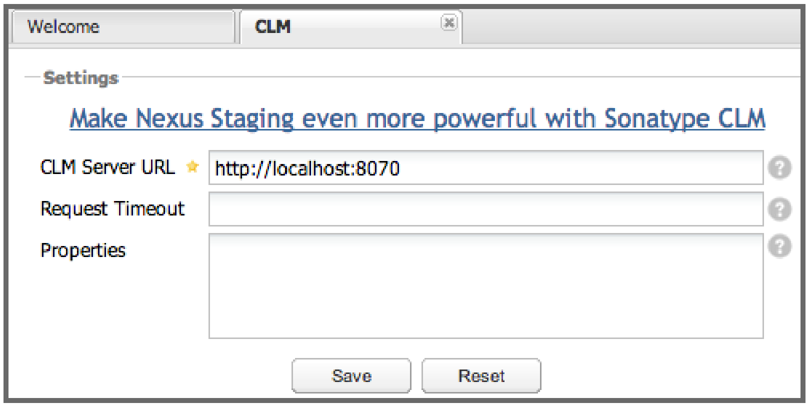
**Note**
Nexus Open Source and Nexus Professional both provide access to RHC, though the capabilities are expanded for Nexus Professional users. For more information on RHC and Nexus in general, please refer to the free book Repository Management with Nexus.

## 3.3 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 our Sonatype CLM Server Install and Configuration User Guide.

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 3.1.



CLM Server URL: http://localhost:8070
Request Timeout:
Properties:

Figure 3.1: CLM configuration tab in Nexus

The CLM connection is established by providing the URL to the CLM Server in the *CLM Server URL* input field and optionally a *Request Timeout*.

Additional details can be configured in the *Properties* input field using a `key=value` definition per line. An example is

```
procArch=false
ipAddresses=true
operatingSystem=false
```

Alternatively you can enable, or if desired disable, and 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.

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.

Press *Save* after you have entered the desired URL and properties, and Nexus will attempt to contact the CLM Server and potentially display an error message if the CLM Server could not be contacted.

**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.

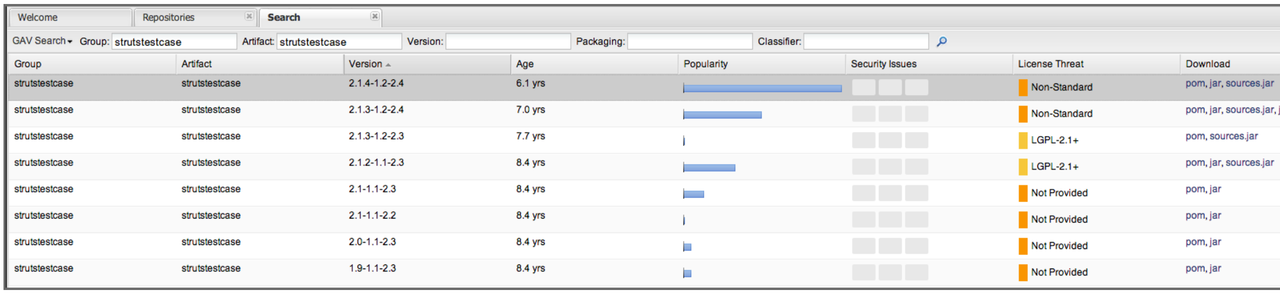
## 3.4 Configuring the CLM Server

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.

For more information of setting up organizations, applications, and policies, please review our Sonatype CLM Policy Management Guide.

## 3.5 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.


Welcome | Repositories | Search

GAV Search Group: strutstestcase Artifact: strutstestcase Version: Packaging: Classifier:

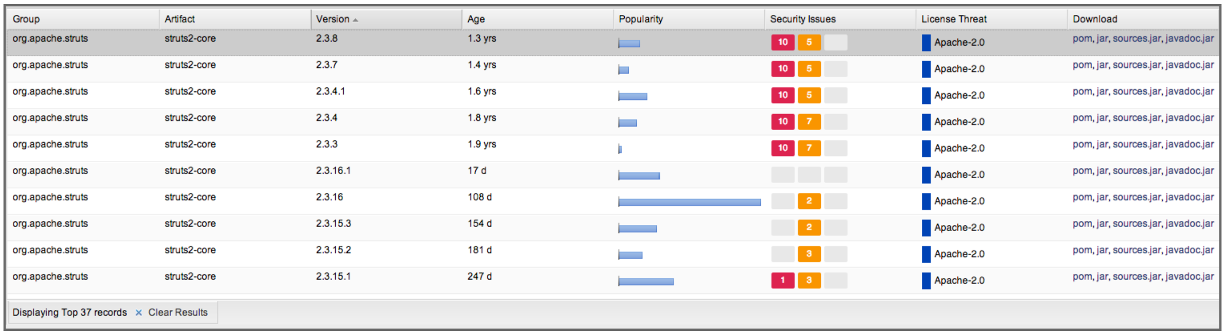
| Group | Artifact | Version | Age | Popularity | Security Issues | License Threat | Download |
|---|---|---|---|---|---|---|---|
| strutstestcase | strutstestcase | 2.1.4-1.2-2.4 | 6.1 yrs | | | Non-Standard | pom, jar, sources.jar |
| strutstestcase | strutstestcase | 2.1.3-1.2-2.4 | 7.0 yrs | | | Non-Standard | pom, jar, sources.jar, j |
| strutstestcase | strutstestcase | 2.1.3-1.2-2.3 | 7.7 yrs | | | LGPL-2.1+ | pom, sources.jar |
| strutstestcase | strutstestcase | 2.1.2-1.1-2.3 | 8.4 yrs | | | LGPL-2.1+ | pom, jar, sources.jar |
| strutstestcase | strutstestcase | 2.1-1.1-2.3 | 8.4 yrs | | | Not Provided | pom, jar |
| strutstestcase | strutstestcase | 2.1-1.1-2.2 | 8.4 yrs | | | Not Provided | pom, jar |
| strutstestcase | strutstestcase | 2.0-1.1-2.3 | 8.4 yrs | | | Not Provided | pom, jar |
| strutstestcase | strutstestcase | 1.9-1.1-2.3 | 8.4 yrs | | | Not Provided | pom, jar |

Figure 3.2: Typical Search Results in Nexus Pro

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 | | 10 5 | Apache-2.0 | pom, jar, sources.jar, javadoc.jar |
| org.apache.struts | struts2-core | 2.3.7 | 1.4 yrs | | 10 5 | Apache-2.0 | pom, jar, sources.jar, javadoc.jar |
| org.apache.struts | struts2-core | 2.3.4.1 | 1.6 yrs | | 10 5 | Apache-2.0 | pom, jar, sources.jar, javadoc.jar |
| org.apache.struts | struts2-core | 2.3.4 | 1.8 yrs | | 10 7 | Apache-2.0 | pom, jar, sources.jar, javadoc.jar |
| org.apache.struts | struts2-core | 2.3.3 | 1.9 yrs | | 10 7 | Apache-2.0 | pom, jar, sources.jar, javadoc.jar |
| org.apache.struts | struts2-core | 2.3.16.1 | 17 d | | | Apache-2.0 | pom, jar, sources.jar, javadoc.jar |
| org.apache.struts | struts2-core | 2.3.16 | 108 d | | 2 | Apache-2.0 | pom, jar, sources.jar, javadoc.jar |
| org.apache.struts | struts2-core | 2.3.15.3 | 154 d | | 2 | Apache-2.0 | pom, jar, sources.jar, javadoc.jar |
| org.apache.struts | struts2-core | 2.3.15.2 | 181 d | | 3 | Apache-2.0 | pom, jar, sources.jar, javadoc.jar |
| org.apache.struts | struts2-core | 2.3.15.1 | 247 d | | 1 3 | Apache-2.0 | pom, jar, sources.jar, javadoc.jar |

Displaying Top 37 records × Clear Results

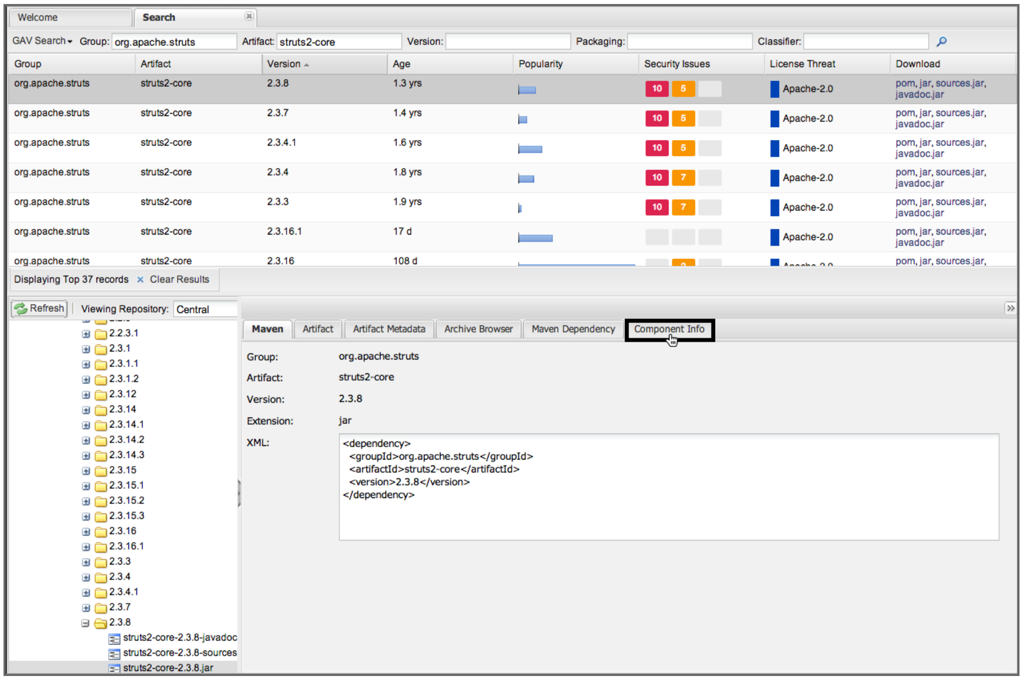
Figure 3.3: Nexus Search Showing All Versions

---

**Note**
Currently both RHC and Sonatype CLM only provide information for open source Java components available via Central.

---

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.



Group: org.apache.struts Artifact: struts2-core Version: Packaging: Classifier:

| Group | Artifact | Version | Age | Popularity | Security Issues | License Threat | Download |
|---|---|---|---|---|---|---|---|
| org.apache.struts | struts2-core | 2.3.8 | 1.3 yrs | | 10 5 | Apache-2.0 | pom, jar, sources.jar, javadoc.jar |
| org.apache.struts | struts2-core | 2.3.7 | 1.4 yrs | | 10 5 | Apache-2.0 | pom, jar, sources.jar, javadoc.jar |
| org.apache.struts | struts2-core | 2.3.4.1 | 1.6 yrs | | 10 5 | Apache-2.0 | pom, jar, sources.jar, javadoc.jar |
| org.apache.struts | struts2-core | 2.3.4 | 1.8 yrs | | 10 7 | Apache-2.0 | pom, jar, sources.jar, javadoc.jar |
| org.apache.struts | struts2-core | 2.3.3 | 1.9 yrs | | 10 7 | Apache-2.0 | pom, jar, sources.jar, javadoc.jar |
| org.apache.struts | struts2-core | 2.3.16.1 | 17 d | | | Apache-2.0 | pom, jar, sources.jar, javadoc.jar |
| org.apache.struts | struts2-core | 2.3.16 | 108 d | | | | pom, jar, sources.jar, |

Displaying Top 37 records × Clear Results

Viewing Repository: Central

- 2.2.3.1
- 2.3.1
- 2.3.1.1
- 2.3.1.2
- 2.3.12
- 2.3.14
- 2.3.14.1
- 2.3.14.2
- 2.3.14.3
- 2.3.15
- 2.3.15.1
- 2.3.15.2
- 2.3.15.3
- 2.3.16
- 2.3.16.1
- 2.3.3
- 2.3.4
- 2.3.4.1
- 2.3.7
- 2.3.8
  - struts2-core-2.3.8-javadoc
  - struts2-core-2.3.8-sources
  - struts2-core-2.3.8.jar

| | |
|---|---|
| Group: | org.apache.struts |
| Artifact: | struts2-core |
| Version: | 2.3.8 |
| Extension: | jar |

XML:
```
<dependency>
  <groupId>org.apache.struts</groupId>
  <artifactId>struts2-core</artifactId>
  <version>2.3.8</version>
</dependency>
```

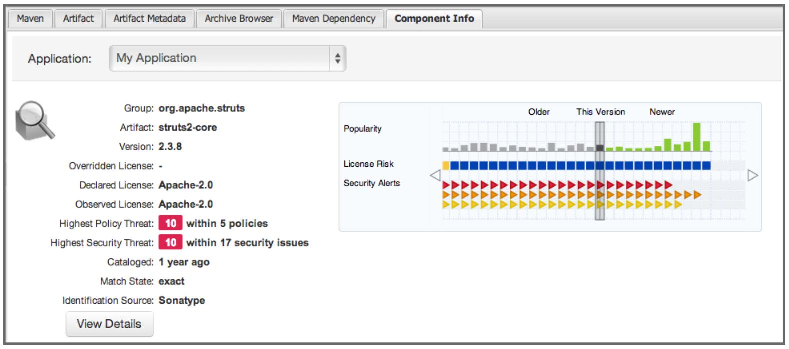
Figure 3.4: Accessing the Component Info Tab

---

**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.



Application: My Application

Group: org.apache.struts
Artifact: struts2-core
Version: 2.3.8
Overridden License: -
Declared License: Apache-2.0
Observed License: Apache-2.0
Highest Policy Threat: 10 within 5 policies
Highest Security Threat: 10 within 17 security issues
Cataloged: 1 year ago
Match State: exact
Identification Source: Sonatype

Figure 3.5: Component Information Panel

---

**Note**

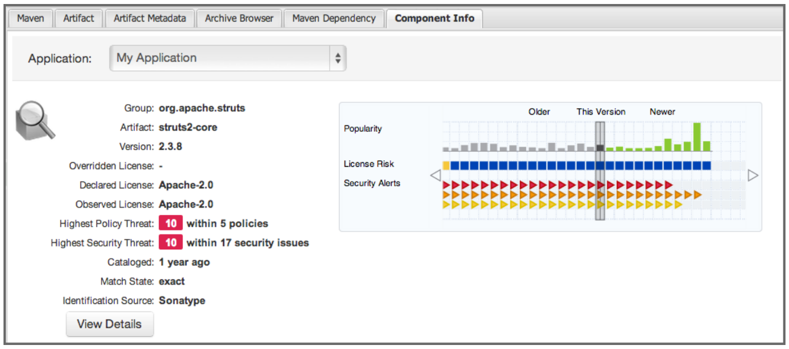
Information on the Component Info tab requires a Sonatype CLM License. Nexus Pro Users will simply be provided with additional details regarding the security vulnerabilities and license issues. Those using Nexus Open Source will not have access to the Component Info tab.

---

## 3.6 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.



Application: My Application

Group: org.apache.struts
Artifact: struts2-core
Version: 2.3.8
Overridden License: -
Declared License: Apache-2.0
Observed License: Apache-2.0
Highest Policy Threat: 10 within 5 policies
Highest Security Threat: 10 within 17 security issues
Cataloged: 1 year ago
Match State: exact
Identification Source: Sonatype

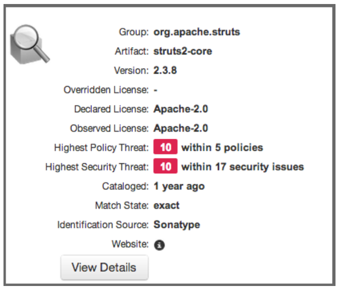
Figure 3.6: Component Information Panel Example

---

**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:



Group: org.apache.struts
Artifact: struts2-core
Version: 2.3.8
Overridden License: -
Declared License: Apache-2.0
Observed License: Apache-2.0
Highest Policy Threat: 10 within 5 policies
Highest Security Threat: 10 within 17 security issues
Cataloged: 1 year ago
Match State: exact
Identification Source: Sonatype
Website:

Figure 3.7: CIP Text

**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.

**Group**
: The group part of the GAV component identifier.

**Artifact**
: The artifact part of the GAV component identifier.

**Version**
: The version part of the GAV component identifier.

**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:

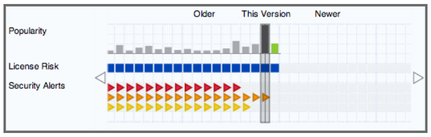


Figure 3.8: CIP Graph

**Popularity**
: The popularity for each version is shown as a bar graph. The larger the graph the more popular the version.

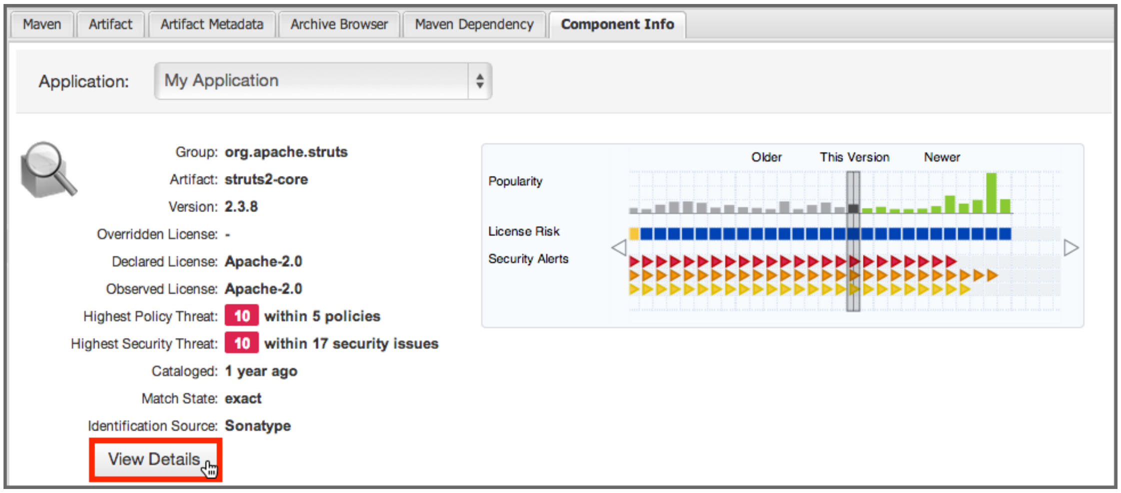
**License Risk**
This will display the license risk based on the application that is selected, and the associated policy and/or license threat groups for that application. Use the application selector to change the application, and corresponding policies the component should be evaluated against.

**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.

## 3.7 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.

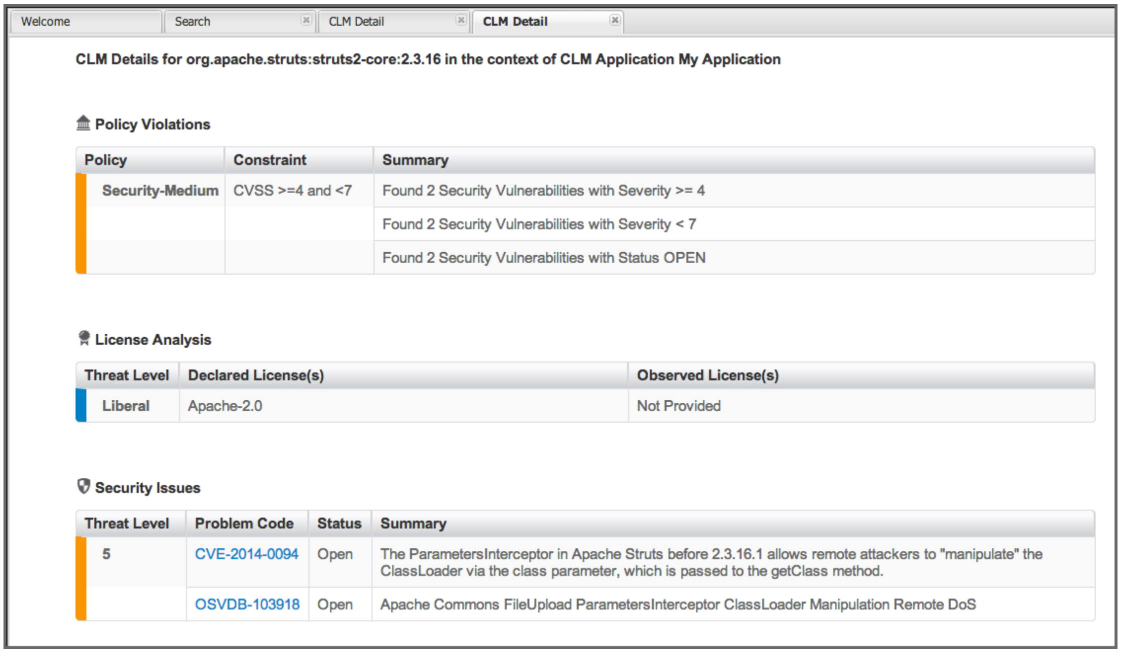
To view these details, click on the *View Details* button located below the Component Information.



Group: org.apache.struts
Artifact: struts2-core
Version: 2.3.8
Overridden License: -
Declared License: Apache-2.0
Observed License: Apache-2.0
Highest Policy Threat: 10 within 5 policies
Highest Security Threat: 10 within 17 security issues
Cataloged: 1 year ago
Match State: exact
Identification Source: Sonatype

Figure 3.9: 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 3.10: 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.

# Chapter 4

# Using CLM for Staging

## 4.1 Introduction

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**

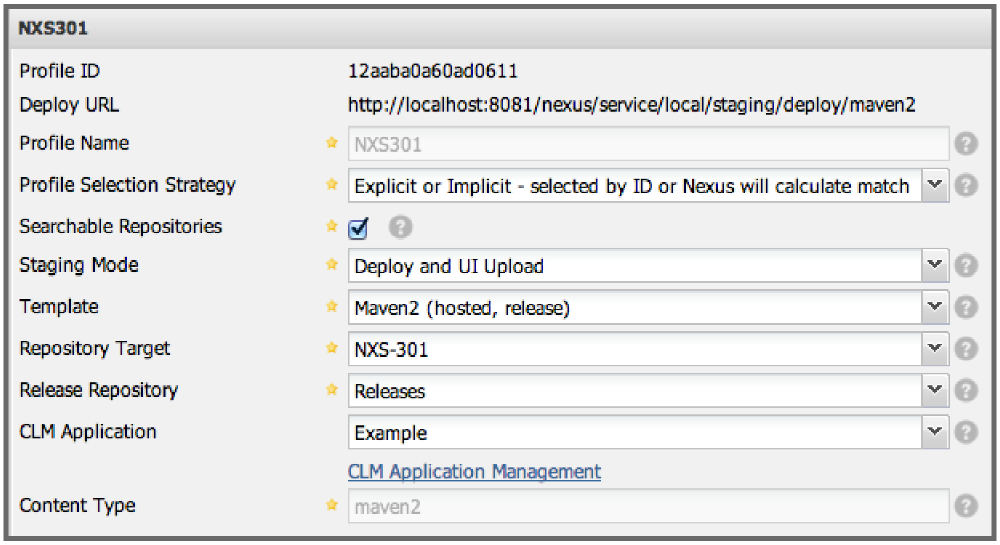
- Created a Staging Profile

**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.

## 4.2 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.



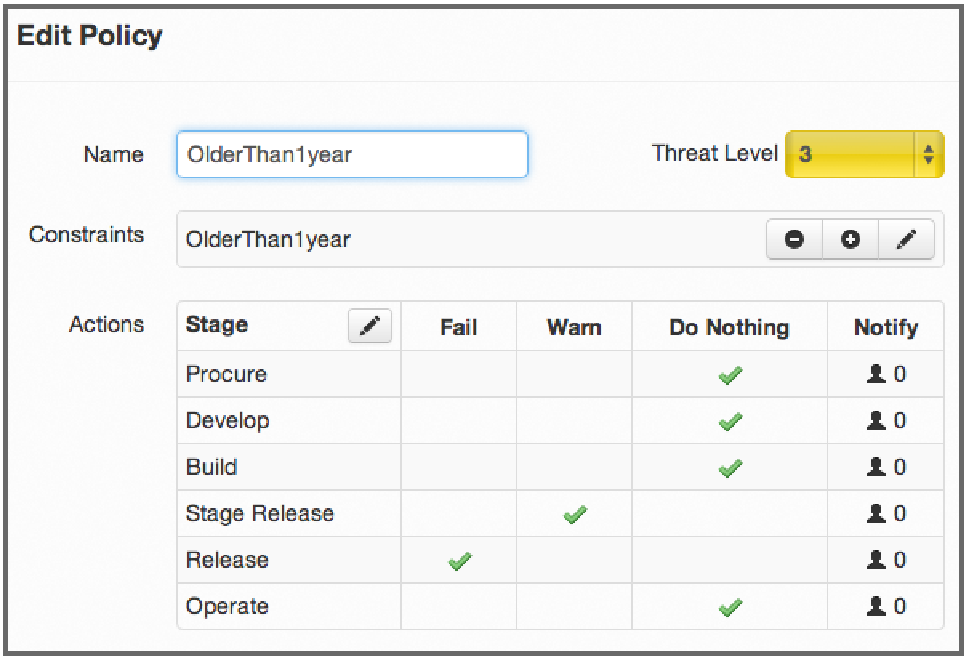
**NXS301**

| | |
|---|---|
| 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 | ✓ |
| 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 4.1: Staging Profile with a CLM Application Configured

## 4.3 Policy Actions

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: OlderThan1year

Threat Level: 3

Constraints: OlderThan1year

Actions

| Stage | Fail | Warn | Do Nothing | Notify |
|---|---|---|---|---|
| Procure | | | ✓ | 0 |
| Develop | | | ✓ | 0 |
| Build | | | ✓ | 0 |
| Stage Release | | ✓ | | 0 |
| Release | ✓ | | | 0 |
| Operate | | | ✓ | 0 |

Figure 4.2: 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 Guide, 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,

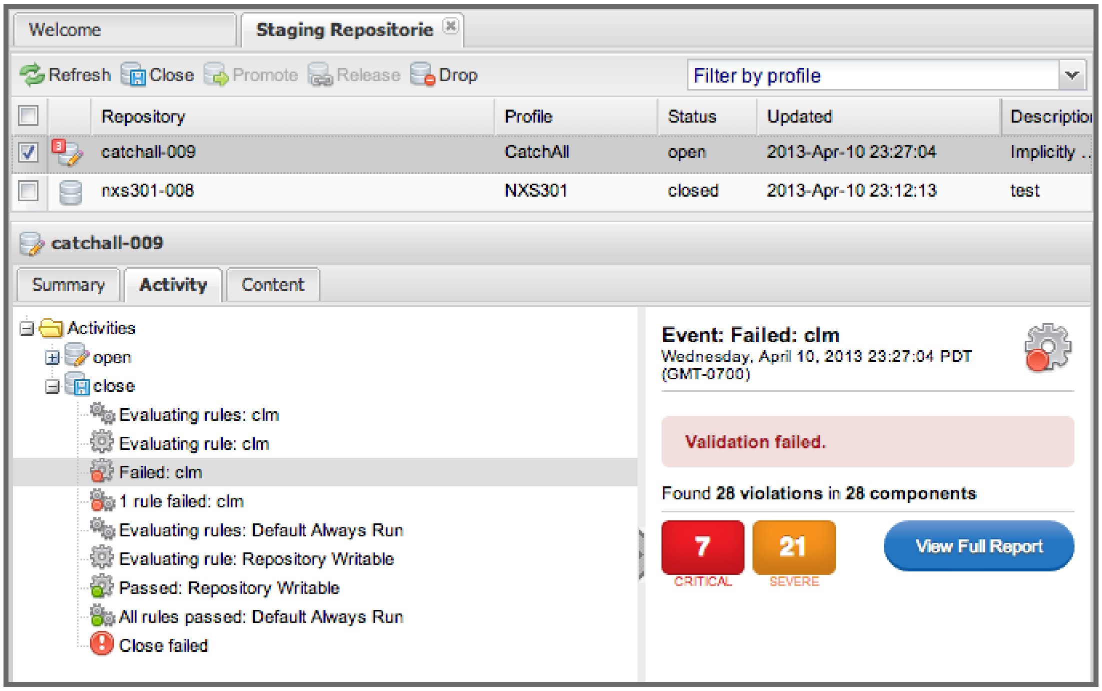
but a warning will be produced.

- If the policy action is set to *Do Nothing*, the staging repository closes successfully regardless of any policy violations.

## 4.4 Release Repository Actions

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 4.3 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.



| | Repository | Profile | Status | Updated | Descriptio |
|---|---|---|---|---|---|
| ☑ | catchall-009 | CatchAll | open | 2013-Apr-10 23:27:04 | Implicitly .. |
| ☐ | nxs301-008 | NXS301 | closed | 2013-Apr-10 23:12:13 | test |

```
Activities
  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
```

**Event: Failed: clm**
Wednesday, April 10, 2013 23:27:04 PDT (GMT-0700)

Validation failed.

Found **28 violations** in **28 components**

7 CRITICAL
21 SEVERE

Figure 4.3: Staging Repository Activity with a CLM Evaluation Failure and Details

# Chapter 5

# CLM Maven Plugin

## 5.1 Introduction

A Sonatype CLM evaluation of a Maven based software project can be assisted by the Sonatype CLM Maven plugin. It can take advantage of the dependency information contained in the project's `pom.xml` files and the information about transitive dependencies available to Maven. It can be run on a command line interface and can therefore be executed on any continuous integration server.

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. However you can also analyze a all modules of a project. This will largely depend on what your CLM policy is enforcing and what you want to validate.

The `index` goal of the plugin allows you to prepare data for analysis with Sonatype CLM for CI.

The `attach` goal aids your integration with Sonatype Nexus CLM Edition and the release process using the staging tools of Nexus.

The `evaluate` goal can trigger an evaluation directly against a Sonatype CLM server.

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:2.1.1:help
```

## 5.2 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 scan.

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 an 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 the CLM Maven plugin is able to create a complete dependency list rather than relying on binary build artifacts.

## 5.3 Skipping CLM Maven Plugin 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>
```