Software Risk Manager Documentation (v2025.9.4)

Contents

Software Risk Manager Plugins Guide

Software Risk Manager Plugins

There are a number of plugins available to make it easier to bring the power of Software Risk Manager to other software development tools, including IDEs, CI/CD Build Systems, and open source Application Security Testing tools. This guide includes instructions for installing and using these plugins.

Note: SAML authentication isn't currently supported for Software Risk Manager plugins.

Jenkins

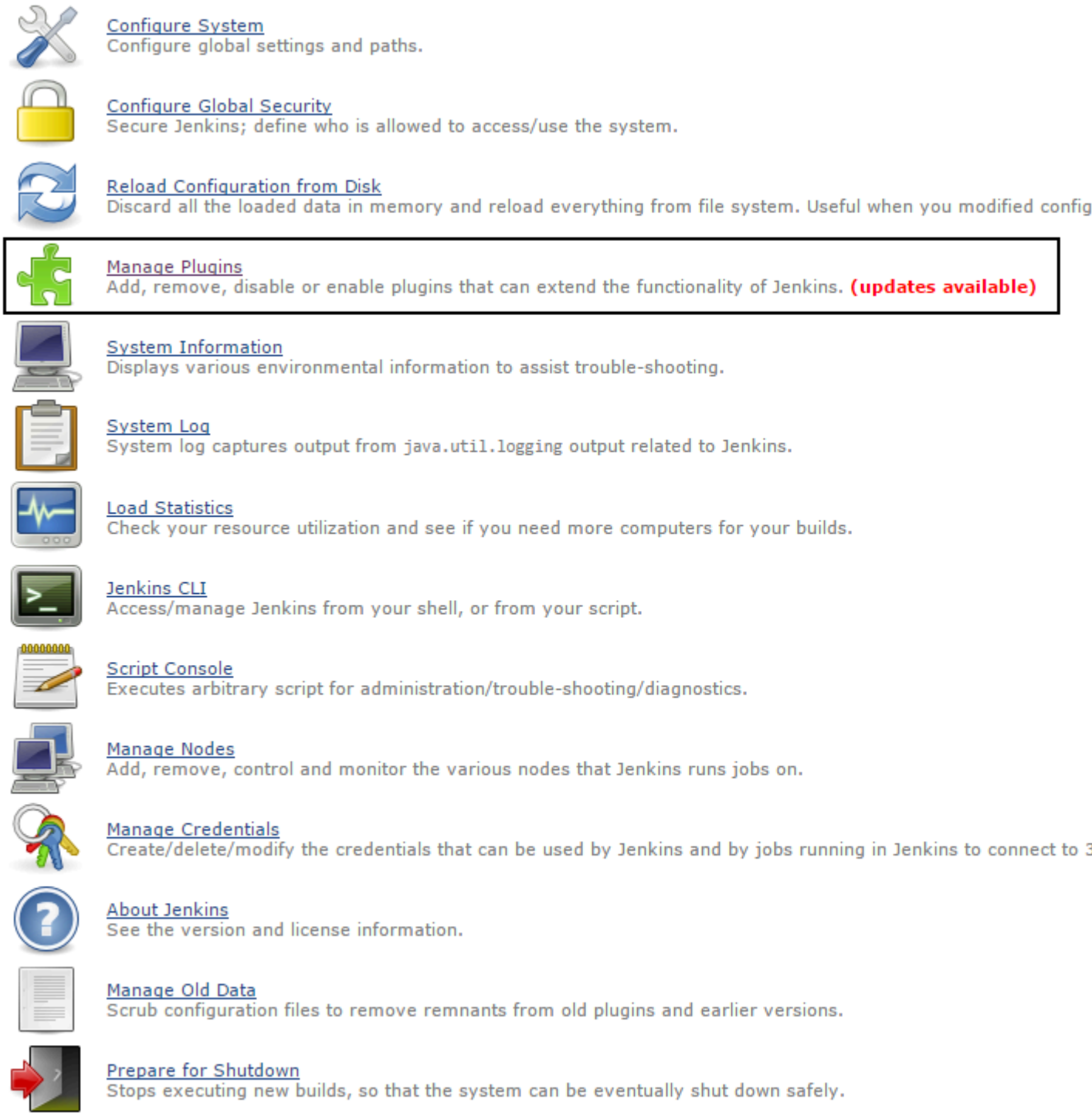
The Software Risk Manager Jenkins plugin integrates the Jenkins continuous integration platform with your Software Risk Manager server. It allows you to push build results to your Software Risk Manager server as part of the build process.

A Software Risk Manager project and an API key are required. The API key must have the `create` role for the project.

This section of the Plugins Guide explains how to install and use the Jenkins plugin. For more information you may visit the Official Code Dx Jenkins Wiki Page or our GitHub Repository. This plugin is open source and we welcome community involvement.

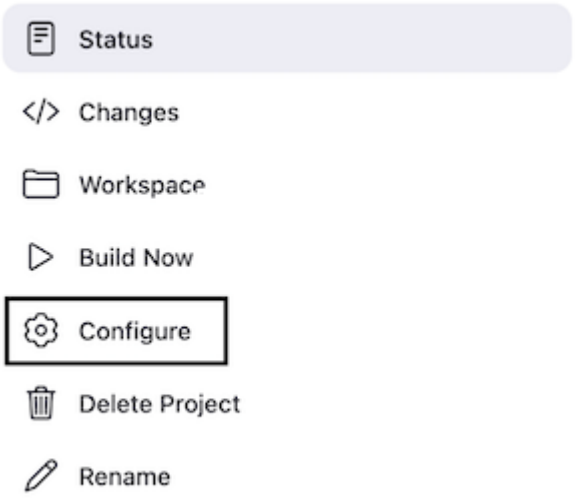
Installation

The Software Risk Manager Jenkins plugin is available for installation through the Jenkins plugin management page. You must be running Jenkins version 2.200 or later, or one of the LTS releases listed here.

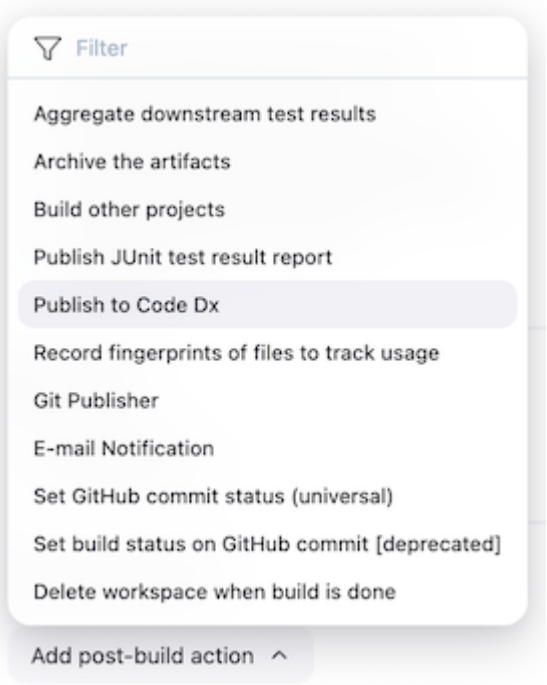


Job Configuration

The first thing you should do is configure your Jenkins Job to publish to Software Risk Manager. Instructions for Freestyle and Pipeline projects are fairly similar; a note on configuring for Pipelines projects can be found below.



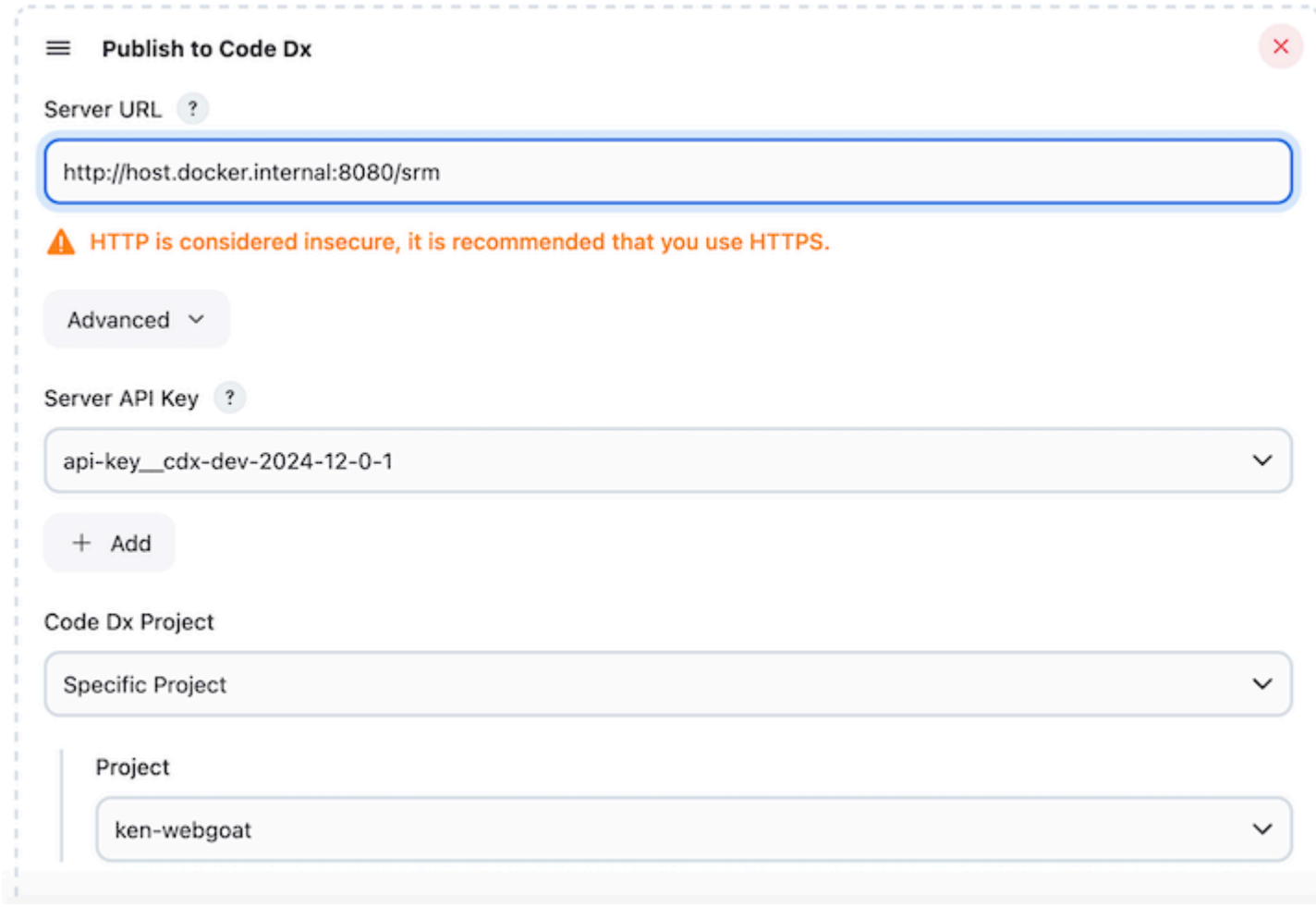
This is accomplished on the configuration page by going to Post-build Actions (toward the bottom) and selecting the *Publish to Code Dx* option from the Add post-build action button.



You can use the new action to setup different options related to Software Risk Manager.

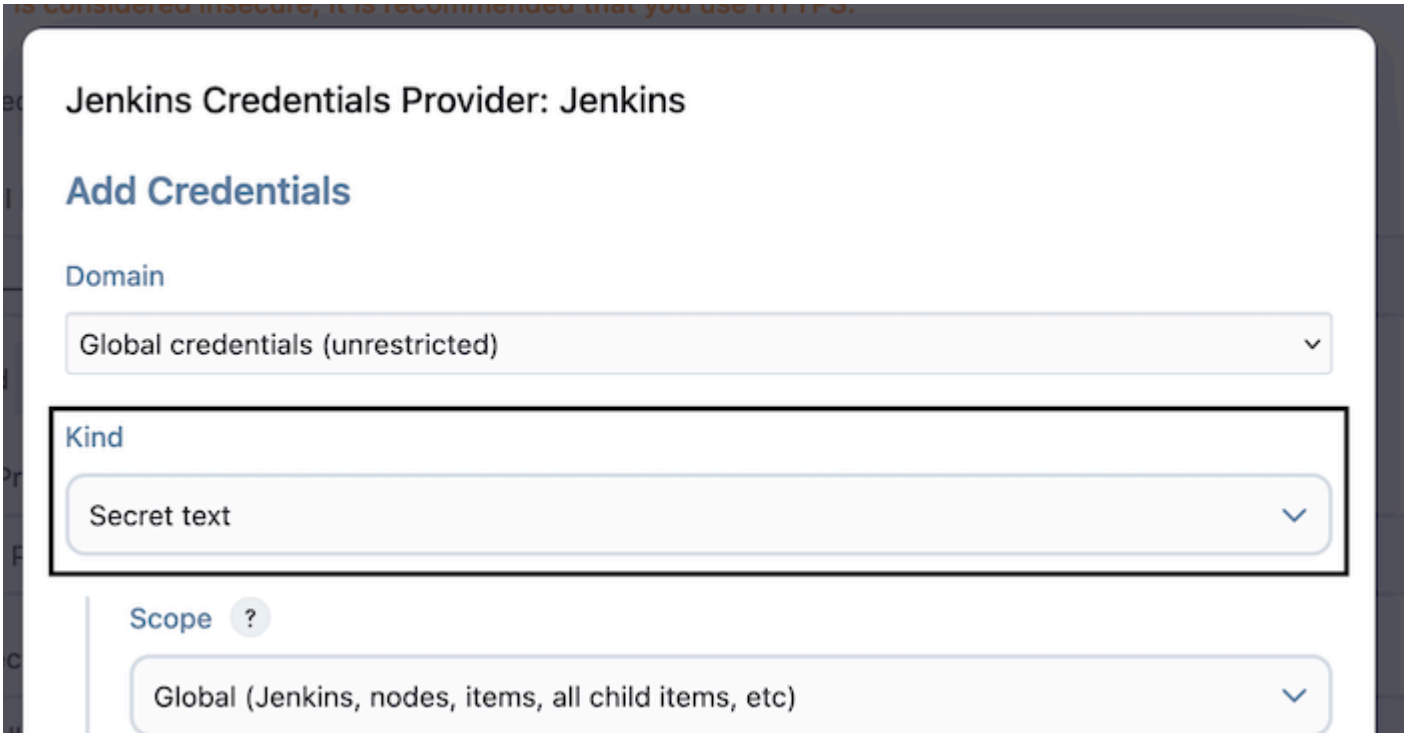
Publishing

The *Server URL*, *Server API Key*, and *Code Dx Project* fields are required for publishing. Ask your Software Risk Manager administrator to generate the server API key that has the `create` role for the project it needs to interact with.



It is highly recommended that you specify an HTTPS URL, since using HTTP is insecure. If you receive a warning regarding an invalid certificate, refer to the section on self-signed certificates.

The Server API Key must be stored in Jenkins as a "Secret Text" credential accessible to the Jenkins project.

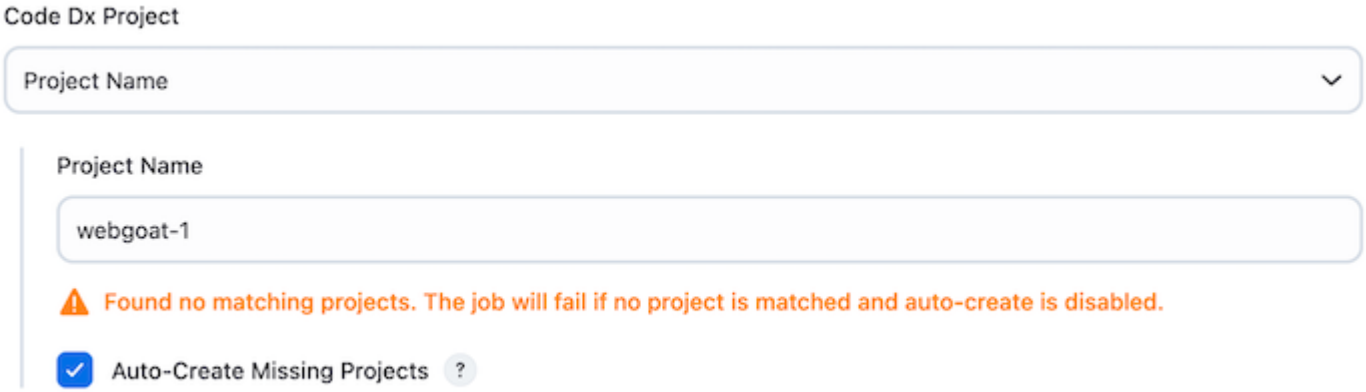


Once the *Server URL* and *Server API Key* fields are populated, the *Code Dx Project* dropdown can be used to select the Software Risk Manager project which will receive analyses. This dropdown has the options "Specific Project" and "Named Project".

The default option, "Specific Project", provides a simple dropdown for selecting an existing project in Software Risk Manager. The exact, selected project will always be used when publishing results from the job, even if the project is renamed.

The "Project Name" option allows you to specify a Software Risk Manager project by name. This must resolve to exactly one SRM project when the Jenkins job runs, otherwise the job may fail.

The "Project Name" option also offers a "Auto-Create Missing Projects" field option, which allows the plugin to create the project in Software Risk Manager if it does not yet exist. If using Software Risk Manager version 2022.4.3 or later, the default branch for the new project will be set to the "Base Branch" (see below) if specified. *Note: The API Key must have either the "Administrator" or "Project Administrator" role to create projects.*

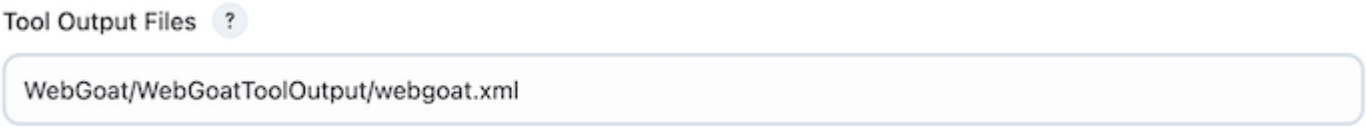


The *Source and Binary Files* field allows you to identify the files in the job workspace for Software Risk Manager to analyze. The format of this field is a comma-separated list of Ant glob file location patterns. You can populate this list by specifying the files (relative to the workspace) that will be sent to Software Risk Manager. By default, this field is set to `**` (all files).

The *Include Git Source* option can be used to have Software Risk Manager download the latest content from the Software Risk Manager project's configured Git source as part of the analysis. Bundled tools will be ran on this content as usual.

You can use the field "Specific Branch Name" to change which branch is fetched. If left empty, the default branch for the project will be used.

In addition to generic listing formats, Software Risk Manager supports importing the results of more than 100 commercial and open source analysis tools. This importing feature is defined in the Jenkins plugin through the *Tool Output Files* field, where you specify a comma-separated list of the paths and filenames of each output file. These paths should be relative to the job workspace.



Software Risk Manager users have access to the "First Seen by SRM" and "Last Modified" filters on Software Risk Manager's Findings page. Each analysis in those filters can have an Analysis Name, which is directly related to the *Analysis Name* field in Jenkins. The *Analysis Name* field lets you set a "name" for each Software Risk Manager analysis published from Jenkins. You can use build/environment variables to construct a different name for each analysis. For example, `Build #${BUILD_NUMBER}` creates the analysis name "Build #26" for the 26th build of the project. You can construct links using a syntax similar to markdown, i.e., `[link text](link url)`. This also works with build variables: `Build [#${BUILD_NUMBER}]($BUILD_URL)`. Some Jenkins plugins, like the Git plugin, provide "macros" which allow for some additional customization. In this example, the analysis name will be set to the first eight characters of the git commit hash: `${GIT_REVISION, length=8}`. For more information about "macros", see the Token Macro Plugin Wiki. *Note: the analysis name feature is only supported by Code Dx versions 2.4.0 and up. If the server you plan to publish to is older than version 2.4.0, the analysis name will be ignored.*

The *Target Branch* and *Base Branch* options allow you to set which branch will be used when storing results in the Software Risk Manager project. The Target Branch typically matches the branch currently checked out in the Jenkins build, but this isn't a requirement. If the Target Branch is not defined, results will be stored in the project's default branch.

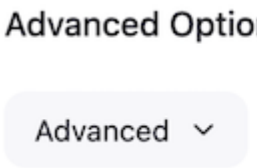
If the Target Branch doesn't exist yet, it will be created in Software Risk Manager using the given Base Branch as its parent. The Base Branch is only used for creating new branches, and can be left empty if the Target Branch already exists.

The Target Branch and Base Branch fields only affect data storage within Software Risk Manager and do not affect the Jenkins build. When Failure or Unstable conditions are configured in the Jenkins plugin, findings will be fetched from the Target Branch to validate those conditions (or the default branch if unspecified).

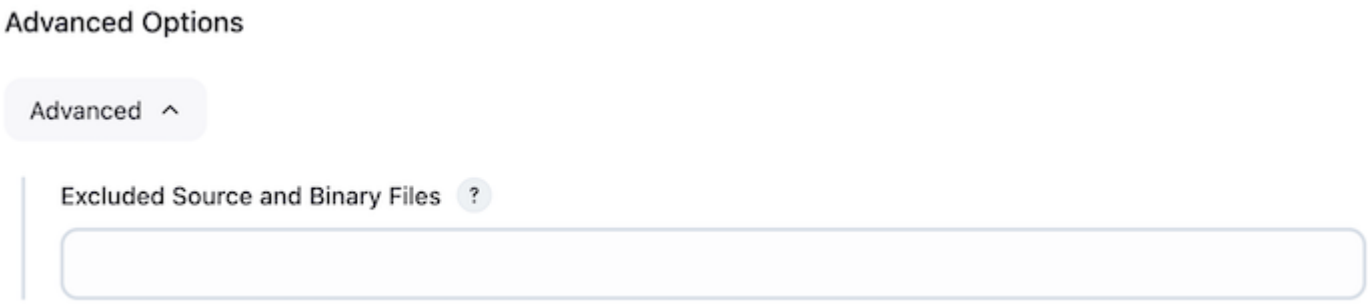
Note: while branching was added in Code Dx 2022.4.0, this plugin requires 2022.4.3 or later when using the branching feature. If the server you plan to publish to is older than version 2022.4.3, the Target Branch and Base Branch fields will be ignored.

The *Error Handling* field lets you change plugin behavior when an error occurs during communication with the Software Risk Manager server. This behavior is applied if there are connection issues or the analysis fails, but is not applied for configuration errors such as an invalid Software Risk Manager project.

In the Advanced Options section, which is located at the bottom, you may specify source and/or binary locations to exclude from the analysis.



Clicking the *Advanced...* button will allow you to enter the files you would like to exclude from the build. These files are also Ant glob patterns.



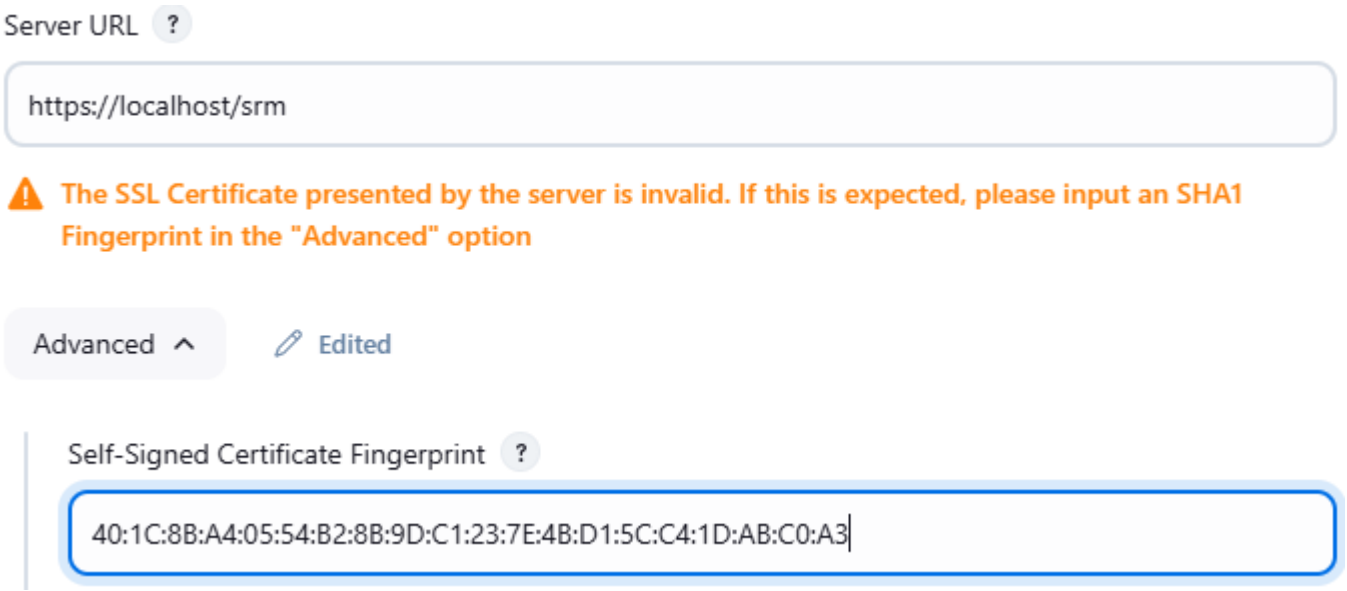
Handling a Self-Signed Certificate

If the server hosting Software Risk Manager is using a self-signed certificate, you'll receive a warning:

Clicking the *Advanced* button will allow you to populate the *Self-Signed Certificate Fingerprint* field with the SHA1 fingerprint of the self-signed certificate used by the server. Contact your Software Risk Manager administrator for the correct value. Or you can navigate to your installation of Software Risk Manager in a browser, and obtain the fingerprint by following the instructions for your particular browser:

- Chrome: Click the lock icon or "Not secure" message next to the URL. If the connection is not secure, click the "Certificate is not valid" section to display a pop-up window. Otherwise, if the connection *is* secure, click the "Connection is secure" section and then the "Certificate is valid" section to display a pop-up window. The SHA1 Fingerprint can be found near the bottom of the pop-up window.
- Edge: Click the lock icon or "Not secure" message to the left of the URL, click on the section "Connection is/isn't secure", and click the certificate icon at the top-right of the window. The SHA1 Fingerprint can be found near the bottom.
- Firefox: Click the lock icon next to the URL, click the section "Connection (isn't) secure", and click the *More information* button. In the new window, click the "View Certificate" button which will open a new tab. The SHA1 Fingerprint can be found in the "Fingerprints" section.
- Safari: Click the lock icon next to the URL, click *Show Certificate*, expand the *Details* section, and the SHA1 Fingerprint can be found near the bottom.

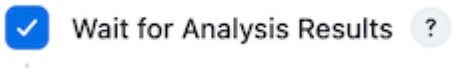
Once you have the correct fingerprint, populating the *Self-Signed Certificate Fingerprint* field will allow you to proceed.



Waiting for Analysis Results

When performing an analysis, the Jenkins Software Risk Manager publisher will zip up the specified workspace files and send them to the Software Risk Manager server. By default, Jenkins will not wait for the results of the analysis.

In some cases you will want to wait for the analysis to complete so you may consider the Jenkins job a success or failure. To take this even further, a team may also want the resulting Software Risk Manager analysis data to influence the state of the build. Additionally, you may want to see a summary of the Software Risk Manager build and analysis results within Jenkins, including the resulting Software Risk Manager tables and graphs. This is all possible by selecting the *Wait for Analysis Results* checkbox.



Upon enabling this option, a new set of fields will be shown on the configuration page. These fields are categorized into three sections: *Policy Behavior*, *Build Failure Conditions*, *Build Unstable Conditions*, and *Graph Options*.

Policy Behavior

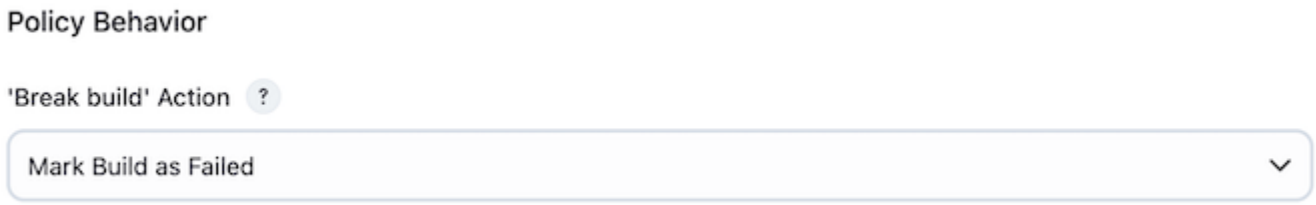
The Software Risk Manager Jenkins plugin can check the project for policy violations after the analysis has completed and then change the build result if there were violations. For a policy violation to affect the Jenkins plugin, the violations must meet the following conditions:

1. Be associated with the Software Risk Manager project
2. Have at least one rule where the action is set to "Break build"
3. Meet the "fix by" criteria of that policy rule
4. Have a violation of that specific rule

The "'Break build' Action" field allows you to change the build result that is used when the conditions listed above are met. However, when set to "No Action," the plugin will not check for policy violations.

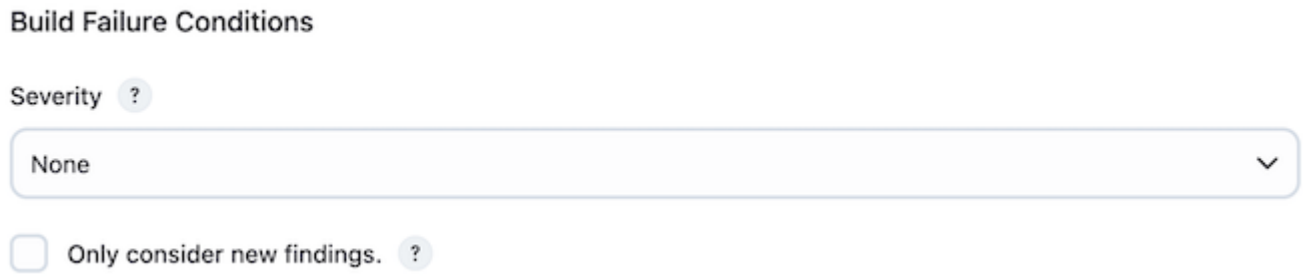
See the section on Policy Configuration in the *Software Risk Manager User Guide* for more information.

Note: the Policies feature is only supported by Code Dx versions 2023.1.0 and up. If the server you plan to publish to is older than version 2023.1.0, the field will be ignored.

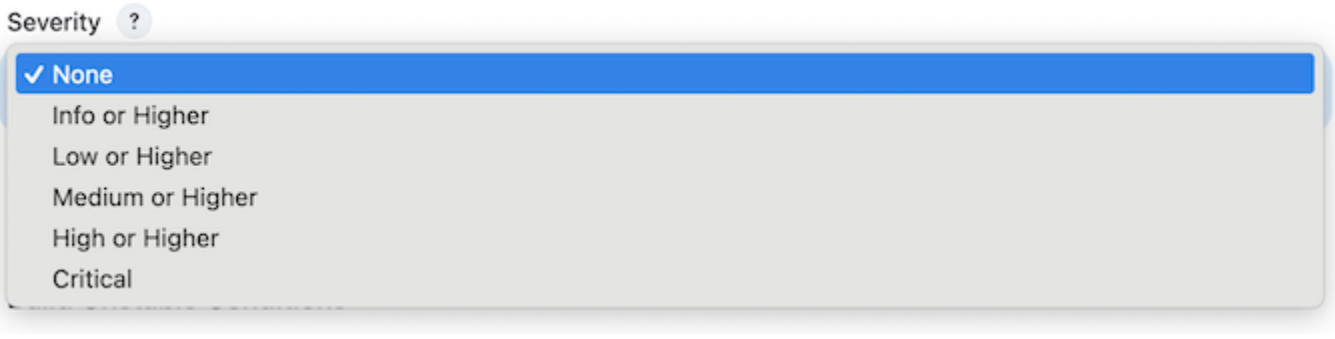


Build Failure Conditions

The *Build Failure Conditions* section allows you to configure the requirements upon which the build will be considered a failure. It is important to note that not all findings will be included in this check. By default, only findings with a status of `Assigned`, `To Be Fixed`, `Reopened`, and `Unresolved` will be considered. To only check findings created during the current build, you can select the *Only Consider New Findings* checkbox.

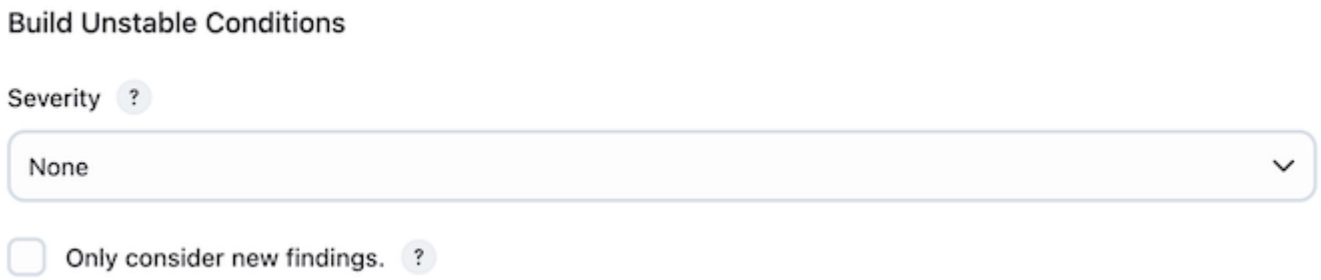


The severity dropdown specifies the range of severities that will cause the build to fail. That is, the build will be considered a failure if one or more findings are detected with the selected severity range.



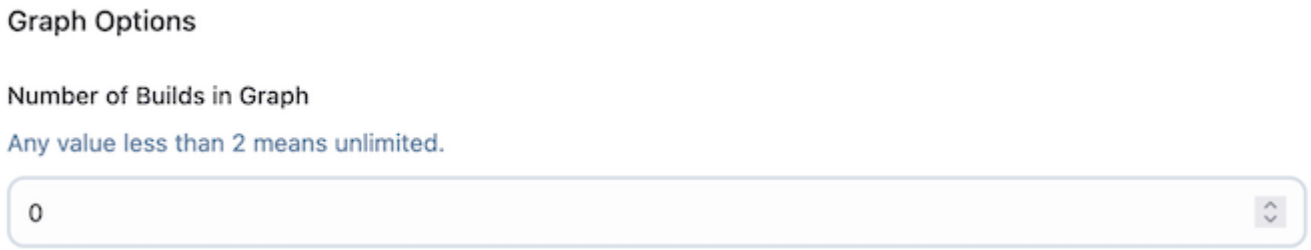
Build Unstable Conditions

The *Build Unstable Conditions* section is identical to the previous section, but configures the requirements upon which the build will be considered unstable. The severity dropdown has the same options as the *Build Failure Conditions* section.



Graph Options

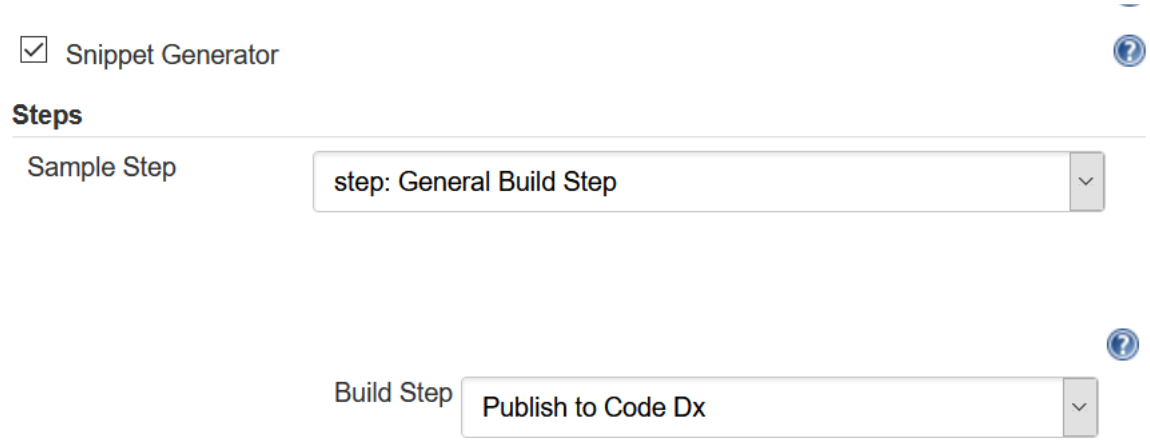
The Software Risk Manager Jenkins plugin will show some helpful graphs on the Job and Build pages when the *Wait for Analysis Results* option is enabled. The number of datapoints in these graphs is configurable using the *Number of Builds in Graph* field. To show an unlimited number of datapoints, set this field to a value less than 2.



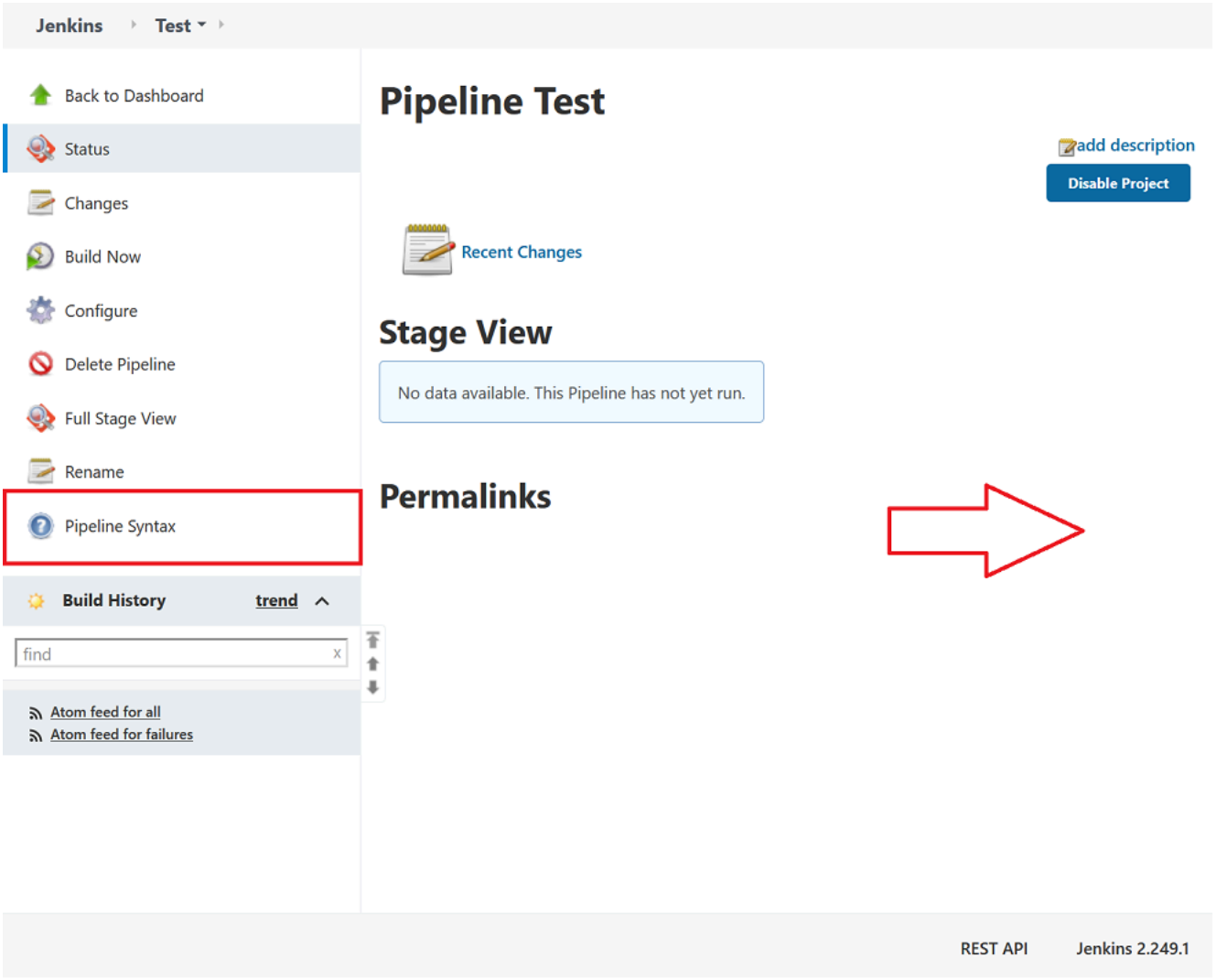
Configuring for Pipelines

When configuring your Pipeline project, use the Snippet Generator to easily create a command.

In older versions of Jenkins this can be found at the bottom of the Configuration page for the project.



In newer versions, it's labeled as "Pipeline Syntax" in the menu on the left, containing other project options.



Set the "Sample Step" to "step: General Build Step", and change "Build Step" to "Publish to Software Risk Manager".

Once selected, you will be given the same configuration options as listed in the previous sections. Behavior of the Pipeline plugin is the same as the Freestyle one. Customize these options as necessary and click "Generate Pipeline Script" to create a snippet you can use in your pipeline.

Analysis Results

If the Software Risk Manager Jenkins plugin is configured to wait for analysis results before allowing the build to complete, it will also show tables and charts on the Job and Build pages.

Finding Tables

The Job and Build pages will display tables that provide a summary of the findings.

These tables show the number of findings for each severity and status. A delta between the current build and previous build will also be shown, if applicable.

Under the tables there is a link to view the latest results within the Software Risk Manager application.

Finding Graphs

The Job page will show graphs of the findings according to severity and status.

TeamCity

The Software Risk Manager TeamCity plugin integrates the TeamCity continuous integration platform with your Software Risk Manager server. It allows you to push build results to your Software Risk Manager server as part of the build process.

A Software Risk Manager project and an API key are required. The API key must have the `create` role for the project.

This section of the Plugins Guide explains how to install and use the TeamCity plugin. For more information you may visit our GitHub Repository. This plugin is open source and we welcome community involvement.

Installation

For instructions on how to install the Software Risk Manager TeamCity plugin, please see TeamCity's official documentation.

The plugin is available on GitHub and TeamCity's plugin marketplace.

Configuration

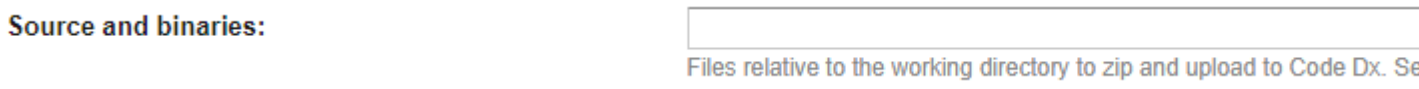
After installing the plugin, the next step is to add a Software Risk Manager Build Step to your project. Navigate to the "Build Steps" page for your project and click the "Add build step" button. The "New Build Step" page will display and a dropdown will ask you to "Choose build runner type". After selecting the "Code Dx" option, the configuration fields will display.

Publishing

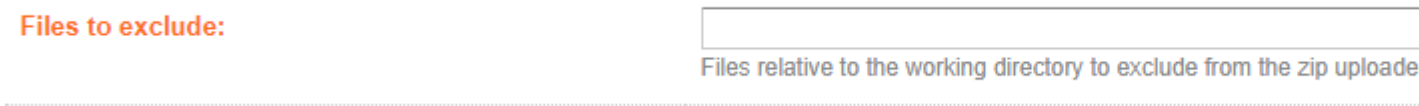
The *Code Dx URL*, *API key*, and *Project* fields are required for publishing. Ask your administrator to generate an API key that has the `create` role for the project it needs to interact with.

Once the *Code Dx URL* and *API key* fields are populated, the *Project* dropdown will automatically list the projects available to the API key. If you receive a warning regarding an invalid/untrusted certificate, refer to the section on self-signed certificates.

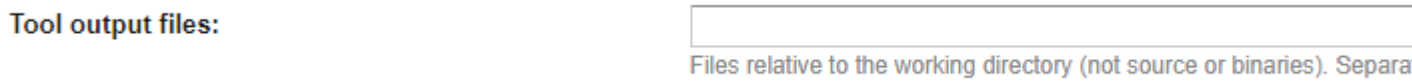
The *Source and binary* field allows you to identify the files in the job workspace for Software Risk Manager to analyze. The format of this field is a comma-separated list of Ant glob file location patterns. You can populate this list by specifying the files (relative to the workspace) that will be sent to Software Risk Manager.



The *Files to exclude* field is an advanced option that can be displayed by clicking "Show advanced options" near the bottom of the page. This field allows you to specify files to omit in the source and binaries zip file that is uploaded to Software Risk Manager. Ant glob file location patterns are also supported.



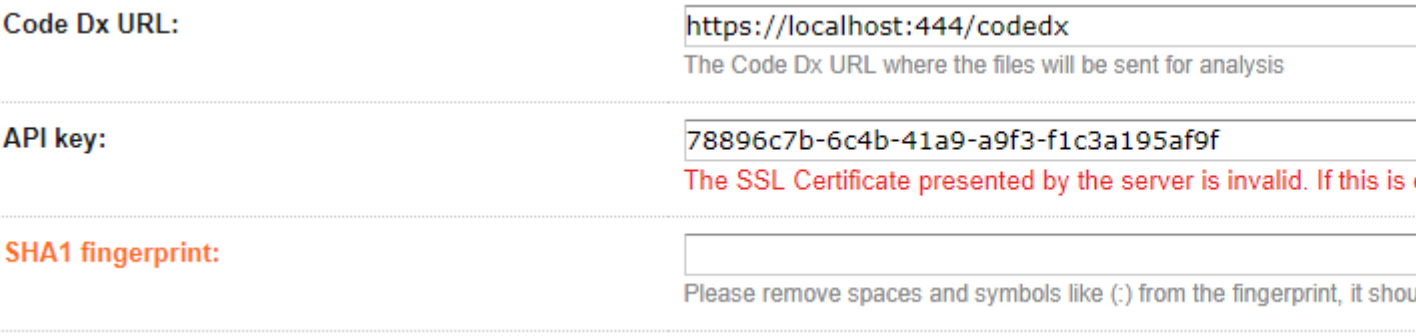
Software Risk Manager supports importing the results of more than 70 commercial and open source analysis tools, in addition to generic listing formats. This feature is supported in the TeamCity plugin via the *Tool output files* field, where you specify a comma-separated list of paths and filenames of each output file.



The *Analysis Name* field allows you to name the analyses performed by TeamCity. You can find the analysis names on the "First Seen by SRM" and "Last Modified" filters on Software Risk Manager's Findings page. You can use build/environment variables to construct a different name for each analysis. For example, `Build #%build.number%` creates analysis name "Build #26" for the 26th build of the project. Clicking the icon next to the input control will list possible values for parameter references. You can also construct links using a syntax similar to markdown, i.e., `[link text](link url)`.

Handling a Self-Signed Certificate in TeamCity

If the server hosting Software Risk Manager is using a self-signed certificate, you'll receive a warning:



Clicking *Show advanced options* will allow you to populate the *Self-Signed Certificate Fingerprint* field with the SHA1 fingerprint of the self-signed certificate used by the server. Contact your Software Risk Manager administrator for the correct value. Or you can navigate to your installation of Software Risk Manager in a browser, and obtain the fingerprint by following the instructions for your particular browser:

- Chrome: Click the lock icon next to the URL, choose the *Connection* tab and follow the link for "Certificate Information". Expand the "Details" section; the SHA1 fingerprint is near the bottom.
- Firefox: Click the lock icon next to the URL, choose the *Security* tab, and click the *View Certificate* button. The SHA1 Fingerprint should be at the bottom of the resulting window.
- Safari: Click the lock icon next to the URL, click *Show Certificate*, expand the *Details* section, and the SHA1 Fingerprint can be found near the bottom.
- Internet Explorer: Click on the *Certificate Error* text to the right of the URL, select the *Details* tab, and find *Thumbprint* and *Thumbprint algorithm* fields. Ensure that the value of the *Thumbprint algorithm* field is "sha1" and use the value of the *Thumbprint* field.

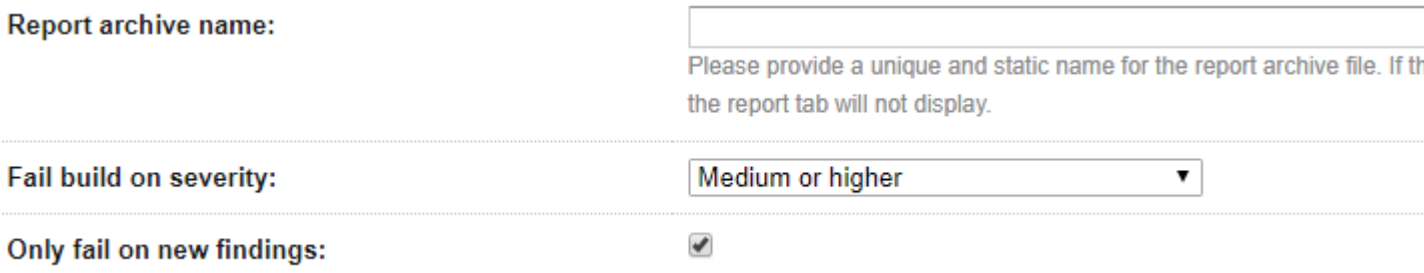
Once you have the correct fingerprint, populating the *Self-Signed Certificate Fingerprint* field will allow you to proceed.

Waiting for Analysis Results

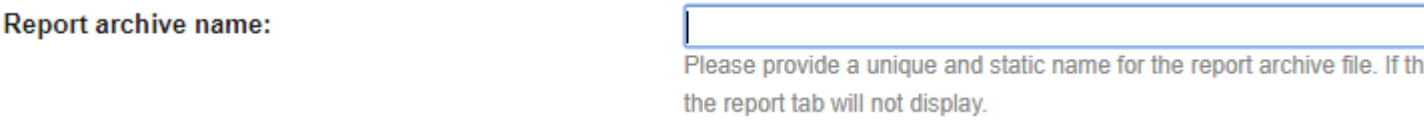
When performing an analysis, the TeamCity build runner will zip up the specified workspace files and send them to the Software Risk Manager server. By default, TeamCity will not wait for the results of the analysis.

In some cases, you will want to wait for the analysis to complete so you may consider the TeamCity job a success or failure. To take this even further, a team may also want the resulting Software Risk Manager analysis data to influence the state of the build. Additionally, you may want to see a summary of the Software Risk Manager build and analysis results within TeamCity, including the resulting Software Risk Manager tables. This is all possible by selecting the *Wait for results* checkbox.

Upon enabling this option, the fields below will be enabled.



The *Report archive name* field allows you to name the build artifact that the build runner produces. The artifact is a zip archive that contains an HTML file. This zipped HTML file can be used to configure a build report tab. The build report tab will display the build statistics tables. If this field is left blank, the build artifact will not be generated.



The *Fail build on severity* field allows you to have the build marked as "failed" if Software Risk Manager reports your project contains findings that match the chosen option. This field is defaulted to "None" and the build step will finish upon successfully uploading all files to Software Risk Manager. If a different option is selected, the build step will not finish until the analysis is complete.

The *Only fail on new findings* option, when checked, means that the build will only be marked as failed if *new* findings that match the *Fail build on severity* option are reported.

Analysis Results

If the Software Risk Manager TeamCity plugin is configured to wait for analysis results, and a name has been provided to the *Report archive name* field, you can configure a build report tab for your TeamCity project.

To create a build report tab for Software Risk Manager in TeamCity, please follow these steps:

1. Configure the Software Risk Manager build runner so that it waits for analysis results
2. Fill out the *Report archive name* field
3. Run the build
4. On the *Edit Build* page, add the build artifact to your build artifact paths. The name of the zip archive should match the configured *Report archive name*
5. On the *Edit Project* page, navigate to the *Report Tabs* section
6. On the *Report Tabs* section, click the *Create new build report tab* button
7. In the dialog that opens, fill out the *Tab Title* field
8. In the *Start page* field, enter the path to the report. This will look something like [report-archive-name].zip!codedx-teamcity-report.html
9. Click the *Save* button
10. Run a new build

The report tab will appear on the *Build Results* page. It contains a link and tables. Clicking the link will open the latest results within the Software Risk Manager application. The tables show the number of findings for each severity and status. If applicable, a delta between the current build and previous build is included.

Bamboo

The Software Risk Manager Bamboo plugin integrates the Bamboo continuous integration platform with your Software Risk Manager server. It allows you to push build results to your Software Risk Manager server as part of the build process.

A Software Risk Manager project and an API key are required. The API key must have the `create` role for the project.

This section of the Plugins Guide explains how to install and use the Bamboo plugin. For more information you may visit our GitHub Repository. This plugin is open source and we welcome community involvement.

Installation

For instructions on how to install the Software Risk Manager Bamboo plugin, please see Atlassian's official documentation.

The plugin is available on GitHub.

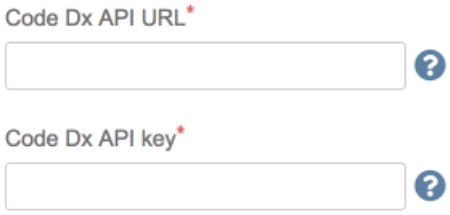
Configuration

After installing the plugin, the next step is to add a Software Risk Manager Scan Task to your project. Navigate to the "Edit Build Tasks" page for a job for your project and click the "Add task" button. After selecting the "Code Dx Scan Task" option, the configuration fields will display.

Publishing

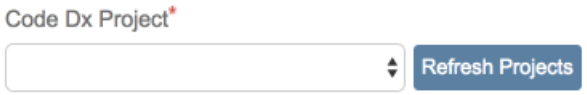
The *Code Dx URL*, *API key*, and *Project* fields are required for publishing. Ask your administrator to generate an API key that has the `create` role for the project it needs to interact with.

The first thing that should be set up are the Software Risk Manager Server Settings.



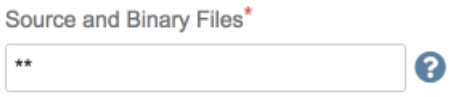
Self-signed certificates can also be set up here. If using self-signed certificates, please refer to the section on self-signed certificates. Otherwise, the field may be left blank.

Once the *Code Dx URL* and *API key* fields are populated, Click the *Refresh Projects* button next to the *Project* dropdown.

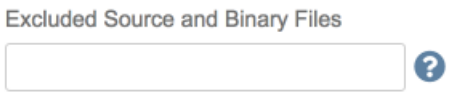


This will populate the list with the Software Risk Manager projects available to the given API key.

The *Source and Binary Files* field allows you to identify the files in the job workspace for Software Risk Manager to analyze. The format of this field is a comma-separated list of Ant glob file location patterns. You can populate this list by specifying the files (relative to the workspace) that will be sent to Software Risk Manager.

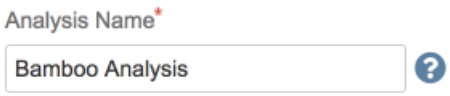


The *Excluded Source and Binary Files* field allows you to specify files to omit in the source and binaries zip file that is uploaded to Software Risk Manager. Ant glob file location patterns are also supported.



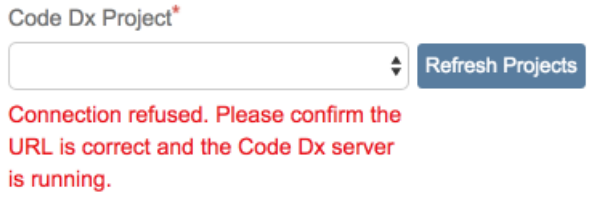
Software Risk Manager supports importing the results of more than 70 commercial and open source analysis tools, in addition to generic listing formats. This feature is supported in the Bamboo plugin via the *Tool Output Files* field, where you specify a comma-separated list of paths and filenames of each output file.

The *Analysis Name* field allows you to name the analyses performed by Bamboo. You can find the analysis names on the "First Seen by SRM" and "Last Modified" filters on Software Risk Manager's Findings page. You can use build/environment variables to construct a different name for each analysis. For example, `Build #${bamboo.buildNumber}` creates analysis name "Build #26" for the 26th build of the project. For information on what environment variables are available, see Bamboo variables.



Handling a Self-Signed Certificate in Bamboo

If the server hosting Software Risk Manager is using a self-signed certificate, you'll have to do some configuration or the connection to Software Risk Manager will be refused:



Bamboo needs to know that your Software Risk Manager server is trusted. Populate the *Self-Signed Certificate Fingerprint* field with the SHA1 fingerprint of the self-signed certificate used by the Software Risk Manager server. Contact your Software Risk Manager administrator for the correct value. You may also navigate to your installation of Software Risk Manager in a browser, and obtain the fingerprint by following the instructions for your particular browser:

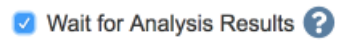
- Chrome: Click the lock icon next to the URL, choose the *Connection* tab and follow the link for "Certificate Information". Expand the "Details" section; the SHA1 fingerprint is near the bottom.
- Firefox: Click the lock icon next to the URL, choose the *Security* tab, and click the *View Certificate* button. The SHA1 Fingerprint should be at the bottom of the resulting window.
- Safari: Click the lock icon next to the URL, click *Show Certificate*, expand the *Details* section, and the SHA1 Fingerprint can be found near the bottom.
- Internet Explorer: Click on the *Certificate Error* text to the right of the URL, select the *Details* tab, and find *Thumbprint* and *Thumbprint algorithm* fields. Ensure that the value of the *Thumbprint algorithm* field is "sha1" and use the value of the *Thumbprint* field.

Once you have the correct fingerprint, populating the *Self-Signed Certificate Fingerprint* field will allow you to proceed.

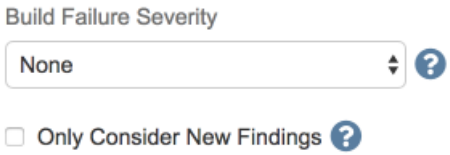
Waiting for Analysis Results

When performing an analysis, the Bamboo build task will zip up the specified workspace files and send them to the Software Risk Manager server. By default, Bamboo will not wait for the results of the analysis.

In some cases, you will want to wait for the analysis to complete so you may consider the Bamboo build a success or failure.



Upon enabling this option, the fields below will appear.



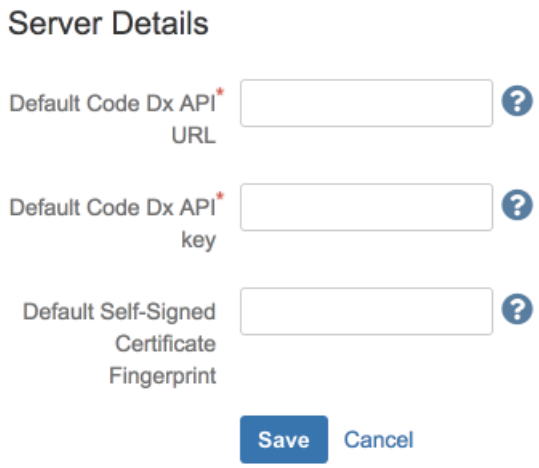
The *Build Failure Severity* field allows you to have the build marked as "failed" if Software Risk Manager reports your project contains findings that match the chosen option. This field is defaulted to "None" and the

build step will finish upon successfully uploading all files to Software Risk Manager. If a different option is selected, the build step will not finish until the analysis is complete.

The *Only Consider New Findings* option, when checked, means that the build will only be marked as failed if *new* findings that match the *Build Failure Severity* option are reported.

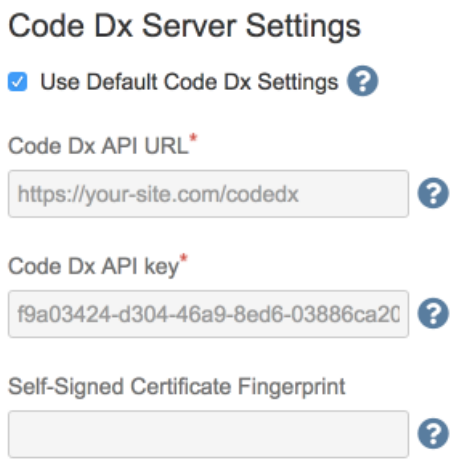
Software Risk Manager Server Defaults

If your team uses one Software Risk Manager server for multiple projects, it may be worth setting up defaults for that server. This will make it easier in the future to create new Bamboo tasks that publish to that server. You may set defaults by clicking on the `Code Dx Plugin` link in the `ADD-ONS` section of the `Bamboo administration`.



Here, you may set defaults for the Software Risk Manager URL, API key, and optionally the self-signed certificate fingerprint.

Once configured, you may opt to use the default server in your task configuration.



Eclipse

The Software Risk Manager Eclipse plugin streamlines the use of Software Risk Manager within the Eclipse IDE. Developers can push out new builds for Software Risk Manager analyses and the results can be viewed from within the IDE.

A Software Risk Manager project is required. To allow users access to the project, they must be assigned to the project where the user roles are consistent with those within Software Risk Manager.

This section of the guide explains how to install and use the Eclipse plugin. For more information you may visit the Eclipse Marketplace.

This plugin supports the Eclipse versions Neon and later.

Installation

The Eclipse plugin is installed through the usual plugin installation method. Navigate to *Help -> Install New Software*. Then click the *Add* button to add a new update site.

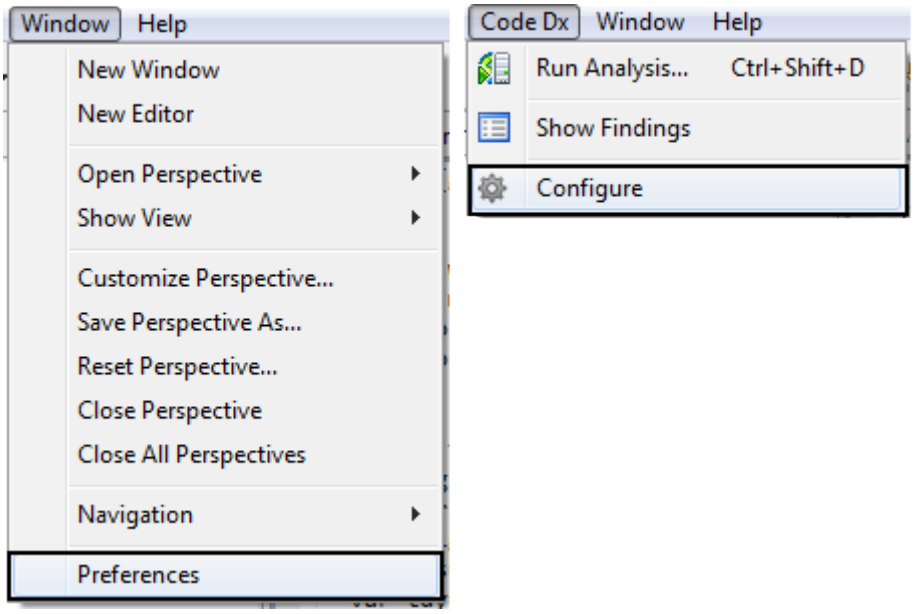
Enter *Code Dx* for *Name*. Enter https://eclipse.codedx.com/ for *Location*.

Then click *OK*.

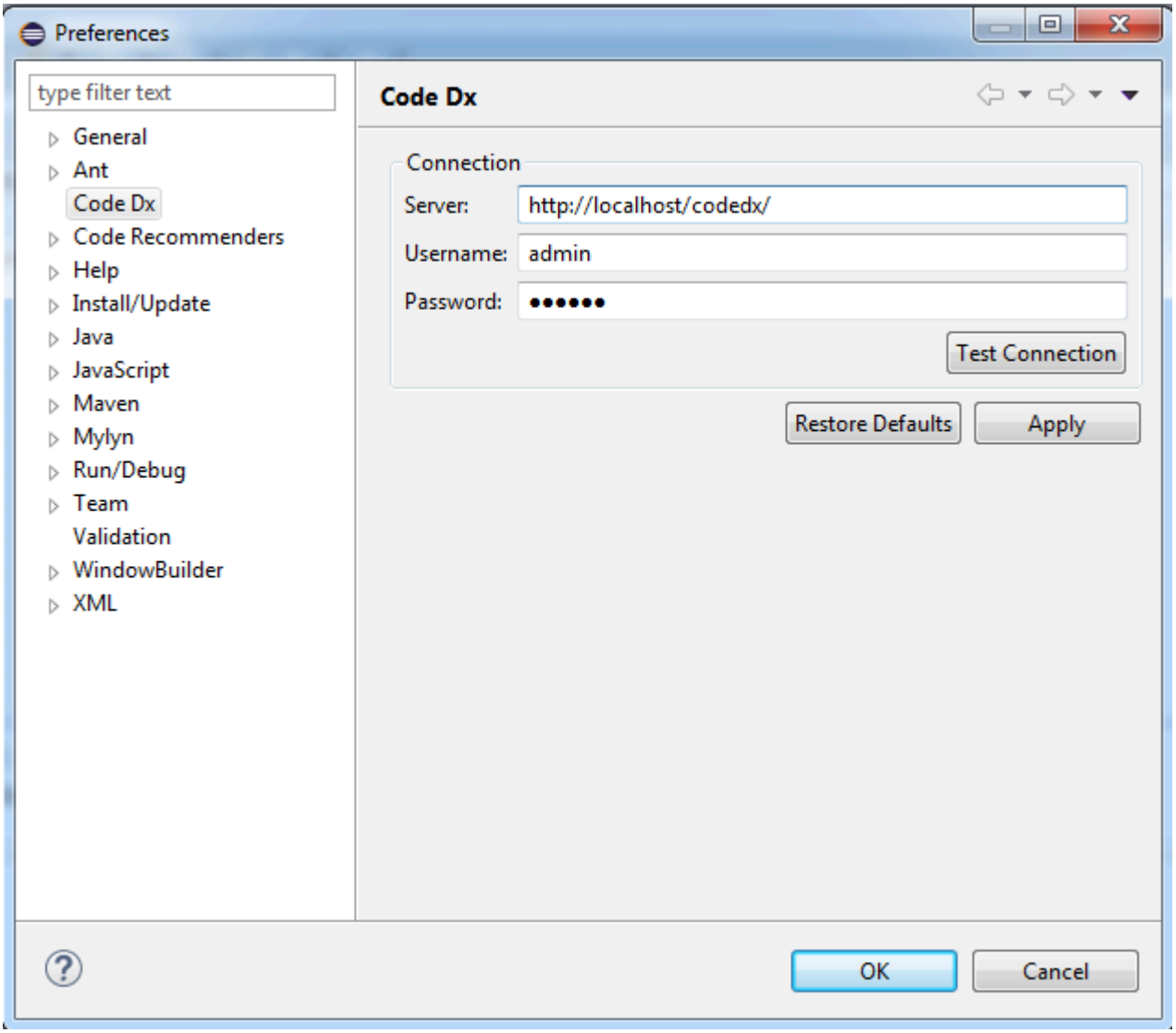
Select the *Code Dx* plugin and continue with the wizard in order to complete the plugin installation process.

Configuration

To configure the Eclipse plugin, you can either navigate to *Window -> Preferences* and select *Code Dx*, or you can navigate to *Code Dx -> Configure*. These methods will result in the same preferences section being shown.

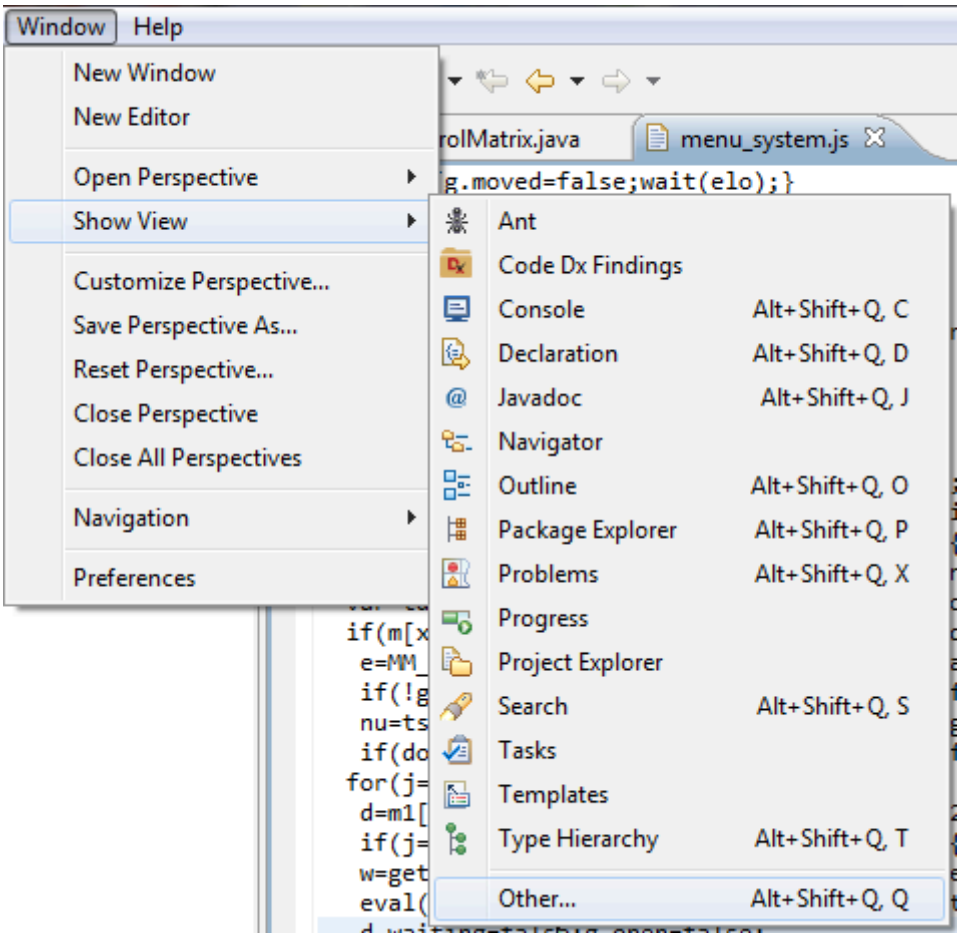


To configure the Eclipse plugin, provide the *Server*, *Username*, and *Password*. Enter the URL of the Software Risk Manager server in the *Server* field. Enter your Software Risk Manager log in credentials in the *Username* and *Password* fields. Verify the Software Risk Manager plugin can communicate with the Software Risk Manager server by clicking the *Test Connection* button. A message will appear indicating a successful connection or explaining why the connection failed.

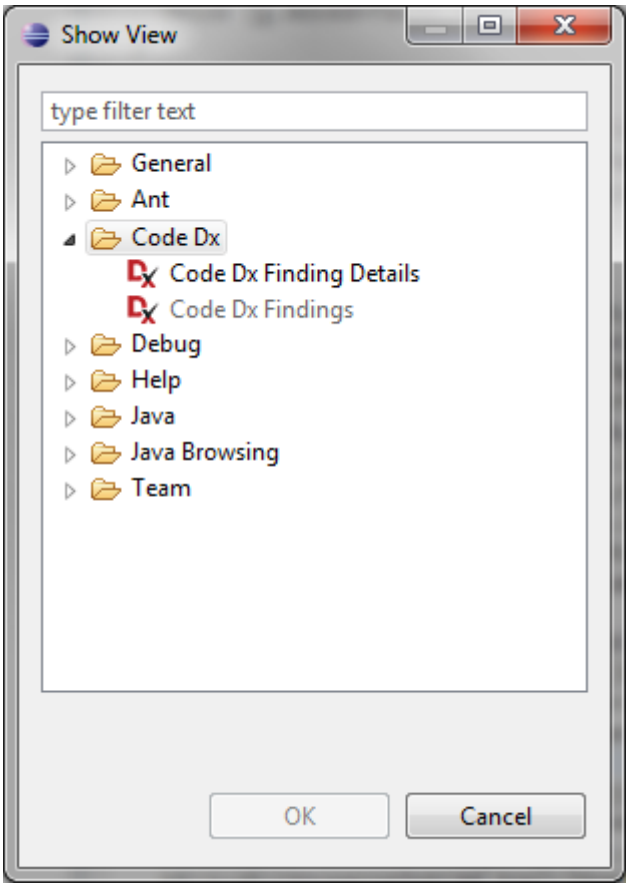


Views

The Eclipse plugin provides two main views: the *Code Dx Findings View* and the *Code Dx Finding Details View*. Both of these are available by navigating to *Window -> Show View -> Other*.

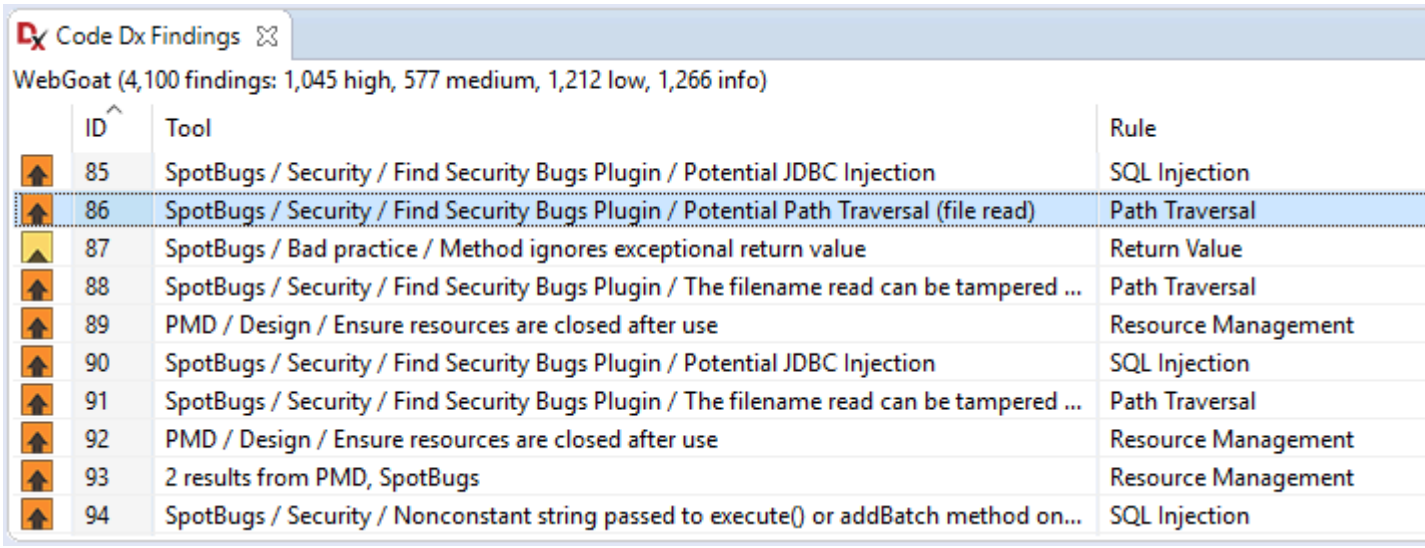


The views will be located under the *Code Dx* folder.



Findings View

This view displays the findings from the selected project. There are three sections: the findings table, the summary area, and a toolbar.



ID	Tool	Rule
85	SpotBugs / Security / Find Security Bugs Plugin / Potential JDBC Injection	SQL Injection
86	SpotBugs / Security / Find Security Bugs Plugin / Potential Path Traversal (file read)	Path Traversal
87	SpotBugs / Bad practice / Method ignores exceptional return value	Return Value
88	SpotBugs / Security / Find Security Bugs Plugin / The filename read can be tampered ...	Path Traversal
89	PMD / Design / Ensure resources are closed after use	Resource Management
90	SpotBugs / Security / Find Security Bugs Plugin / Potential JDBC Injection	SQL Injection
91	SpotBugs / Security / Find Security Bugs Plugin / The filename read can be tampered ...	Path Traversal
92	PMD / Design / Ensure resources are closed after use	Resource Management
93	2 results from PMD, SpotBugs	Resource Management
94	SpotBugs / Security / Nonconstant string passed to execute() or addBatch method on...	SQL Injection

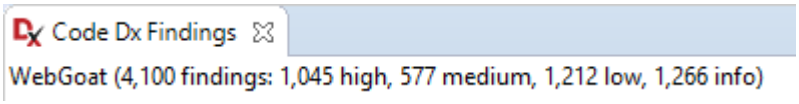
The table has columns for many of the finding properties such as:

- Severity (shown using severity icons)

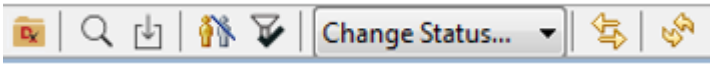
- ID
- Tool
- Rule
- CWE
- Codebase Locations
- Status

The table can be sorted by any of these columns except the Tool column. Right-clicking on a row provides a context menu for performing actions on the selected finding.

The summary area is located above the table. It provides the project name, the total number of findings, and the number of findings for each severity category.

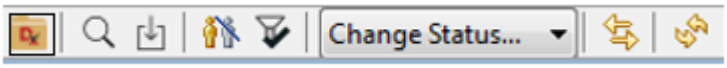


The toolbar buttons include (from left to right) the ability to switch Code Dx projects, show details of a finding, show remote source code residing on the Code Dx server, show only findings assigned to you, filter by status, change status, synchronize the Eclipse Package Explorer view with the table contents, and refresh the table.

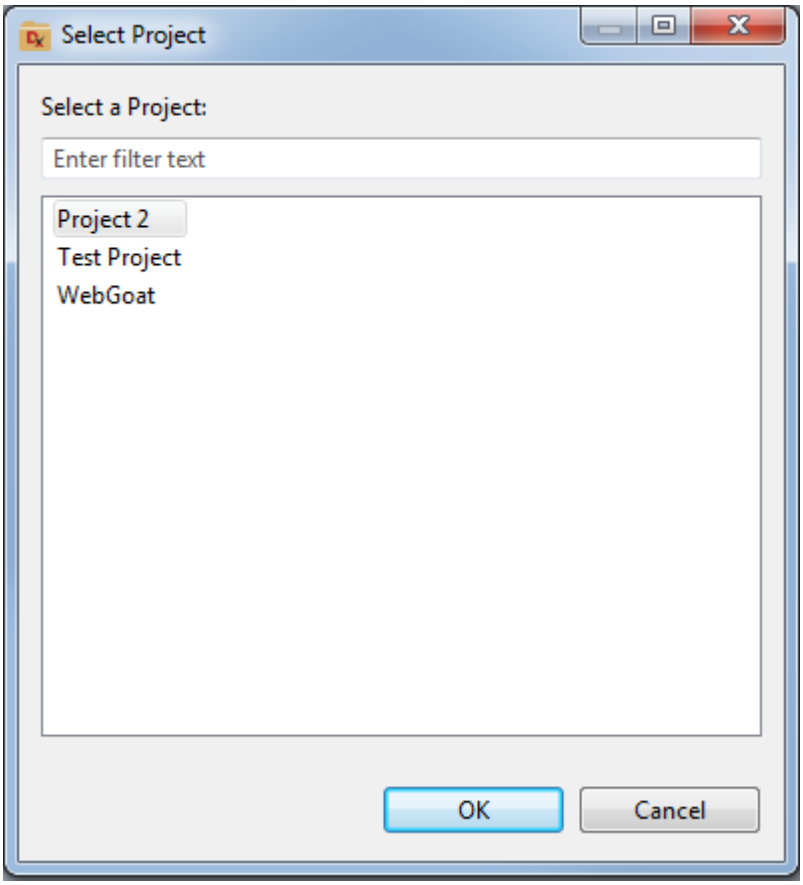


Switching Code Dx Projects

The table of findings can easily be changed to show the results of different Code Dx projects. This is done using the following toolbar button:

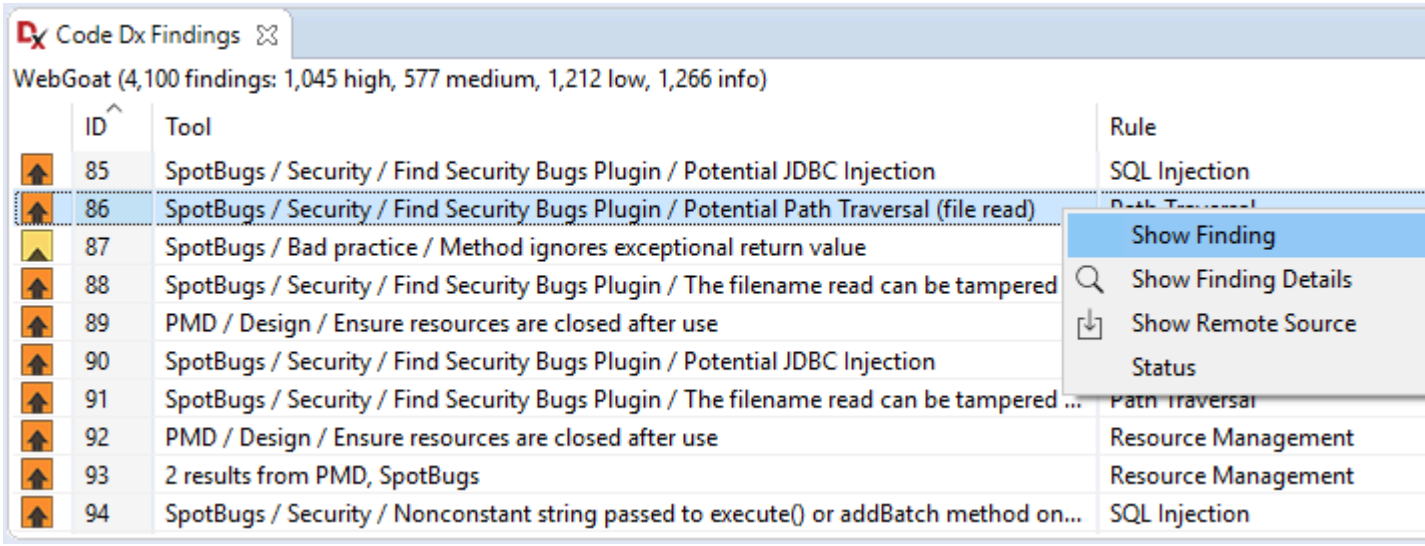


A dialog is displayed after you click the *Select a Project* button. Note that only the projects accessible to your user credentials (provided during configuration) will be shown in this dialog.



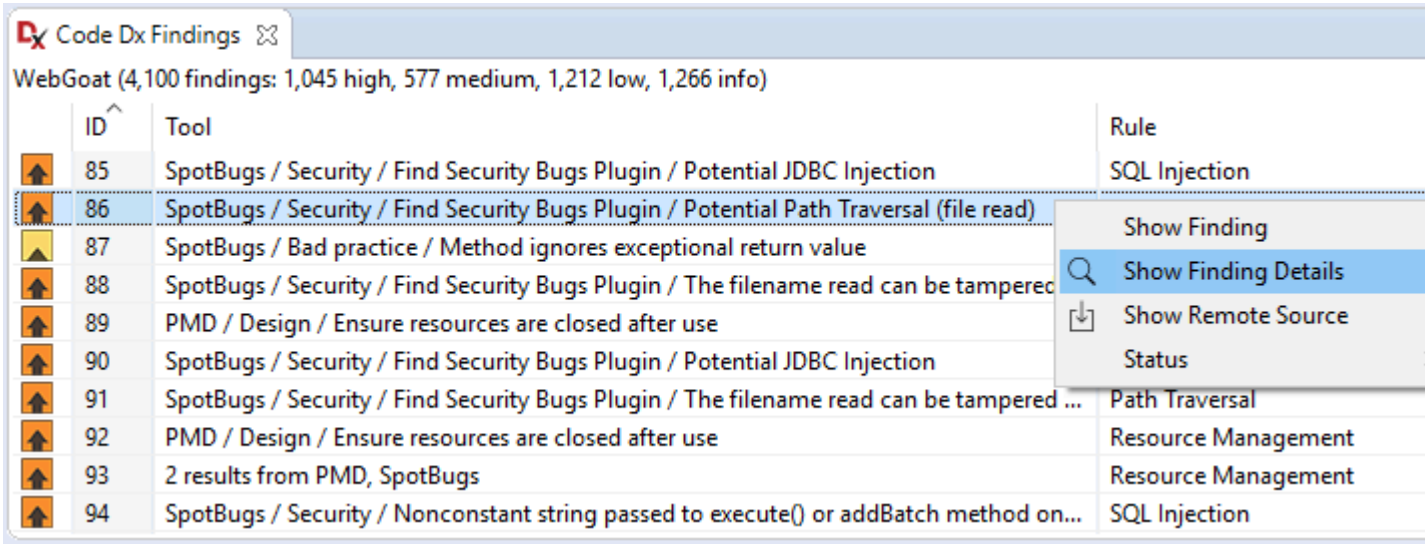
Showing Source Code for a Finding

You can view and edit the local source code for a finding by double-clicking on it, or by right-clicking on the selected finding and choosing *Show Finding* in the context menu. The editor view will open with the associated source code. If there is a line number for the finding, the editor will automatically scroll to that location.



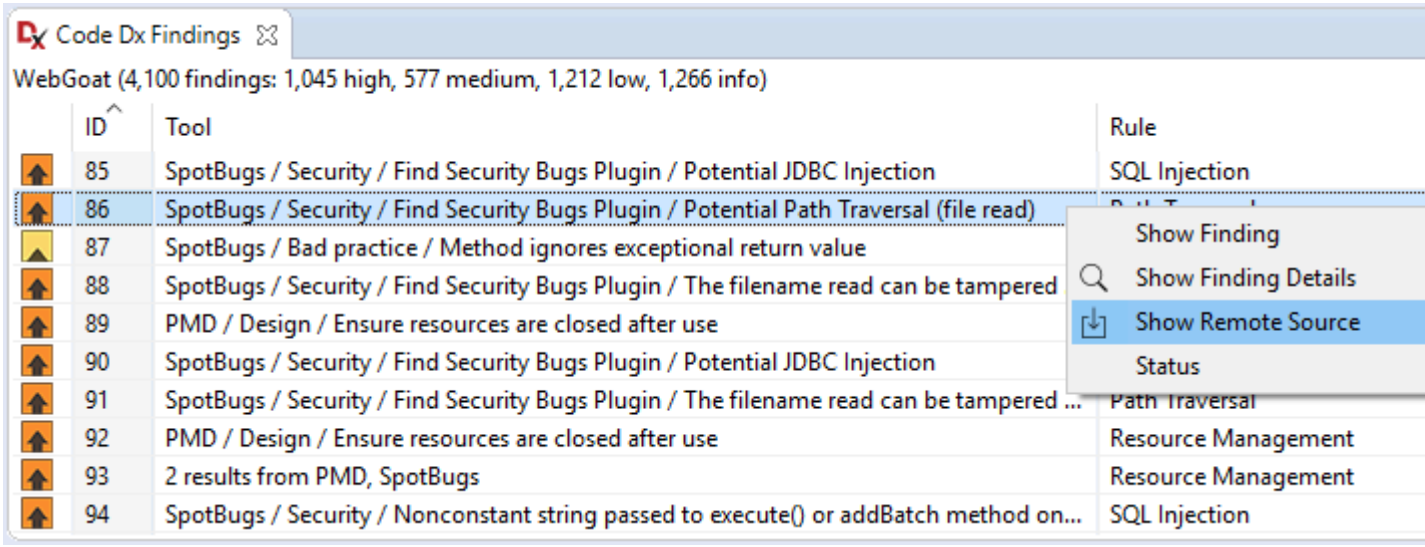
Showing Finding Details

The details of a finding can be viewed by selecting a row in the table and either using the toolbar button or the context menu of the row.

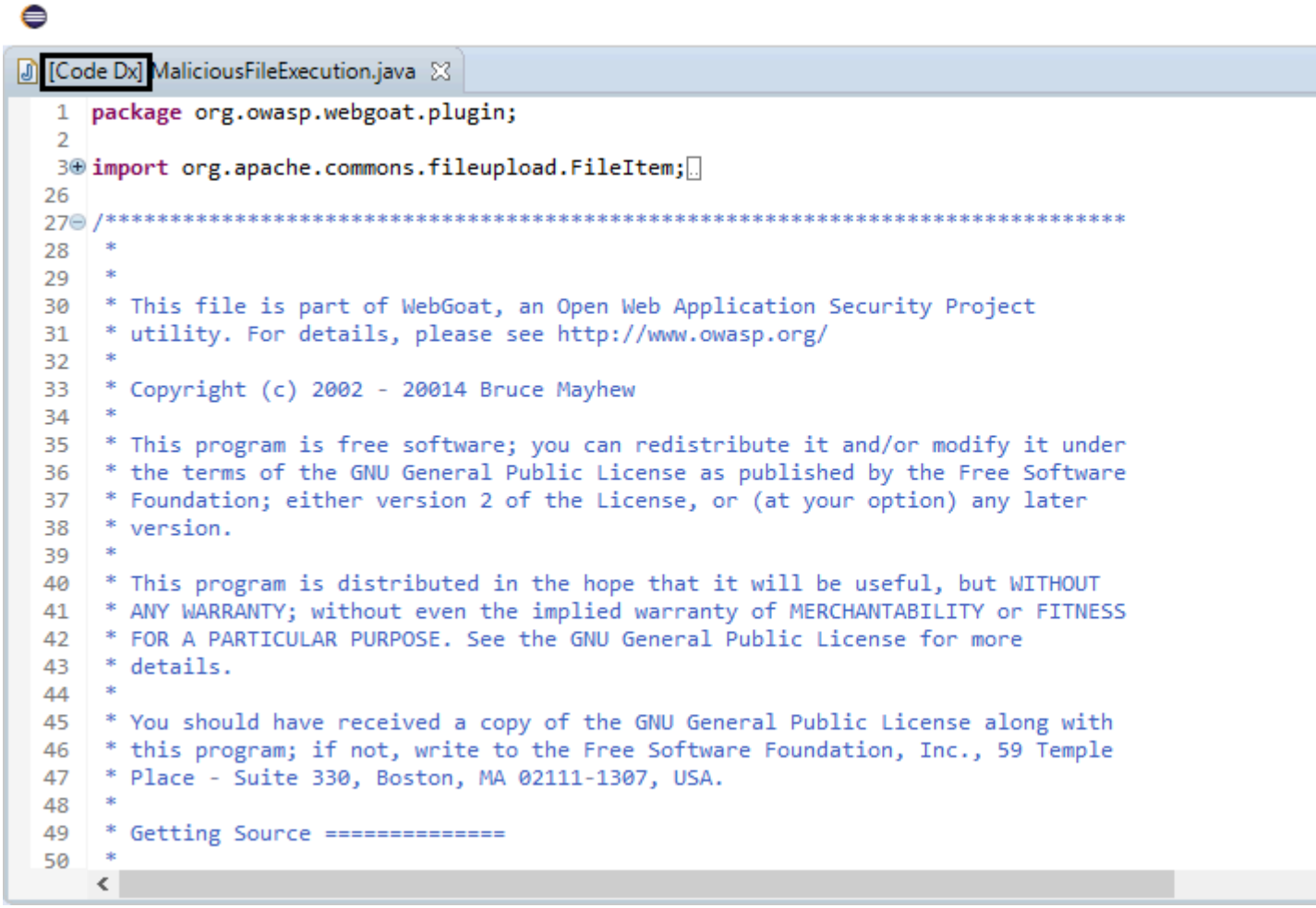


Showing Remote Source

It is sometimes useful to see the current version of a source file on the Code Dx server. For example, if the code local to eclipse is different from the remote code. This can be done by selecting a row in the table and either using the toolbar button or the context menu of the row.



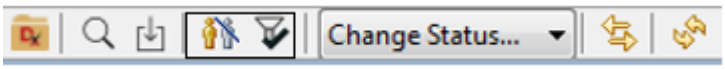
The file will be downloaded from the Code Dx server and displayed in a read-only editor.



The title of the editor will be marked with a `[Code Dx]` label before the filename so that it can be distinguished from local source.

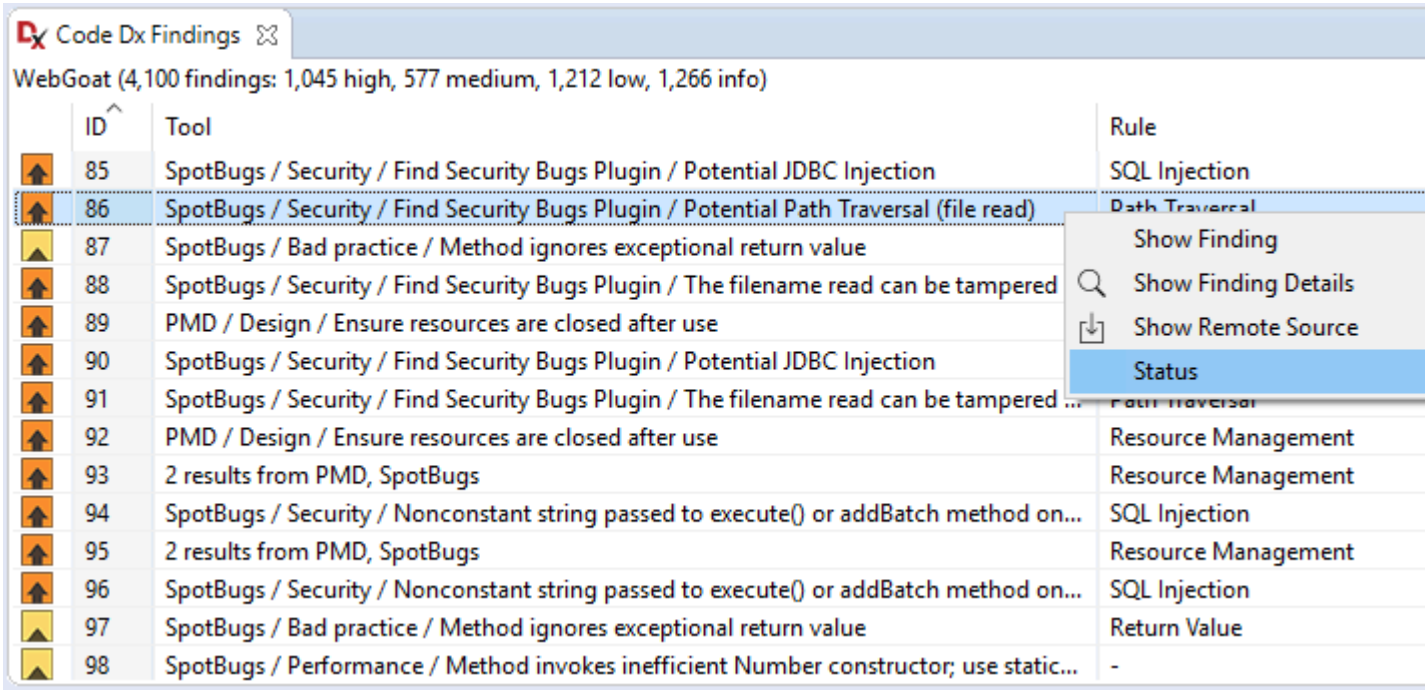
Filtering Findings

The table can be filtered to show only the findings that are assigned to you (left button) and to hide findings that have a status of `Ignored`, `False Positive`,`Fixed`, `Gone` and `Mitigated` (right button).



Changing the Status of Findings

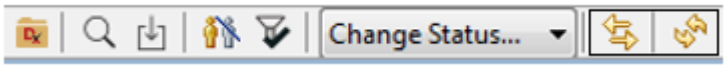
You can change the status of a single finding or a group of findings. First select the finding(s) then use either the *Change Status* dropdown located on the toolbar or the *Status* option in the context menu.



Sync and Refresh

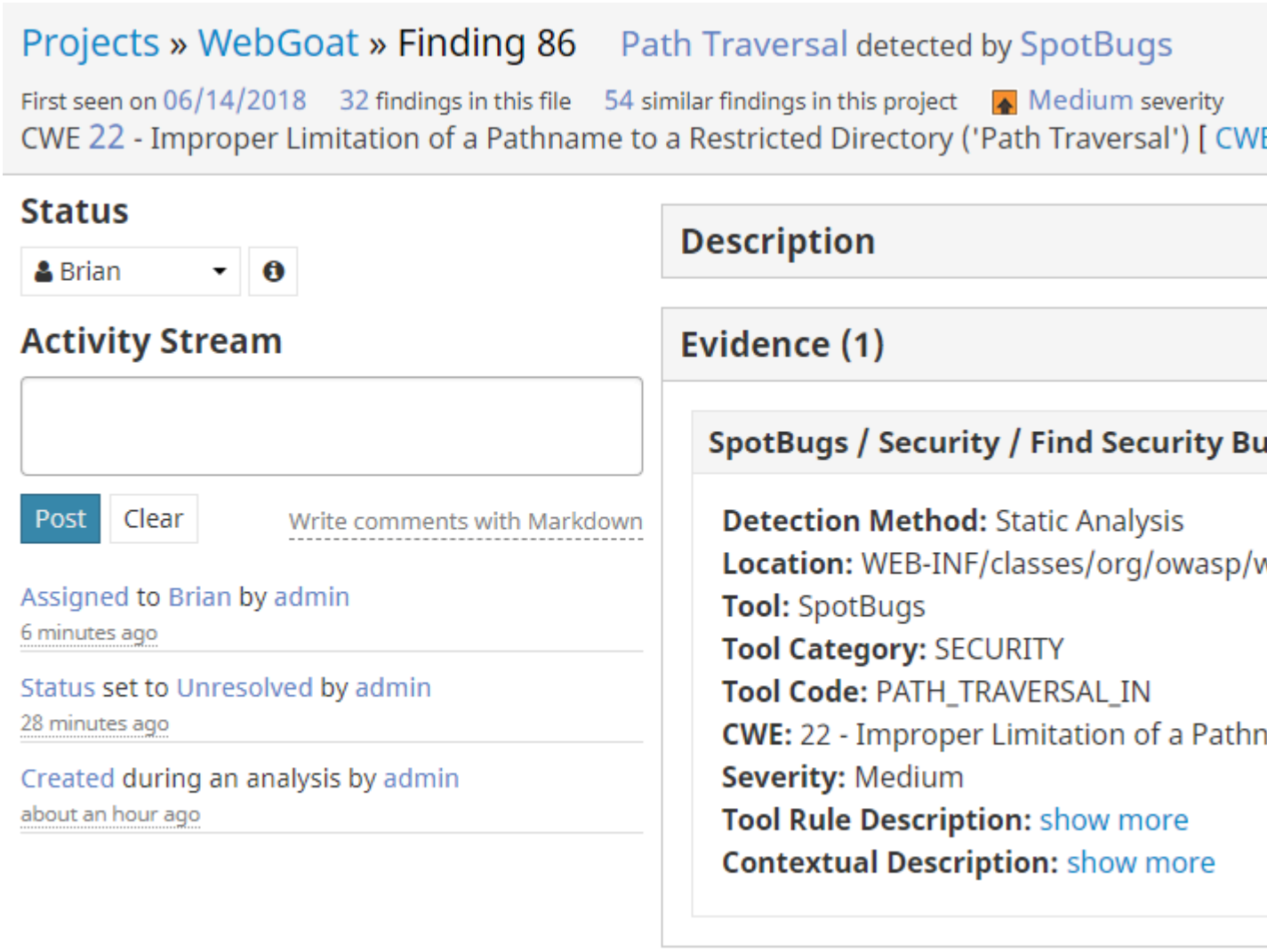
The *Sync* button is situated to the immediate right of the *Change Status* dropdown and is used in conjunction with the Package Explorer view and editors. With the button enabled, just click a file in the Package Explorer and the table will display only those findings associated with the selected file. Selecting an editor window of an open file will do the same.

The *Refresh* button is located to the right of the *Sync* button. It is used to refresh the table with the findings on the Code Dx server.

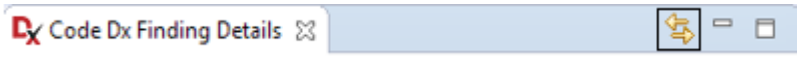


Finding Details View

The Finding Details view shows a minimal version of the Finding Details page in the Code Dx application. If you have the `update` role for a project, you can change the status of a finding in that project and post to its activity stream.

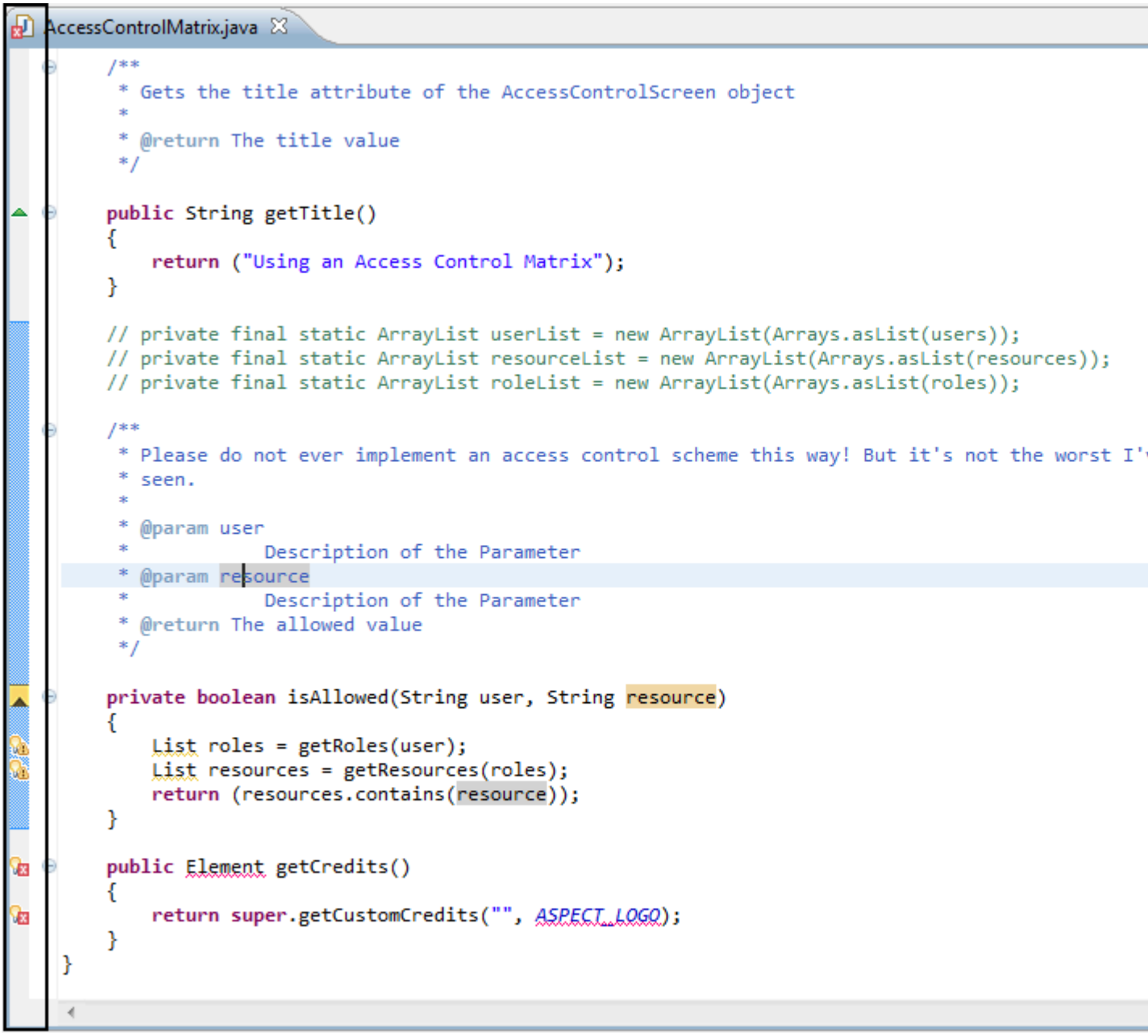


The toolbar contains a *Sync* button. When enabled, selections in the Findings table will automatically update the information in the Finding Details view.



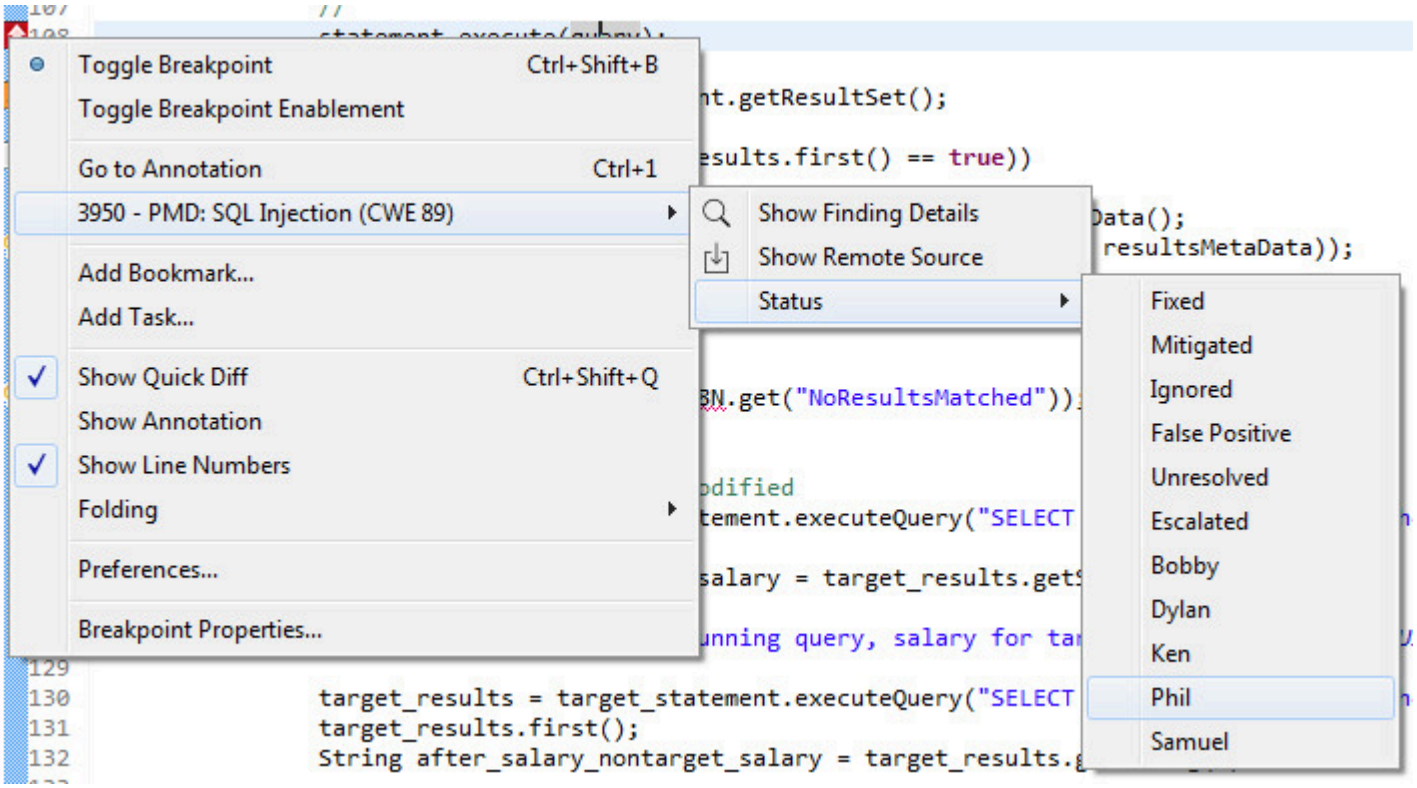
Markers

The Software Risk Manager Eclipse plugin automatically adds source code markers to help you determine the location of the findings within the code.



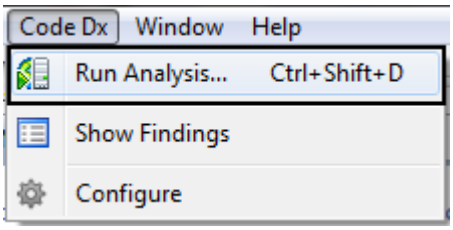
There are markers in both the left and right gutters. The markers in the left gutter use the Software Risk Manager severity icons to show the highest level severity on the given line. The markers in the right gutter show the findings throughout the entire file.

The context menu on the marker will show a submenu for each finding on the given line.

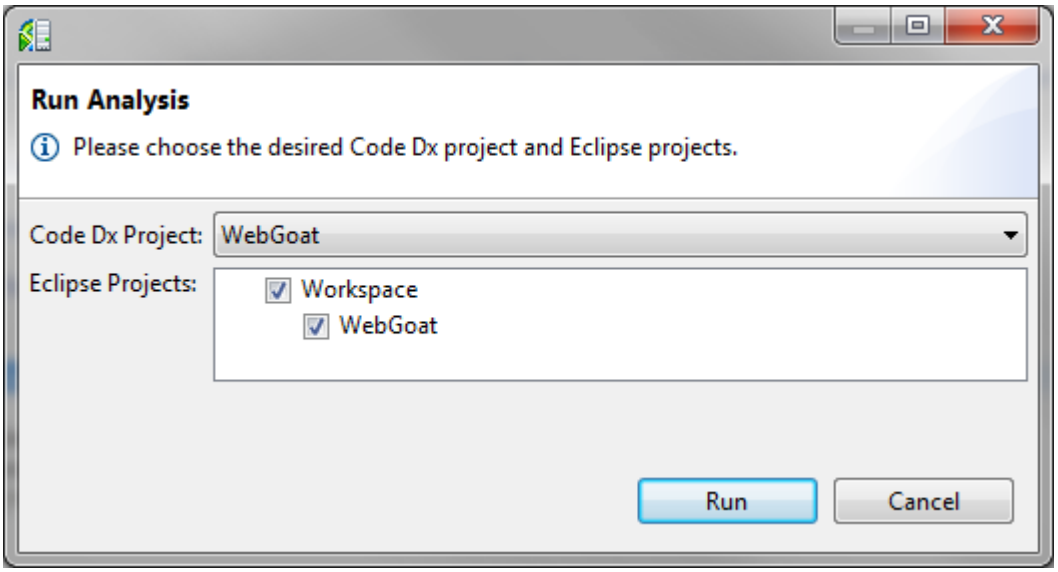


Running an Analysis

If you have the `create` role on a Software Risk Manager project, you have the ability to perform Software Risk Manager analyses on that project from within the Eclipse IDE. Just select the *Run Analysis* option from the *Code Dx* menu.



When the dialog is displayed, select the Software Risk Manager project from the dropdown, the Eclipse projects from the list, and click *Run*.



An Eclipse job is created and the progress is displayed in the bottom right corner of the IDE. This will create a zip file containing all local source code from the configured project and send it to be analyzed by Software Risk Manager. The Findings table will automatically be updated upon completion of the analysis.

Visual Studio

The Code Sight plugin streamlines the use of Software Risk Manager for developers by allowing them to push out new builds for analysis and view analysis results from within the Visual Studio IDE.

A Software Risk Manager project is required. To allow users access to the project, they must be assigned to the project where the user roles are consistent with those within Software Risk Manager.

For information on how to install and use the Code Sight plugin, please refer to the Code Sight documentation.

Visual Studio Code

The Code Sight plugin streamlines the use of Software Risk Manager for developers by allowing them to view analysis results from within the Visual Studio Code IDE.

A Software Risk Manager project is required. To allow users access to the project, they must be assigned to the project where the user roles are consistent with those within Software Risk Manager.

For information on how to install and use the Code Sight plugin, please refer to the Code Sight documentation.

IntelliJ

The Code Sight plugin streamlines the use of Software Risk Manager for developers by allowing them to view analysis results from within the IntelliJ IDE.

A Software Risk Manager project is required. To allow users access to the project, they must be assigned to the project where the user roles are consistent with those within Software Risk Manager.

For information on how to install and use the Code Sight plugin, please refer to the Code Sight documentation.

Burp Suite

The Software Risk Manager Burp Suite plugin provides a way to upload Burp Suite findings to your Software Risk Manager server from within Burp Suite.

A Software Risk Manager project and an API key are required. The API key must have the `create` role on the project it needs to interact with.

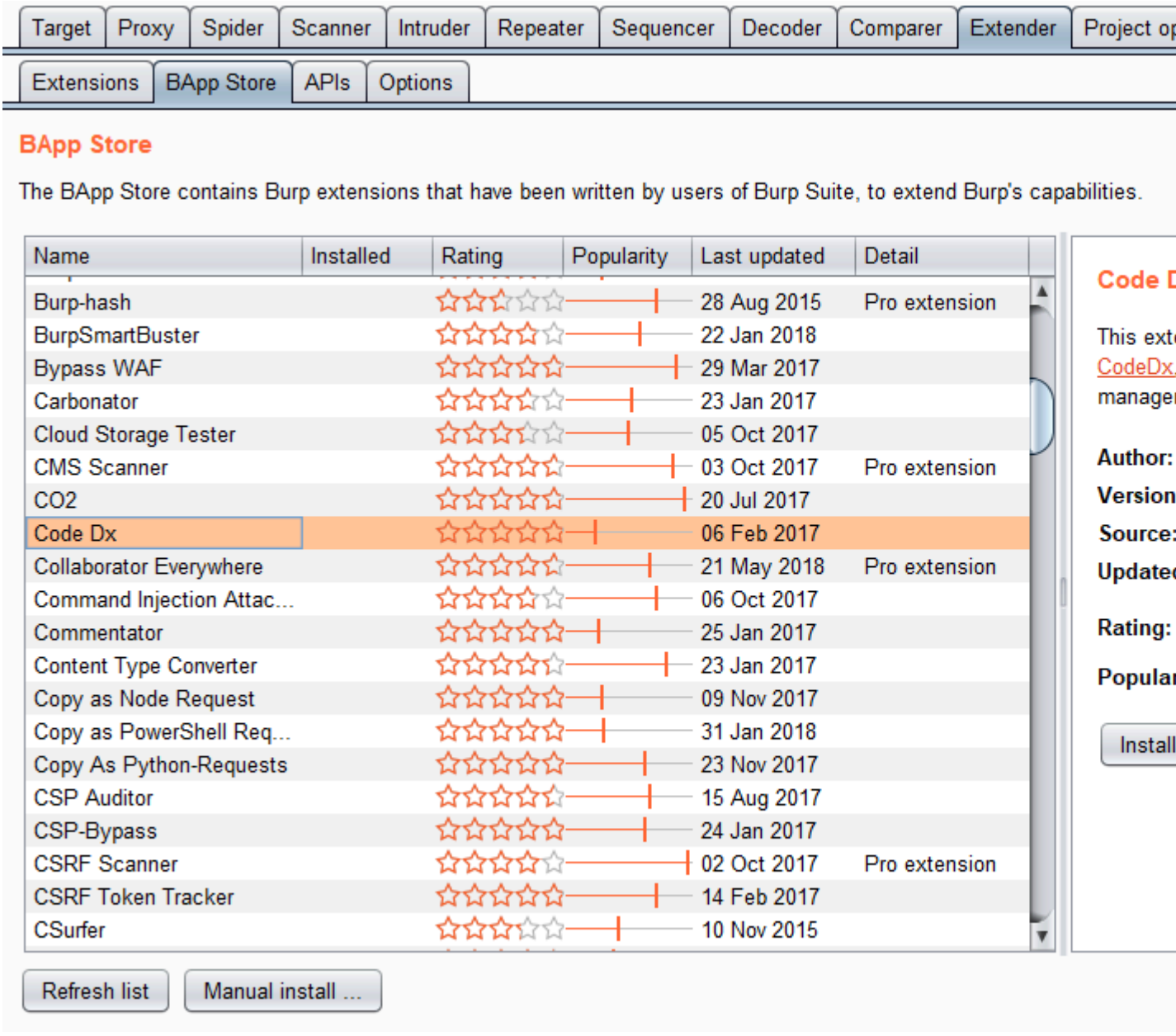
This section of the Plugins Guide explains how to install and use the Burp Suite plugin. For more information, you may visit our GitHub Repository. This plugin is open source and we welcome community involvement.

Installation

The Software Risk Manager Burp Suite plugin is available for download from the BApp Store and from our GitHub Repository.

BApp Store

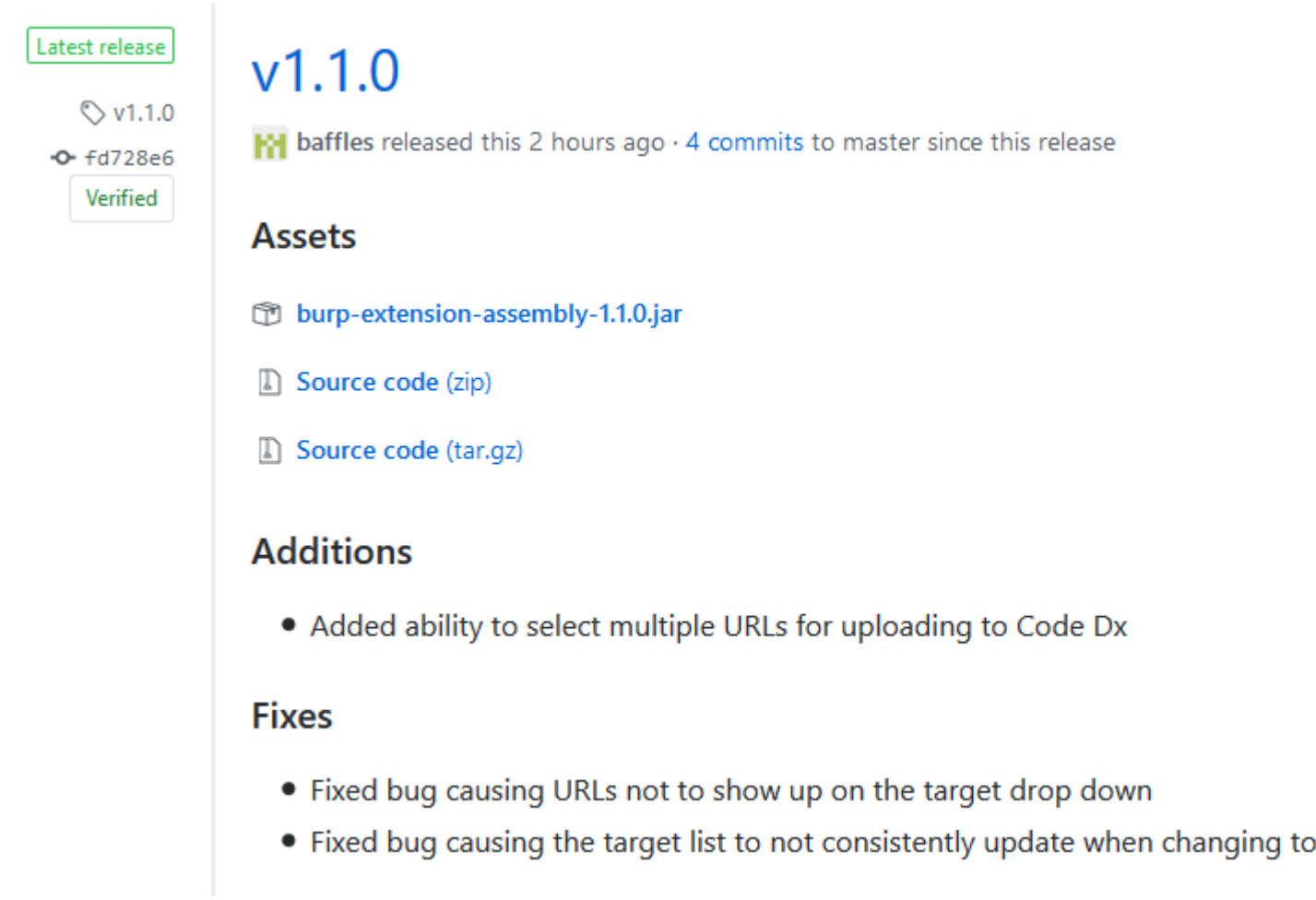
To install the Software Risk Manager Burp Suite plugin from the BApp Store, go to the Extender tab in Burp Suite, click the BApp Store tab, and click on *Code Dx*.



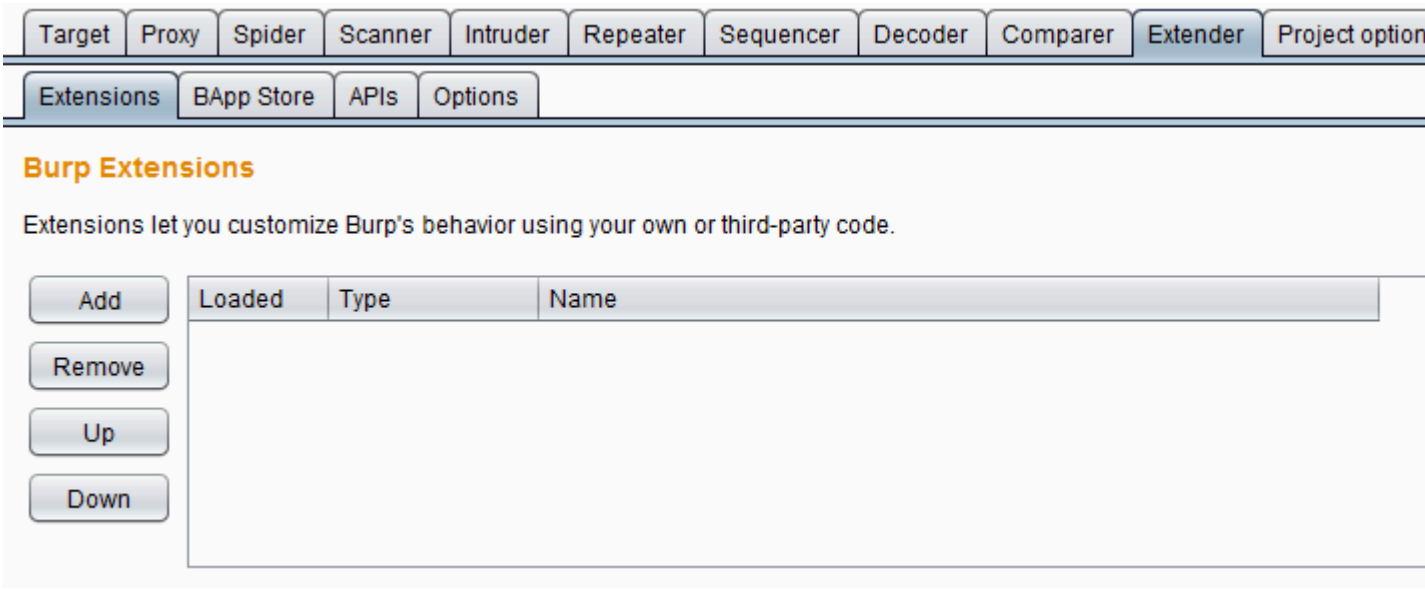
On the right panel, click the *Install* button.

GitHub Repository

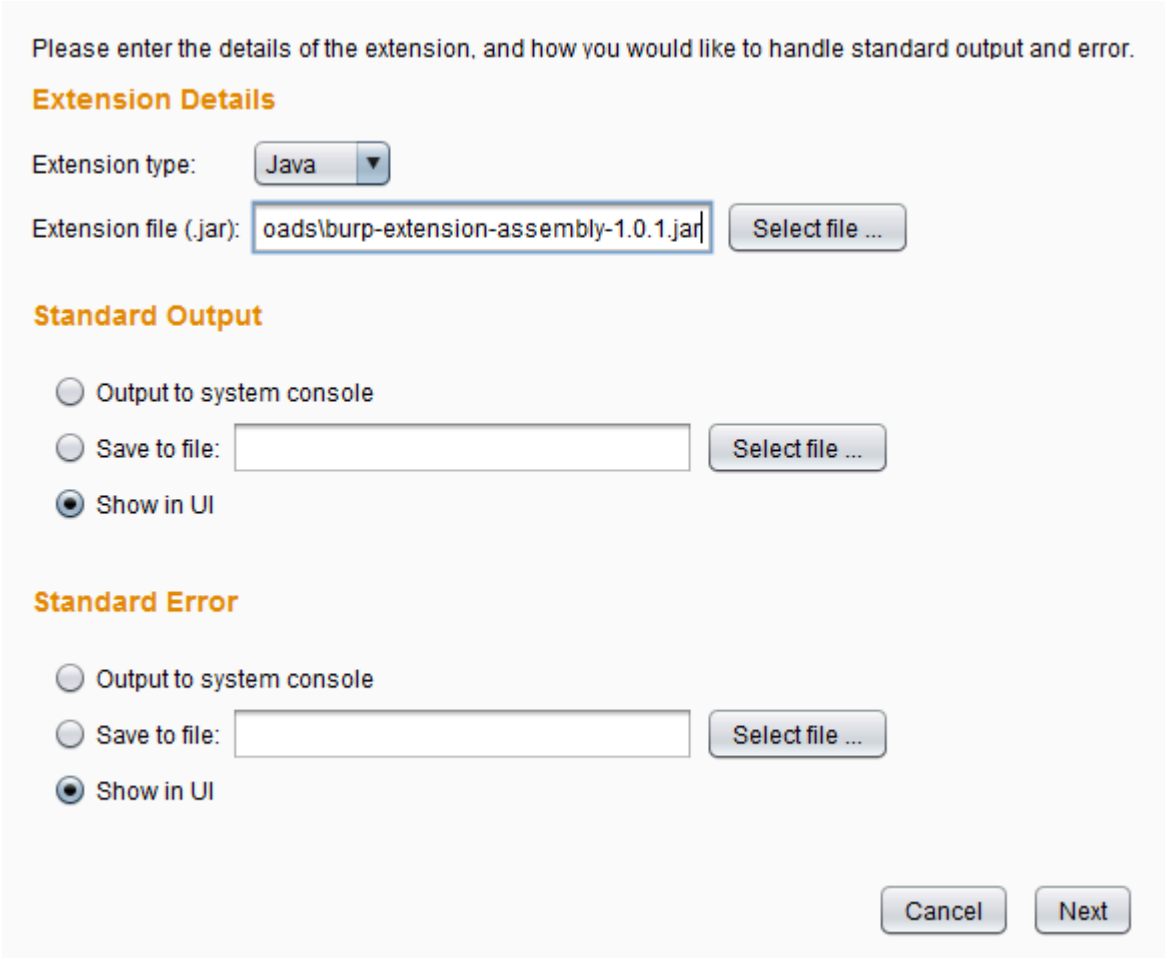
The Burp Suite plugin can be found on our GitHub Repository.



To install the extension, go to the Extender tab in Burp Suite and click *Add* in the Burp Suite Extensions section.



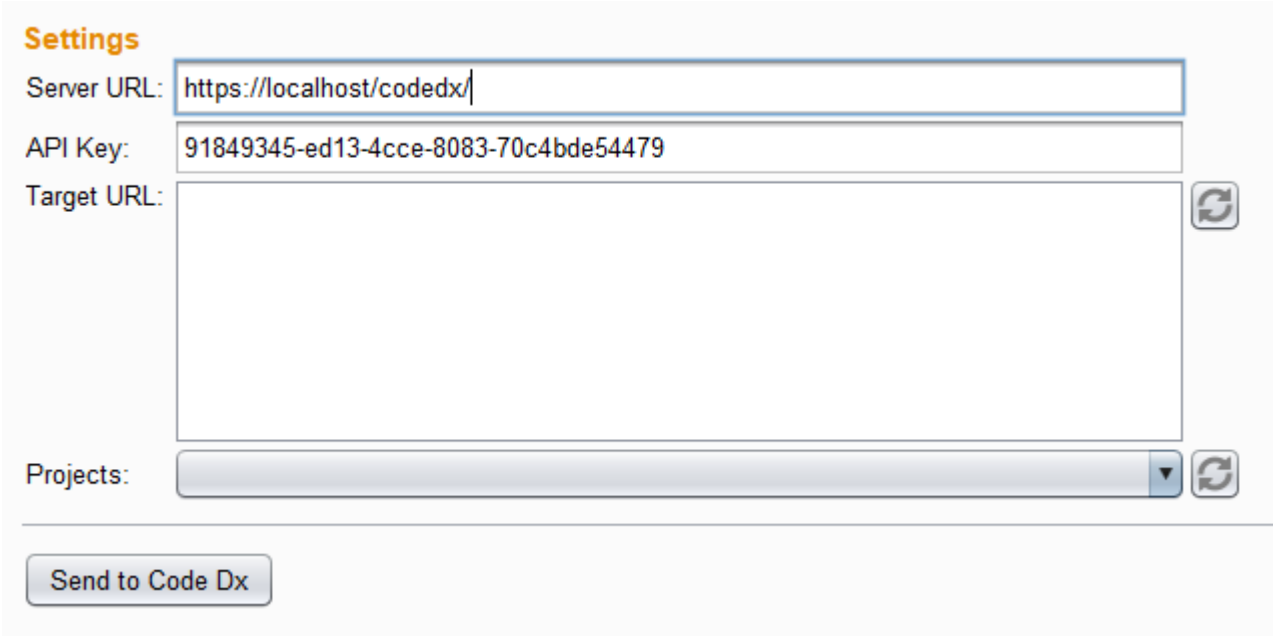
Click *Select file* for the Extension file field and navigate to the burp-extension-assembly jar, then click *Next* to load the extension.



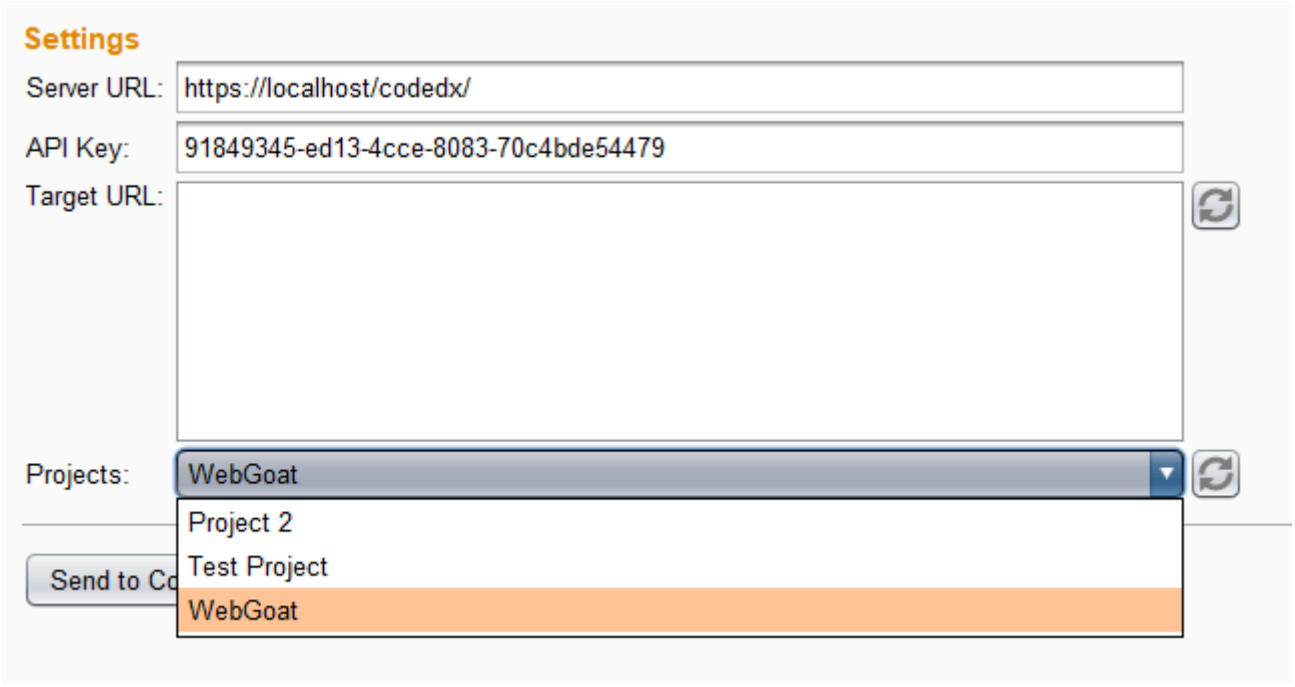
Configuration

To configure the Burp Suite plugin, navigate to the Code Dx tab.

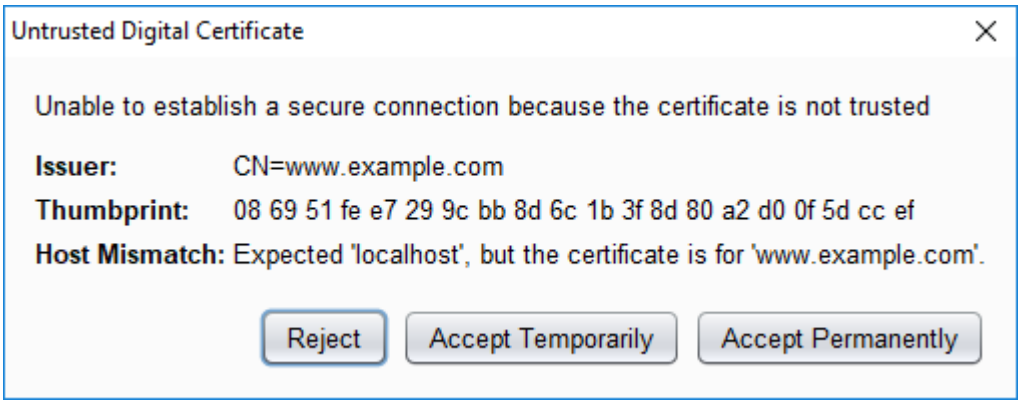
The *Server URL* and *API Key* are required fields for sending data to Software Risk Manager. Ask your Software Risk Manager administrator to generate the server API key with the `create` role for the project(s) which the plugin must interact with.



Once the *Server URL* and *API Key* fields are populated, click the *Refresh* button to list the projects available to the API key in the *Project* dropdown. It is highly recommended that you specify an HTTPS URL, since using HTTP is insecure.



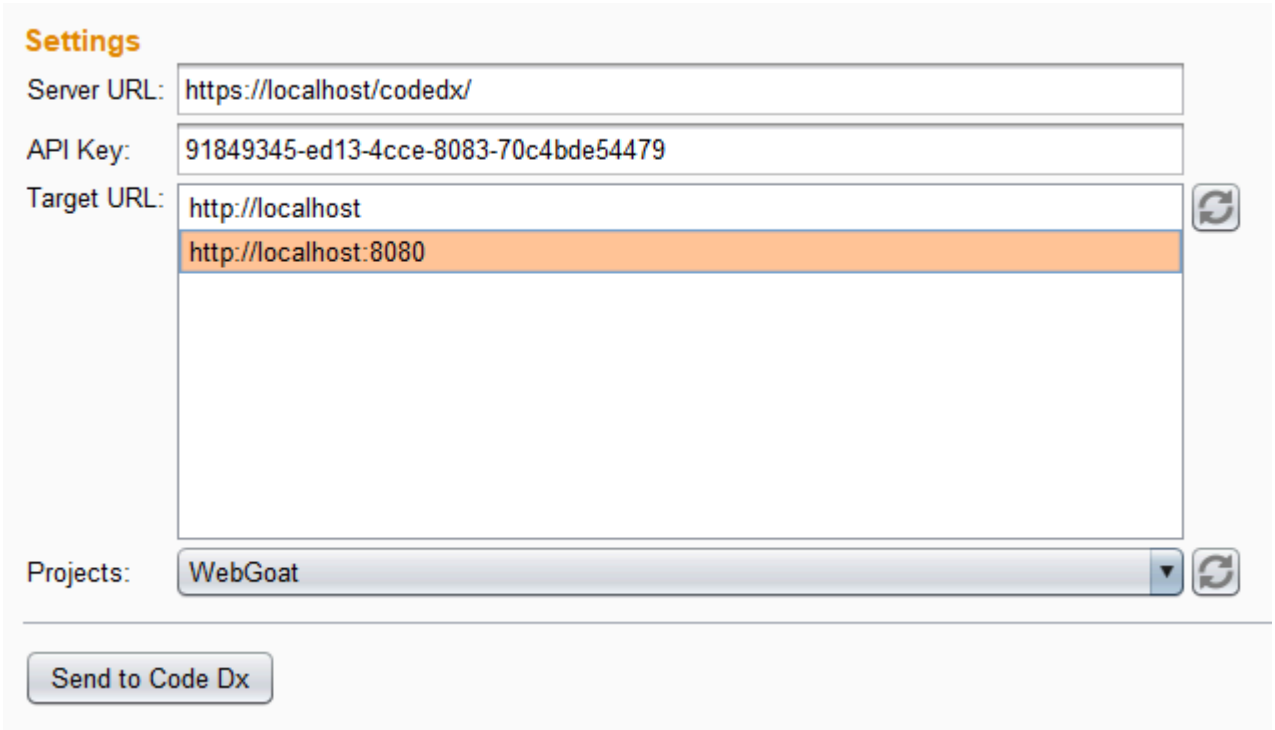
If you receive a warning regarding an invalid certificate, you will be prompted to *Reject*, *Accept Temporarily*, or *Accept Permanently*. Accepting temporarily will remember the exception until the session ends. Accepting permanently will create a .usertrust directory containing the truststore information. On Windows this will be in your appdata directory, on Mac it will be in the Application Support folder, and on Linux it will be in the .codedx folder in the home directory.



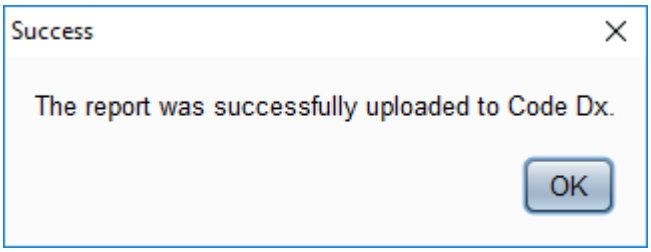
Sending Results to Software Risk Manager

After scanning with Burp Suite, there are two ways you can send the results to Software Risk Manager. The first is to choose a *Target URL* from the list in the Software Risk Manager Settings in Burp Suite. After performing a scan, click the *refresh* button to list all of the available targets. Multiple targets can be selected from the list using *Ctrl + Click*.

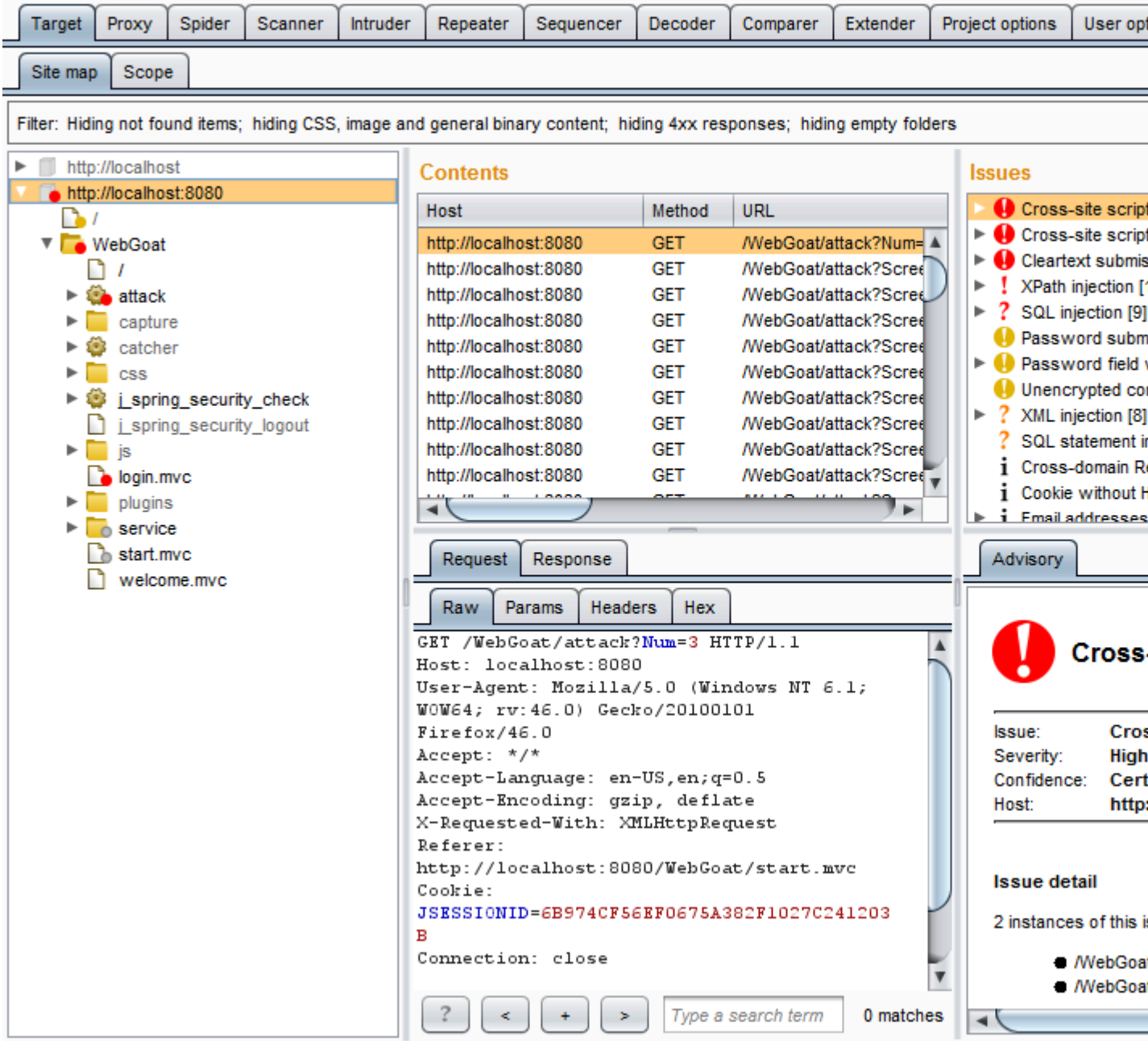
Select the project you would like to use, then click the *Send to Code Dx* button to send the results.



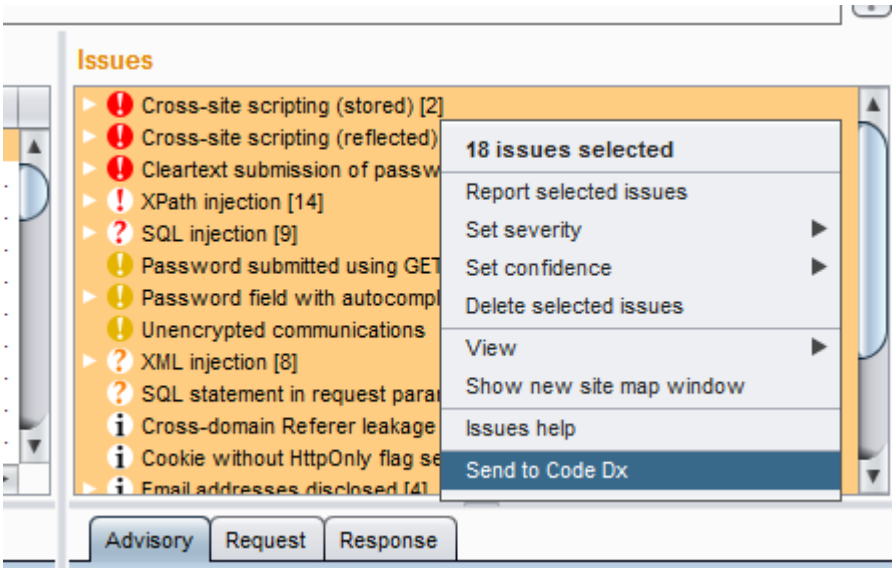
You will receive a message indicating whether or not the action was successful.



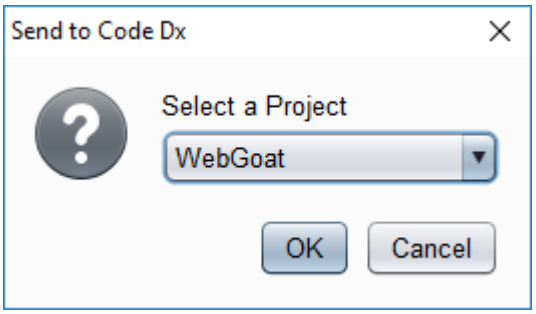
The second method to send the results is to use the context menu in the Issues panel of the Target view. To do this, open the Target view and select your target or targets from the Site map.



Select the issues that you want to analyze and right click in the Issues panel. Click the *Send to Code Dx* button at the bottom of the context menu.



A menu will pop up and will ask you to select the Software Risk Manager project. Note that this option is independent of the project and target settings from the Software Risk Manager view.



As with the previous method, you will receive a message indicating whether or not the action was successful.

OWASP ZAP

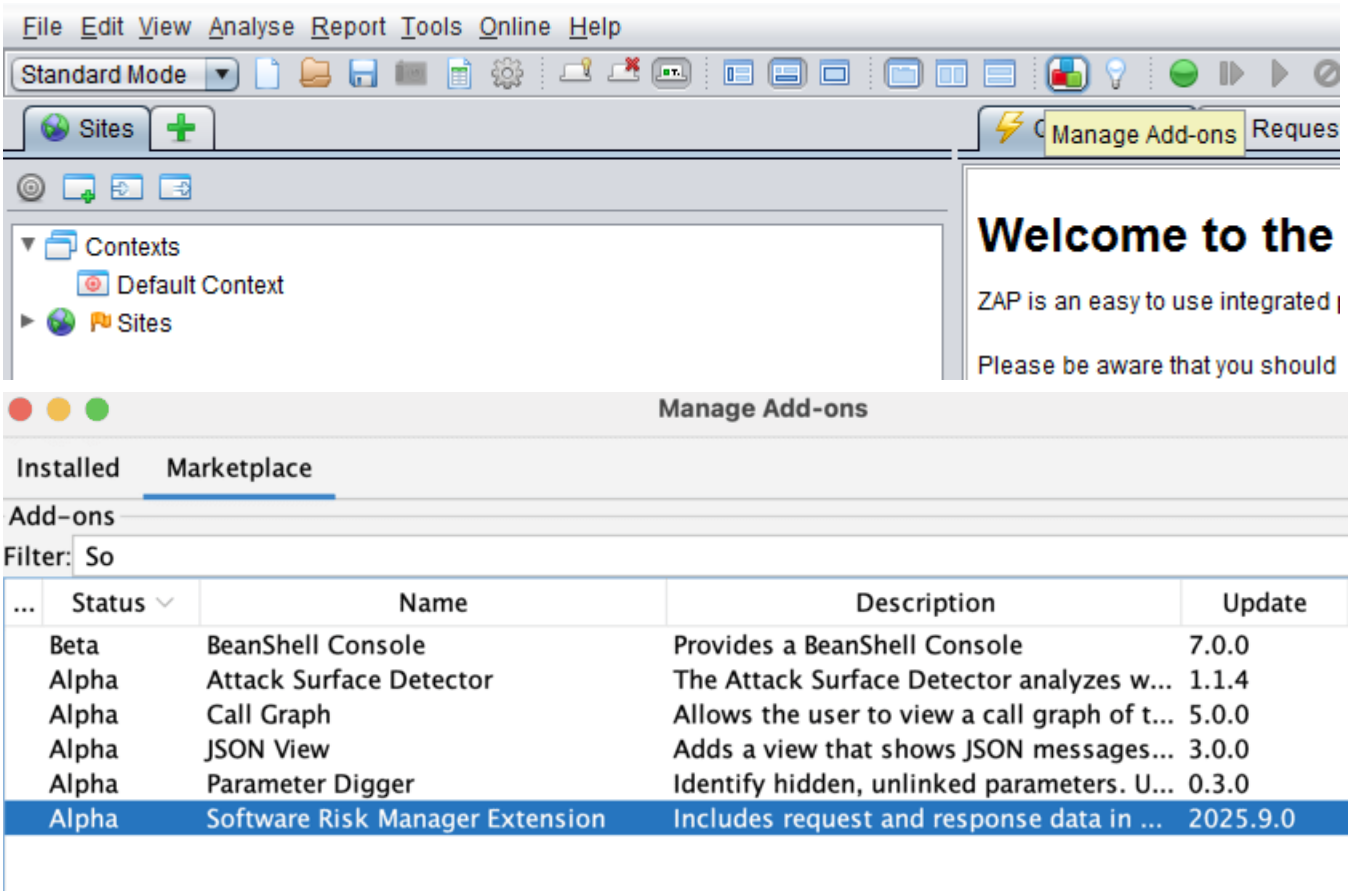
The Software Risk Manager OWASP ZAP plugin provides a way to upload OWASP ZAP alerts to your Software Risk Manager server from within OWASP ZAP.

A Software Risk Manager project and an API key are required. The API key must have the `create` role for the project.

This section of the Plugins Guide explains how to install and use the OWASP ZAP plugin. For more information, you may visit our GitHub Repository. This plugin is open source and we welcome community involvement.

Installation

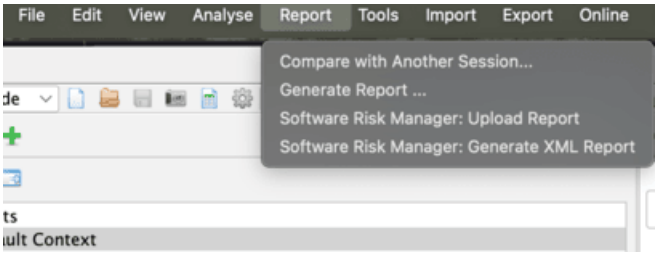
The Software Risk Manager OWASP ZAP extension is available for installation through the OWASP ZAP Marketplace.



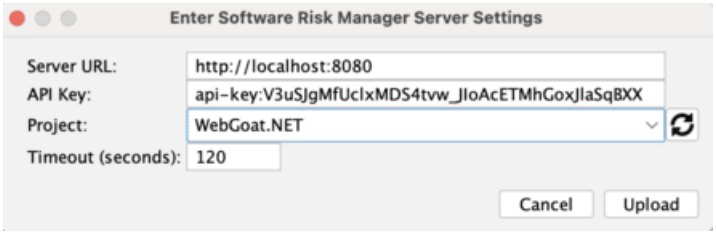
Sending Results to Software Risk Manager

The OWASP ZAP plugin can generate a compatible XML file which can be uploaded manually, or it can upload a report directly to Software Risk Manager.

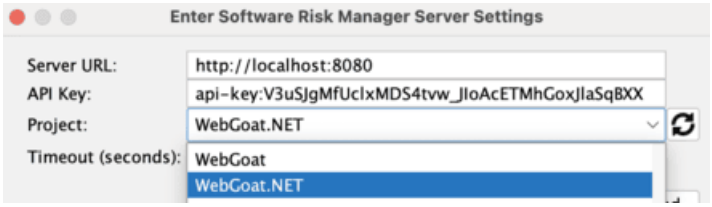
To upload a report to Software Risk Manager, select the *Software Risk Manager: Upload Report* option from the *Report* menu.



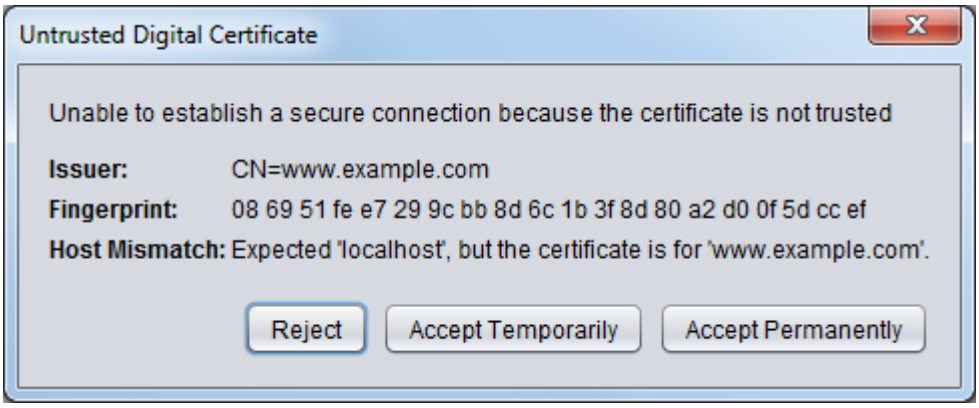
You will be prompted for the *Server URL*, *API Key* and *Project*. Your settings will be remembered between sessions and are stored in the srm.properties file located in the OWASP ZAP folder in your user directory.



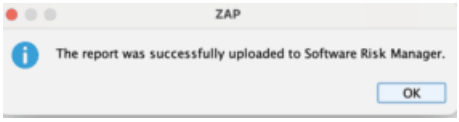
After entering the *Server URL* and *API Key*, click the *Refresh* button to populate the *Project* dropdown.



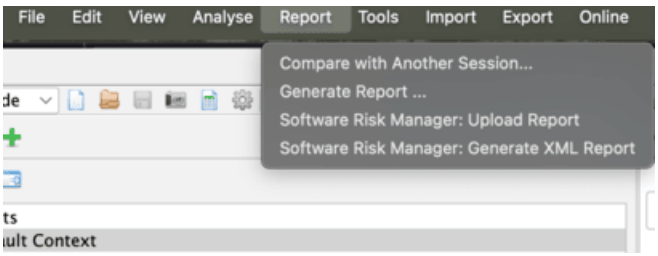
If you receive a warning regarding an invalid certificate, you will be prompted to *Reject*, *Accept Temporarily*, or *Accept Permanently*. Accepting temporarily will remember the exception until the session ends. Accepting permanently will create a .usertrust directory containing the truststore information. On Windows this will be in your appdata directory, on Mac it will be in the Application Support folder, and on Linux it will be in the home directory.



You will receive a message indicating whether or not the action was successful.



You can generate an XML file for use with Software Risk Manager by selecting the *Software Risk Manager: Generate XML Report* option from the *Report* menu.



API

The plugin also provides an API to use its functions programmatically. More information on how to use the ZAP API can be found on the ZAP GitHub Wiki.

Note that as a security measure, ZAP will not include messages with Exceptions by default. If you want to enable messages, you can check *Report error details via API* in *Tools -> Options -> API*.

Actions

uploadReport

Uploads a report to Software Risk Manager. Note that uploading an empty report with no alerts will cause an Exception to be thrown as Software Risk Manager won't be able to read it and will return a non-200 response.

Parameters

- filePath: Absolute path to the report file
- serverUrl: Software Risk Manager server URL
- codeDxApiKey: Software Risk Manager API Key
- projectId: Software Risk Manager Project ID
- fingerprint: Optional SHA1 hash of an invalid certificate to make an exception for
- acceptPermanently: Optional boolean for if the exception should be stored permanently in a truststore file.

Returns

`OK` if the report is uploaded successfully.

generateAndUpload

Generates a Software Risk Manager report, saves it to a temporary file, uploads to Software Risk Manager, then deletes the file.

Parameters

- serverUrl: Software Risk Manager server URL
- codeDxApiKey: Software Risk Manager API Key
- projectId: Software Risk Manager Project ID
- fingerprint: Optional SHA1 hash of an invalid certificate to make an exception for
- acceptPermanently: Optional boolean for if the exception should be stored permanently in a truststore file.

Returns

`OK` if the report is uploaded successfully.

`EMPTY` if the generated report is empty. The report will not be uploaded to Software Risk Manager.

Views

generateReport

Generates an XML report with request and response data.

Returns

An XML report String.

Splunk

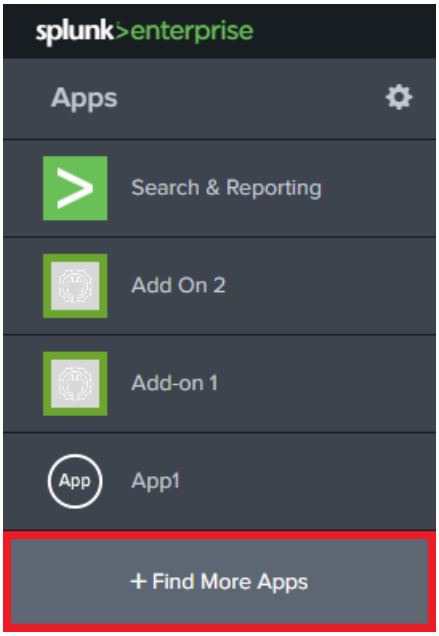
The Software Risk Manager Splunk add-on integrates the Splunk data analysis/monitoring/visualization platform with your Software Risk Manager server. It allows you to push Software Risk Manager findings to your Splunk server as Splunk events.

An API key and at least one Software Risk Manager project are required. The API key must have the `read` role for each project it needs to interact with.

This section of the Plugins Guide explains how to install and use the Splunk add-on. For more information you may visit Splunkbase.

Installation

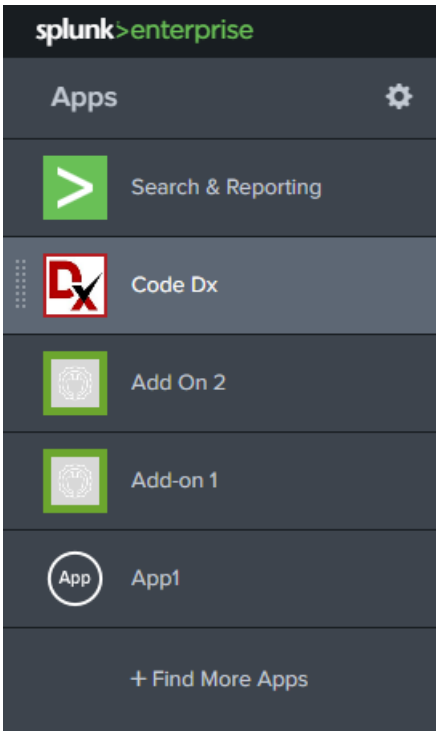
Under *Apps* in the Splunk home screen, click *Find More Apps*.



Search for `Code Dx` and install the *Code Dx Add-On*.

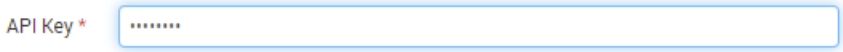
Configuration

Under *Apps* in the Splunk home screen, click *Code Dx* to open the Software Risk Manager add-on.



Go to *Configuration* and click *Add-on Settings*. Then input the URL of your Software Risk Manager server. Keep in mind that your server's URL may or may not have `/codedx` at the end.

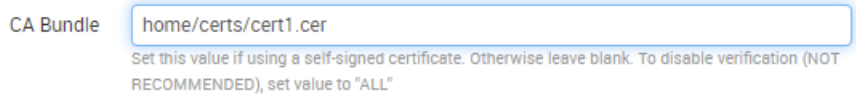
Input your API key.



Click *Save*.

Handling a Self-Signed Certificate

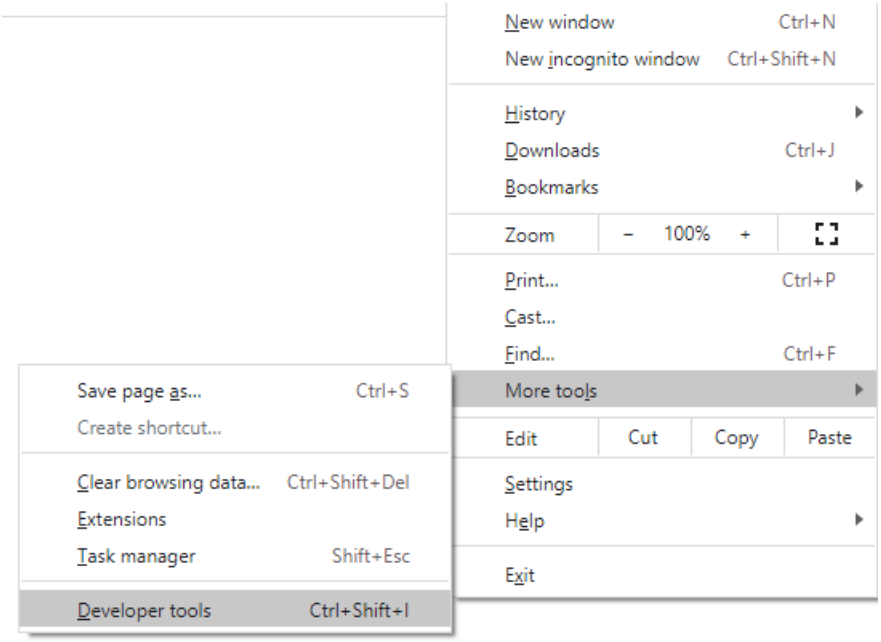
If the server hosting Software Risk Manager is using a self-signed certificate, you must also populate the *CA Bundle* field with the path of a `.pem` or `.cer` certificate file. Alternatively (although not recommended), you can set the value for *CA Bundle* to `ALL` to disable certificate verification.



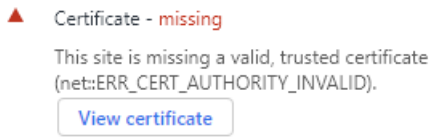
Get a copy of the certificate as a file

The following steps use Windows and Chrome.

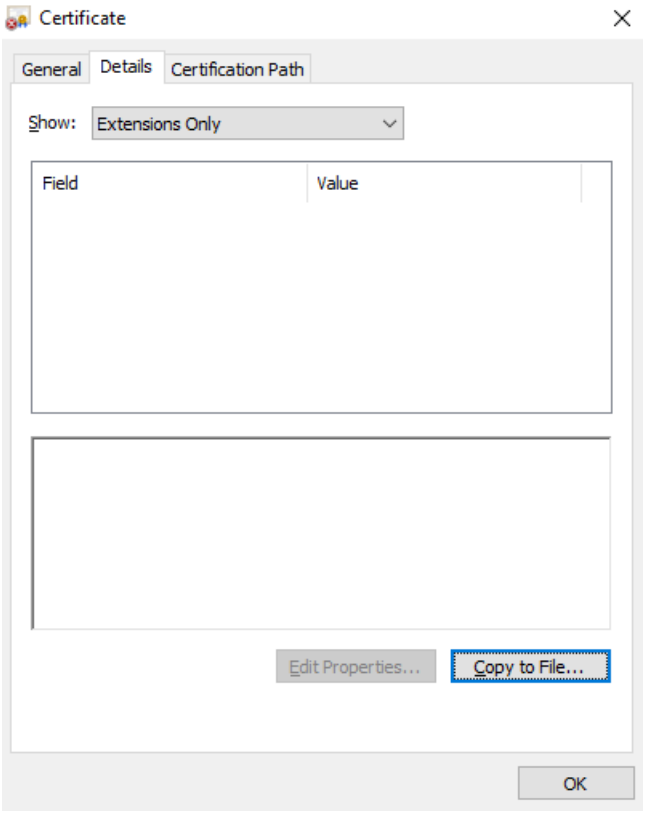
Navigate to the page with the untrusted certificate using your browser (i.e. Chrome). Then bring up Chrome developer tools by opening the Chrome menu (#), then going to *More Tools -> Developer Tools*.



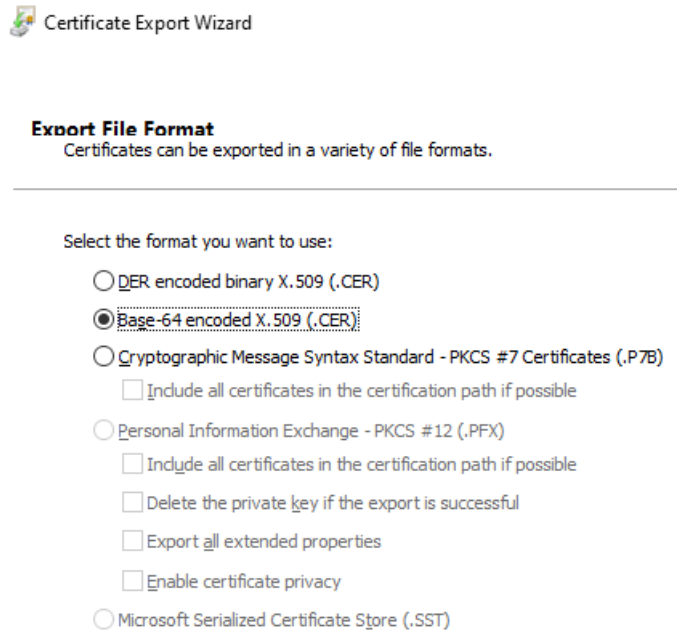
Click the *Security tab* and click the *View certificate* button to bring up the certificate details popup.



Go to the *Details* tab in the *Certificate* popup and click the *Copy to File...* button.



Use the `Base-64 encoded X.509 (.CER)` format, then choose a name and location for the file.

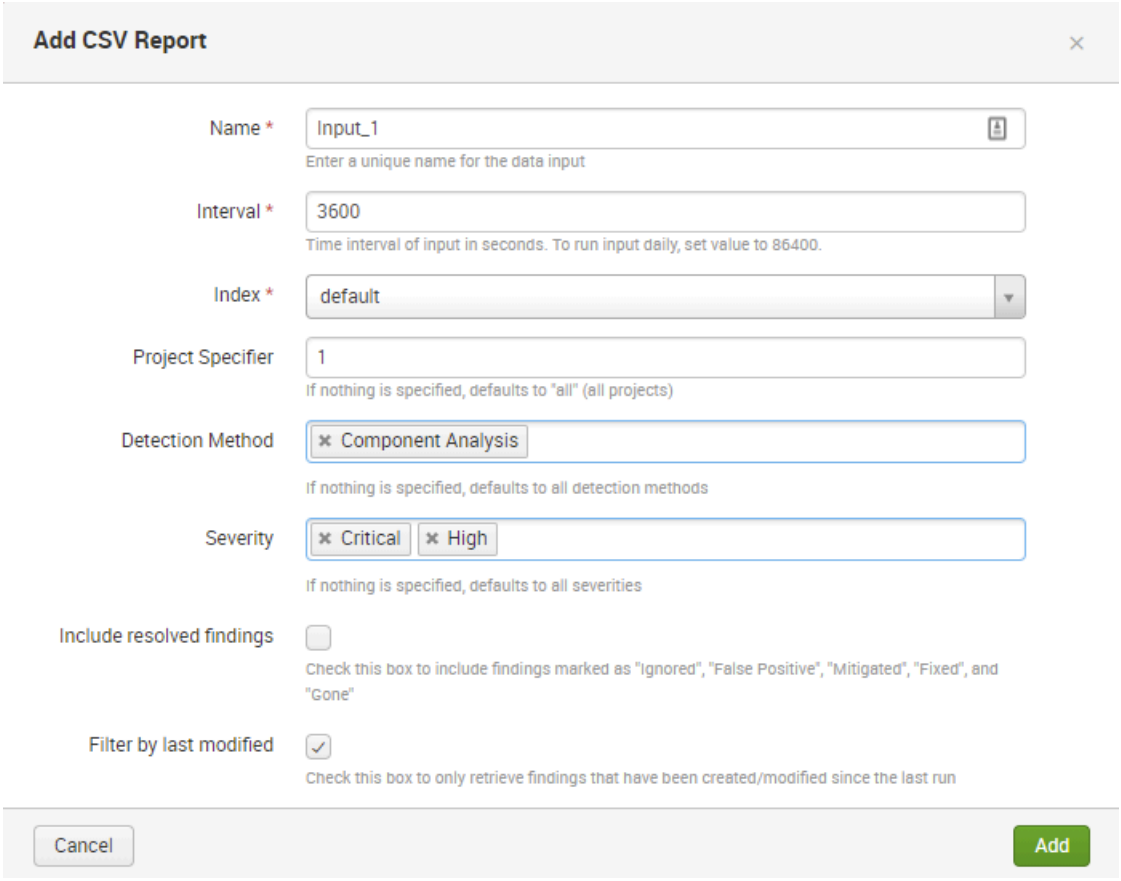


When you click the *Finish* button, it should say "the export was successful."

Using Software Risk Manager with Splunk

Getting Software Risk Manager data into Splunk

Go to *Inputs* and click *Create New Input*. Then fill out the fields to create your first Software Risk Manager data input.

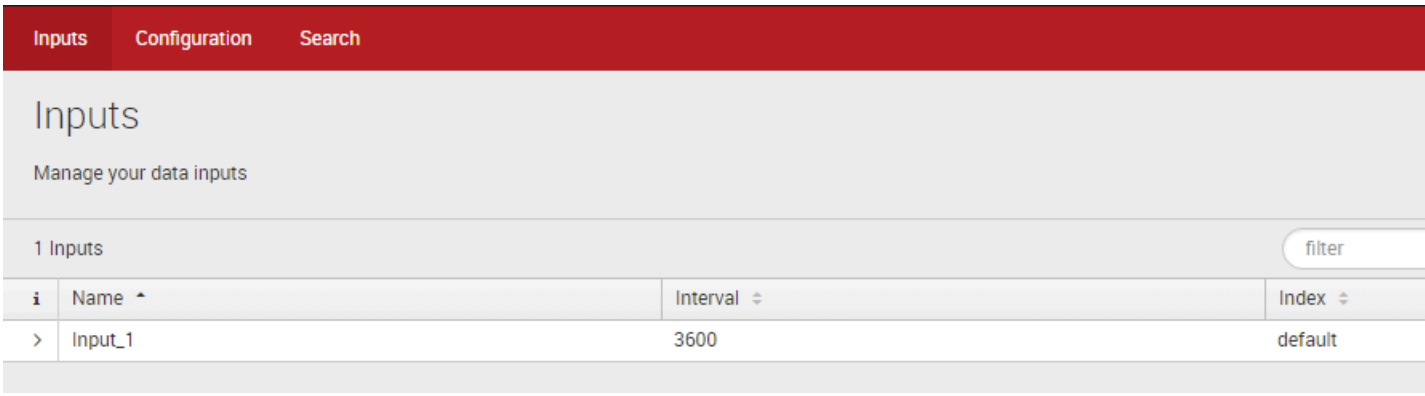


Notes:

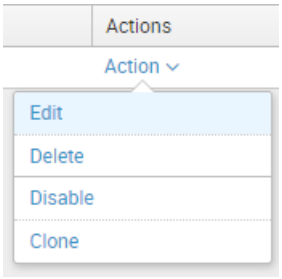
- `Interval` determines how often your data input will run (every `x` seconds)
  - If a run's start-to-finish duration exceeds the time specified by the `Interval` field (for example, if the `Interval` is set to `60` and a particular run takes more than 60 seconds to finish), the next run will wait for that previous run and start as soon as it finishes
- `Project Specifier` can be a project ID or a special string representation of a set of projects:
  - To represent a single project, use that project's ID number, e.g. `12`
  - To represent all projects, use `all`
  - To represent an arbitrary set of projects, join the IDs of each project with an underscore, e.g. `12_42_123_124`
  - To include 'descendant' projects, add a `d` before the IDs of the main projects, e.g. `d12` or `d12_42_123_124` (note that there is only one `d` needed; it applies to each of the specified projects)
- `Detection Method` and `Severity` will filter the data by detection method and severity respectively

- These are both multi-value fields, so if you like you can specify multiple detection methods and/or severities to filter by

To manage a specific input, click on its *Action* button, in the rightmost, *Actions*, column.



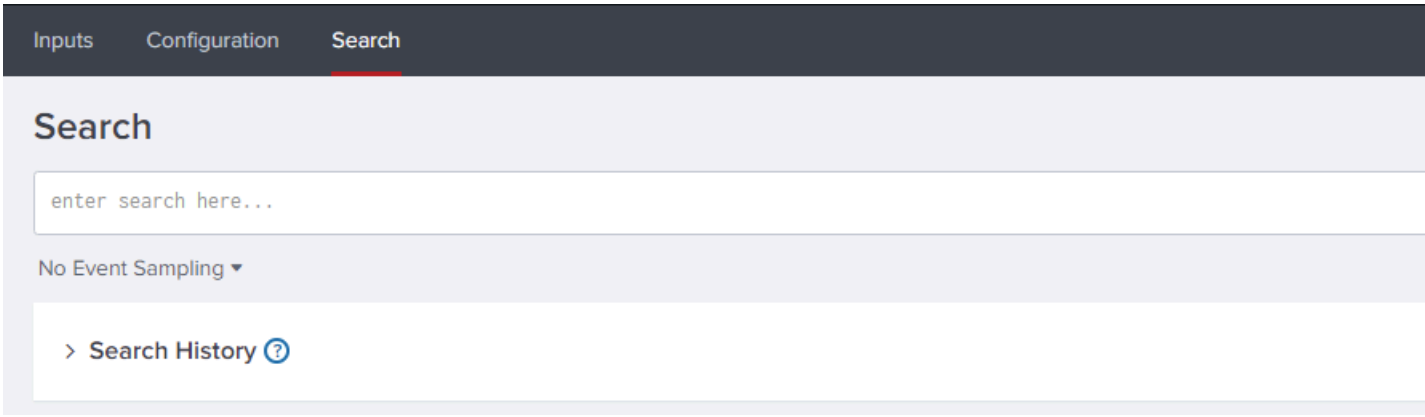
From there you are given 4 options:



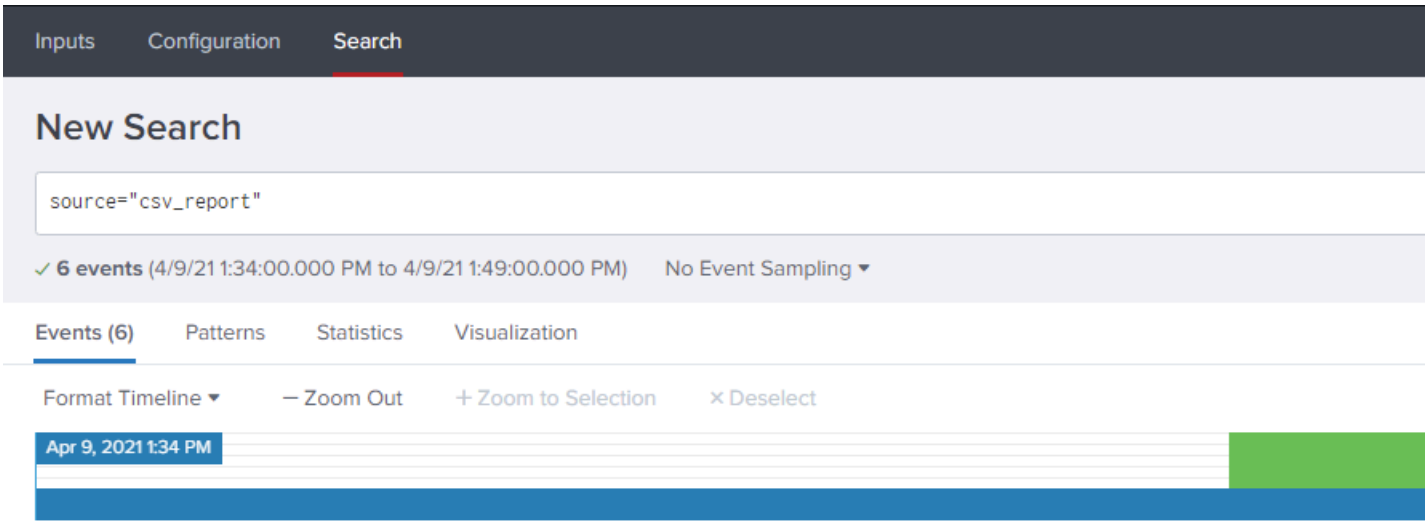
- *Edit*: view and potentially edit the input
- *Delete*: remove the input
- *Enable/Disable*: toggle whether the input is enabled or not
  - Inputs are automatically enabled when first created
  - An input will not run if it is disabled
- *Clone*: create a new input with the same default settings as this input

View your Software Risk Manager data in Splunk

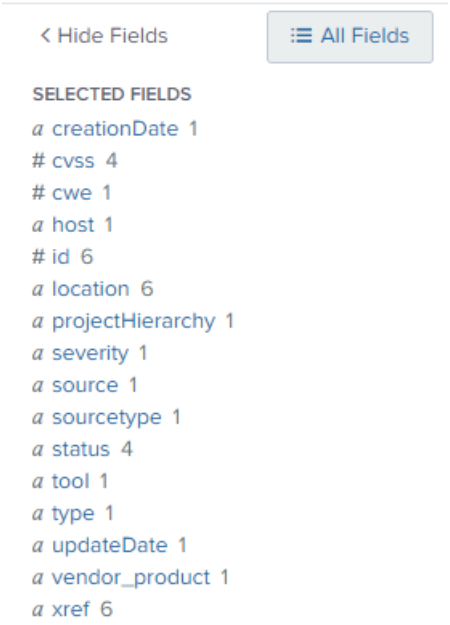
Go to *Search* and search for whatever Software Risk Manager data you want to find.



`source="csv_report"` - A simple search to start off with that gets results from all inputs (all inputs retrieve data from Software Risk Manager through CSV reports)



By default, the `host`, `source`, and `sourcetype` fields are included in *Selected Fields* (on the left sidebar after running a search). You can change which fields are selected by clicking on *All Fields* (also at top of left sidebar) and selecting/deselecting fields.



GitHub Action

The Software Risk Manager GitHub Action enables an Action workflow to upload files and initiate a scan with your Software Risk Manager server.

A Software Risk Manager project and an API key or PAT are required. It must have the `create` role for the project.

The Action can be used on any Runner machine type. The machine should have access to your Software Risk Manager server. If Software Risk Manager is inaccessible to public traffic, we recommend using a Self-Hosted Runner to run the action.

Instructions for installation and configuration can be found on the Marketplace listing and the GitHub repository. This plugin is open source and we welcome community involvement.

Black Duck Bridge CLI Command Line Interface

Overview

The Black Duck Bridge CLI Command Line Interface (Black Duck Bridge CLI) allows you to run scans and receive results from the command line, either by scripting or running the commands manually. Black Duck Bridge CLI invokes adapters that handle Coverity and Black Duck scanning, along with abstracting the various extensions and variables for them, so that usually no manual configuration is required.

For more information on Black Duck Bridge CLI and how to use the command line interface, see *Using Bridge CLI with Software Risk Manager.*

Configuring Coverity Thin Client for use with Black Duck Bridge CLI and SRM

When you use Black Duck Bridge CLI to start a Coverity static scan, Black Duck Bridge CLI calls the Coverity Thin Client to manage code capture. Black Duck Bridge CLI uploads the resulting data and artifacts to SRM.

In some cases a configuration file, usually named `coverity.yaml`, helps Coverity Thin Client by providing capture settings.

When to use a Configuration file

A scan is likely to succeed without any intervention for uncompiled languages like Python, JavaScript, and PHP, because Coverity Thin Client can automatically detect these languages and configure the scan.

Some scans require a configuration file to include project settings or optimize Coverity for best results.

Use a configuration file in the following situations:

- When scanning a compiled language, such as C/C++, C#, Java, and Kotlin, use the configuration file to provide your build and clean commands.
- When scans for a project have failed or not returned useful results.

Note: Build environment components must be installed and accessible when integrating with compiled languages, as illustrated in the "Build commands and dependencies for various build systems" section further down in this guide. If you plan to run build capture, make sure the build tools are installed where the capture will run.

Things you can specify with the Coverity Thin Client configuration file:

- Your build and clean commands
- Whether source code in a specific language should be captured or excluded.
- A custom compiler configuration. This is helpful if you are compiling C++ and Coverity does not automatically configure the compiler (for example, when using a GCC cross-compiler).

How to start a configuration file

Save a `coverity.yaml` file in your project's root directory and use the following examples to customize that file. When Coverity Thin Client is called by Black Duck Bridge CLI, it automatically detects the file, reads the contents, and executes custom configurations.

Basic examples

For more build and clean commands see the reference section at the end of the chapter.

Java with Maven

The following configuration specifies the clean and build commands needed by Maven. The project will be cleaned by invoking `mvn clean` and built using `mvn install`.

```
capture:
   build:
      clean-command: mvn clean
      build-command: mvn install
```

About this example:

- This example shows the complete contents of a `coverity.yaml` file that you can use to test a Maven project.

- Note that white space matters in YAML files. The three levels of indentation indicate that the build property is nested under the capture property and build-command and clean command are nested under build.

Standard Java example

```
capture:
   build:
      build-command:  javac HelloWorld.java
```

C# with msbuild

```
capture:
   build:
      clean-command: dotnet msbuild /t:Restore;Clean OWASPTop10.SqlInjection.Web.csproj
       build-command: dotnet msbuild /t:Restore;Build OWASPTop10.SqlInjection.Web.csproj
```

C# with dotnet

```
capture:
   build:
      clean-command: dotnet clean Foo.sln
      build-command: dotnet build Foo.sln
```

Executable script as a build command

```
capture:
   build:
      clean-command: clean.sh
      build-command: build-it.sh
```

Coverity Thin Client will only accept one build command and one clean command in the config file, but you can point to a script containing commands if you need more.

Configuration properties

The tables in this section describe each of the keys that can be used in the `coverity.yaml` file, starting at the top level with capture.

Capture settings

The following table describes the capture configuration; subsections describe the fields that make up the Capture type. All the Keys in the first column of this table can be nested directly under the capture key in the `coverity.yaml` file. (For example, build can be seen in the first row of the table and is used in the example immediately above.)

Key	Type	Description
build	Build Configuration	Specifies that build capture should be used to capture the project and provides the build configuration to use. If not specified and the project directory contains compiled source files, automatic build capture will be used to capture compiled source files in the project directory. For compiled languages only: Java, Kotlin, C#, C, C++, Objective-C, Objective-C++. See the Build configuration table below for keys that can be nested under this one.
encoding	string	Specifies the encoding to use when parsing and emitting the source files in C, C++, JavaScript.**Default**: US-ASCII

Key	Type	Description
languages	Languages Configuration	Specifies which languages to include or exclude for capture. See Language configuration for a list of supported languages. **Default**: all languages are included See the table in the Language configuration section for keys that can be nested under the languages key.

Build settings

The following keys can be defined for compiled languages only. All the keys in this table can be nested directly under the build key in the configuration file.

Key	Type	Description
aspnet-compiler	Boolean	Specifies whether to enable or disable the automatic invocation of `Aspnet_compiler.exe` for any ASP.NET 4 and earlier Web applications that are detected in the build. The output of `Aspnet_compiler.exe` is required by the C# security checker.
build-command	String	**Required**: The build command will be invoked to use build capture to capture the project. A build command specified on the command-line will override this setting.
clean-command	String	The clean command will be invoked prior to doing build capture to capture the project.

For example:

```
capture:
   build:
      clean-command: dotnet msbuild /t:Restore;Clean OWASPTop10.SqlInjection.Web.csproj
      build-command: dotnet msbuild /t:Restore;Build
      aspnet-compiler: true
```

Language configuration

Use the following keys to specify which languages to include or exclude from capture. You may not specify both fields.

Language strings may be the following:

- apex
- c-family *(This family includes C, C++, Objective C, and Objective C++)*
- csharp
- go
- java *(Includes JSP and android config files)*
- javascript *(Includes JavaScript and TypeScript)*
- kotlin
- php
- python
- ruby

- swift
- vb *(Visual Basic)*

The following languages are not supported by the Coverity CLI:

- CUDA
- Fortran
- Scala

Keys in this table can be nested under the `languages` key, which appears in the first table above and is nested under the `capture` key.

Key	Type	Description
include	Array of String	Specifies the languages for which the source code should be included in the capture. This key is mutually exclusive with the exclude key. **Default**: all languages are included.
exclude	Array of String	Specifies the languages for which the source code should be excluded in the capture. This key is mutually exclusive with the include key. **Default**: No languages are excluded.

For example:

```
capture:
   languages:
      exclude:
         - python
```

Build commands and dependencies for various build systems

You may need to install the machine dependencies in order to run the corresponding build. If build and clean commands don't work—check for the dependencies.

Build System	How to Detect	Minimal Set of Machine Dependencies	Build and Clean commands
Maven	Look for the presence of "pom.xml" files in the project directory	Maven build tool Java compiler	`build-command:` mvn -DskipTests -DskipITs install `clean-command:` mvn clean
Gradle	Look for the presence of "build.gradle" files in the project directory		
	With a "gradlew" file (Linux/Mac)	Java compiler (Gradle will be downloaded automatically)	`build-command:` gradlew --no-daemon -x test build testClasses `clean-command:` gradlew clean
	With a "gradlew.bat file (Windows)	Java compiler (Gradle will be downloaded automatically)	`build-command:` gradlew.bat --no-daemon -x test build testClasses `clean-command:` gradlew.bat clean

Build System	How to Detect	Minimal Set of Machine Dependencies	Build and Clean commands
	Without a "gradlew" or "gradlew.bat" file	Gradle build tool Java compiler	`build-command:` gradle --no-daemon -x test build testClasses `clean-command:` gradle clean
Make	Look for the presence of a file called "Makefile"	Make build tool Compiler for the appropriate language	`build-command:` make `clean-command:` make clean
Ant	Look for the presence of a "build.xml"	Ant build tool Java compiler	`build-command:` ant `clean-command:` ant clean
MSBuild	Look for the presence of files with ".sln" or ".csproj" extensions.	.NET SDK	`build-command:` dotnet msbuild /t:Build `clean-command:` dotnet msbuild /t:Restore;Clean
XCode	Look for the presence of directories ending with ".xcodeproj"	XCode	`build-command:` xcodebuild -workspace <xcodeproj dir> -scheme <scheme> build -UseModernBuildSystem=No CODE_SIGN_IDENTITY= CODE_SIGNING_REQUIRED=NO `clean-command:` xcodebuild -workspace <xcodeproj dir> -scheme <scheme> clean

GCC configuration

The GNU Compiler Collection (GCC) requires its own `coverity.yaml` configuration settings. To use GCC, add lines to:

- Specify the GCC compiler using `cov-configure`.
- Designate the language and CPU target:

```
cov-configure:
    - [ --template, --compiler, arm-linux-gnueabi-gcc, --comptype, gcc ]
    - [ --template, --compiler, arm-linux-gnueabi-g+, --comptype, g+ ]
```

Below is an example `coverity.yaml` settings block for GCC, telling it to run a `make clean` then run the `make` command in parallel with at least 10 job slots (`make -j 10`).

```
C/C++ Cross Compiler coverity.yaml example:
-------------------------------------------
# Coverity configuration file.
# The schema is available here: <install-dir>/doc/configuration-schema.json
capture:
  build:
    clean-command: "make clean"
    build-command: "make -j 10"
  compiler-configuration:
    cov-configure:
    - [ --template, --compiler, arm-linux-gnueabi-gcc, --comptype, gcc ]
    - [ --template, --compiler, arm-linux-gnueabi-g+, --comptype, g+ ]
```

Note: Pay special attention to GCC binary naming conventions if using more than one language and/or CPU target.