

# Parasoft Findings for SonarQube

---

In this section:

- Introduction
- Requirements
- Installing the Parasoft Findings Plugin
- Activating Parasoft Profiles
- Setting the Report Path
- Uploading Project Results
- Viewing Test Execution Results
- Unit Test Execution and Coverage

---

View online: <https://docs.parasoft.com/x/Ny1-C>

# Introduction

The Parasoft Findings Plugin for SonarQube allows you to view static analysis test results within SonarQube. It grants SonarQube the ability to analyze data from Parasoft XML reports and use it to report bugs, vulnerabilities, or code smells from within SonarQube. See [Uploading Project Results](#) for more details. While viewing test execution and coverage results is not part of the implementation for Parasoft Findings for SonarQube, see [Unit Test Execution and Coverage](#) to learn how SonarQube can support that functionality natively.

The plugin can consume the following report types:

- Static analysis, metrics analysis, and unit test reports generated by 2022.2+ versions of C/C++test, Jtest, and dotTEST.

# Requirements

- For version 10.6.1, a Parasoft product must be installed on the same machine as the SonarQube server since the plugin loads rules from a Parasoft product installation.
  - For C/C++test, the Standard edition must be installed even if you are sending C/C++test Professional reports to SonarQube.
  - This requirement does not apply to version 10.6.2+.
- SonarQube versions 8.9+ are supported.
- Language projects versions 2022.2+ are supported.

# Installing the Parasoft Findings Plugin

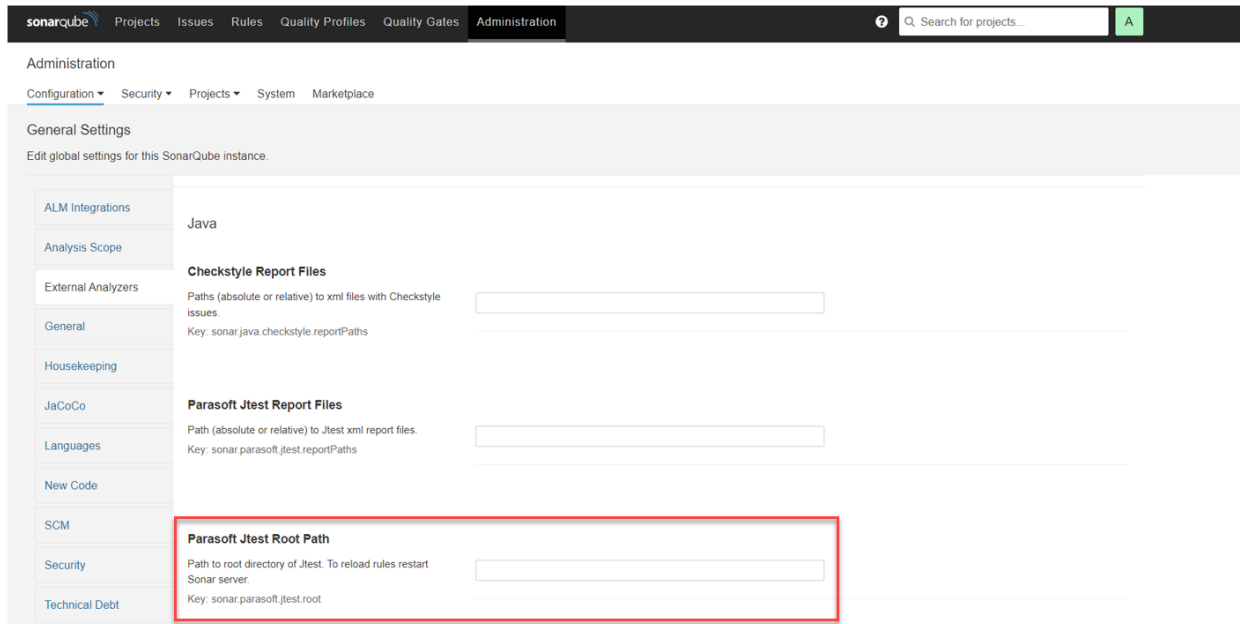
1. Either download the plugin or build it yourself (10.6.2+ only):
  - a. To download the plugin, go to <https://customerportal.parasoft.com/lightningportal/s/marketplace> and download the Parasoft SonarQube plugin jar `parasoft.findings.sonar-<VERSION>.jar`.
    - Be sure to download the version of the plugin that matches the version of your Parasoft product. For example, if you are using Jtest, dotTEST, or C++Test 2022.2, you would download `parasoft.findings.sonar-2022.2.jar`.
  - b. To build the plugin yourself, see [Building Your Own Plugin](#).
2. Copy the plugin jar into the SonarQube `extensions/plugins` directory.
3. Restart the SonarQube server.
4. For version 10.6.1: set the root path to the Parasoft product (see [Setting the Root Path to the Parasoft Product for Version 10.6.1](#)).
  - a. Starting with version 10.6.2, rule files are packaged in the plugin jar, so it is not necessary to set the root path to load rules with version 10.6.2+.

See <https://docs.sonarqube.org/latest/setup/install-plugin/> for more details.

## Setting the Root Path to the Parasoft Product for Version 10.6.1

1. In the SonarQube web UI go to **Administration > Configuration > General Settings > External Analyzers**.

2. Locate the **Root Path** setting under the appropriate programming language.



3. In the field enter the absolute path to the Parasoft product installation and click **Save**.
4. Restart the SonarQube server so that the rule definitions are loaded.
5. After the server has restarted, go to **Quality Profiles** in the web UI and select the built-in Parasoft profile under the appropriate supported language. Verify that the rule definitions are loaded successfully under the profile.

If zero or only one rule is loaded, then loading of rules failed. In this case:

- a. Check that the root path is correct.
- b. Review the SonarQube web server logs for any error messages.

## Building Your Own Plugin

You can build your own plugin for your Parasoft products. To do so, you will need JDK 11+, Maven 3.3+, and your Parasoft products installed on the same machine.

1. Clone the source code from GitHub found here: <https://github.com/parasoft/parasoft-findings-sonar>.

View online: <https://docs.parasoft.com/x/Ny1-C>

2. Run a Maven package command that is appropriate for the Parasoft products you have installed. The example below includes Jtest, dotTEST, and C++Test; if you don't have one or more of these installed, remove its root path reference:

```
mvn clean package -DjtestRootPath="<JTEST-INSTALL-ROOT-PATH>" -DdottestRootPath="<DOTTEST-INSTALL-ROOT-PATH>" -DcpptestRootPath="<CPPTTEST-INSTALL-ROOT-PATH>"
```

- For C/C++test, the root path must be set to Standard edition, even if you are sending C/C++test Professional reports to SonarQube.

The plugin jar that is created will be in the `<SOURCE-CODE-ROOT-PATH>/target` folder and will include the rule files for your Parasoft product.

# Activating Parasoft Profiles

There are two ways to enable the Parasoft profile for your projects:

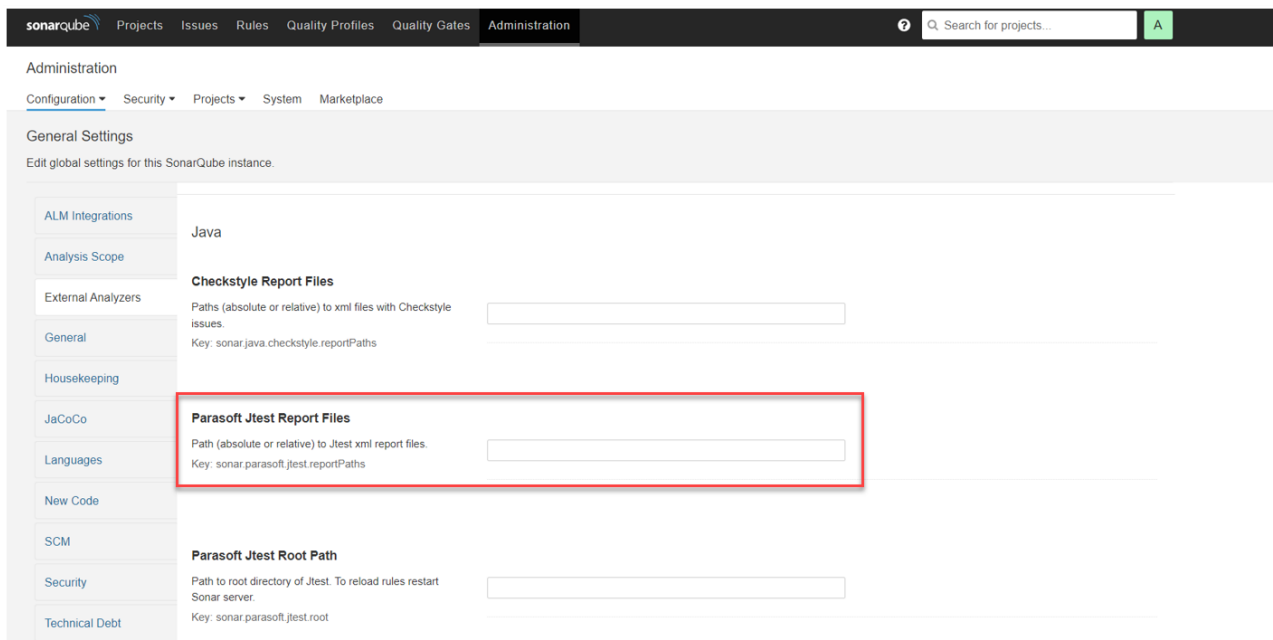
- To set a Parasoft profile as the default for new projects, go to **Quality Profiles** and select **Set as Default** from the gear menu to the right of the profile.
- To set a Parasoft profile for a given project, go to **Quality Profiles** section in the project settings. Select the **Change Profile** button on the right side of the appropriate language.



# Setting the Report Path

For Parasoft results to be uploaded, the path to the report files must be set. There are several ways to configure this.

- To configure the path globally go to **Administration > Configuration > General Settings > External Analyzers** and find the **Report Files** setting for your Parasoft product.



Enter the path to the report.xml file of your project. For example:

```
target/jtest/report.xml
```

You can add multiple paths, each in a separate field. When the Parasoft scanner runs, results will be loaded from each valid report file path.

- To configure the path for a project, go to the **External Analyzers** section in the project's settings.
- To configure the path when running the sonar scanner, include the report path's settings key. For example:
  - Jtest using Maven:

```
mvn sonar:sonar -Dsonar.login=<your sonar key> -Dsonar.host.url="http://  
<your sonar server>:<your sonar server port>/"  
-Dsonar.parasoft.jtest.reportPaths=target/report.xml...<additional sonar  
settings>
```

- Jtest using sonar-scanner:

```
sonar-scanner-<version>-windows/bin/sonar-scanner.bat -Dsonar.login=<your  
sonar key> -Dsonar.host.url="http://<your sonar server>:<your sonar server  
port>/" -Dsonar.parasoft.jtest.reportPaths=target/report.xml...<additional  
sonar settings>
```

- C/C++test using sonar-scanner:

```
build-wrapper-win-x86-64.exe --out-dir  
build_wrapper_output_directory...<your build commands i.e.: make clean all  
"C:/cpptest/examples/FlowAnalysisCpp">
```

```
sonar-scanner-<version>-windows/bin/sonar-scanner.bat -Dsonar.login=<your  
sonar key> -Dsonar.host.url="http://<your sonar server>:<your sonar server  
port>/" -Dsonar.parasoft.cpptest.reportPaths=target/  
report.xml...<additional sonar settings>
```

- dotTEST using .NET's sonar-scanner (**Note:** Sonar-scanner must be installed via the command: `dotnet tool install --global dotnet-sonarscanner`):

```
dotnet sonarscanner begin /k:"<your solution>" /d:sonar.login="<your sonar  
key>" /d:sonar.parasoft.dottest.reportPaths="target/report.xml"...<addition  
al sonar settings>
```

```
dotnet build <your solution>
```

```
dotnet sonarscanner end /d:sonar.login="<your sonar key>"
```

- dotTEST using Sonar's provided sonar-scanner:

```
sonar-scanner-<version>-windows/bin/sonar-scanner.bat -Dsonar.login=<your  
sonar key> -Dsonar.host.url="http://<your sonar server>:<your sonar server  
port>/" -Dsonar.parasoft.dottest.reportPaths=target/  
report.xml...<additional sonar settings>
```

- Sonar-scanner can be downloaded from: <https://docs.sonarqube.org/latest/analyzing-source-code/scanners/sonarscanner/>
- You can include multiple report files:

```
mvn sonar:sonar -Dsonar.parasoft.jtest.reportPaths=target/report.xml  
-Dsonar.parasoft.jtest.reportPaths=target/report_2.xml...<additional sonar  
settings>
```

# Uploading Project Results

To upload Parasoft results for a project, first run the command which generates the report.xml file, and then run the usual Sonar scanner command. For example, with a Maven project setup using Jtest, a single command can be run from the root directory of your project:

```
mvn jtest:jtest sonar:sonar -Dsonar.projectKey=<project key> -Dsonar.host.url=<server url> -Dsonar.login=<login token>
```

See the documentation for your Parasoft product for details on how to run static analysis and generate a report:

- For Jtest: <https://docs.parasoft.com/display/JTEST20231/Running+Static+Analysis+1>
- For dotTEST: <https://docs.parasoft.com/display/DOTTEST20231/Running+Static+Analysis+1>
- For C/C++test: <https://docs.parasoft.com/display/CPPTTEST20231/Running+Static+Analysis+1>

**Note:** The `report.location_details` property must be set to `true` in order for C/C++test Professional to work properly.

See <https://docs.sonarqube.org/latest/analysis/overview/> for details on how to run the Sonar scanner on projects using different build systems.

# Viewing Test Execution Results

After the results are uploaded, they can be viewed in the project in the **Issues** tab.

The screenshot displays the SonarQube web interface for a project named "Demo Project". The top navigation bar includes links for Projects, Issues, Rules, Quality Profiles, Quality Gates, and Administration. A search bar and a user profile icon are also present. Below the navigation bar, the "Issues" tab is selected, showing a list of issues. The left sidebar contains filters for Type (BUG, 128), Severity (Blocker: 70, Critical: 22, Major: 36, Minor: 0, Info: 0), Scope, Resolution, Status, Security Category, Creation Date, and Language. The main content area shows a list of issues for the file "src/.../java/examples/flowanalysis/AlwaysCloseGSS.java". The issues are categorized by severity and type, with details on their status and resolution.

Type	Severity	Issue Description	Status	Resolution	Creation Date
Bug	Blocker	Security context not disposed: GSSManager.getInstance().createContext(tokens)	Open	Not assigned	4 minutes ago
Bug	Blocker	Security context not disposed: GSSManager.getInstance().createContext(tokens)	Open	Not assigned	4 minutes ago
Bug	Blocker	"context" may possibly be null	Open	Not assigned	4 minutes ago
Bug	Blocker	"context" may possibly be null	Open	Not assigned	4 minutes ago
Bug	Blocker	Image resource not closed: imgReader	Open	Not assigned	4 minutes ago
Bug	Blocker	Image resource not closed: imgReader	Open	Not assigned	4 minutes ago
Bug	Blocker	"imgReader" may possibly be null	Open	Not assigned	4 minutes ago

View online: <https://docs.parasoft.com/x/Ny1-C>

## Viewing Test Execution Results

The screenshot displays the SonarQube web interface for a project named "Demo Project" on the "master" branch. The top navigation bar includes links for Projects, Issues, Rules, Quality Profiles, Quality Gates, and Administration. A search bar and a green "A" button are also present. The main header shows the last analysis had 1 warning on August 8, 2022, at 10:16 AM, with version 1.0.0.

The left sidebar contains a list of issues categorized by type (Bug, Error, Warning, Info) and severity (Critical, Major, Minor, Info). The main content area shows the source code for the file `...amples/flowanalysis/AlwaysCloseGSS.java`. The code is a Java class `AlwaysCloseGSS` with two methods: `process` and `processClose`. Both methods use `GSSManager.getInstance().createContext(tokens)` to create a security context. The `process` method has a try-catch block that catches `IOException` and prints an error message. The `processClose` method also has a try-catch block that catches `IOException` and prints an error message.

Two security issues are highlighted in the code editor:

- Security context not disposed: GSSManager.getInstance().createContext(tokens)** (Bug, Critical). This issue is located at line 11 in the `process` method. It is marked as a "Blocker" and has a severity of "Critical".
- Security context not disposed: GSSManager.getInstance().createContext(tokens)** (Bug, Critical). This issue is located at line 22 in the `processClose` method. It is also marked as a "Blocker" and has a severity of "Critical".

Both issues have a "Why is this an issue?" link and a "5 minutes ago" timestamp. The user "bd.res, parasoft" is listed as the reporter.

View online: <https://docs.parasoft.com/x/Ny1-C>

# Unit Test Execution and Coverage

Reporting test execution and test coverage should be done using the existing industry standard solutions supported by SonarQube. Other unit test and coverage frameworks are not currently supported.

1. Run unit tests to generate test execution and coverage reports.
2. Run SonarQube scanner to upload reports.

See <https://docs.sonarqube.org/latest/analysis/coverage/> for more details.