opentext™

# OpenText™ Fortify Static Code Analyzer

Software Version: 24.4.0

# User Guide

Document Release Date: October 2024
Software Release Date: October 2024

# Legal Notices

Open Text Corporation

275 Frank Tompa Drive, Waterloo, Ontario, Canada, N2L 0A1

# Copyright Notice

Copyright 2003 - 2024 Open Text.

The only warranties for products and services of Open Text and its affiliates and licensors (“Open Text”) are as may be set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. Open Text shall not be liable for technical or editorial errors or omissions contained herein. The information contained herein is subject to change without notice.

# Trademark Notices

“OpenText” and other Open Text trademarks and service marks are the property of Open Text or its affiliates. All other trademarks or service marks are the property of their respective owners.

# Documentation Updates

The title page of this document contains the following identifying information:

- Software Version number
- Document Release Date, which changes each time the document is updated
- Software Release Date, which indicates the release date of this version of the software

This document was produced for OpenText™ Fortify Static Code Analyzer CE 24.4 on October 16, 2024. To check for recent updates or to verify that you are using the most recent edition of a document, go to:

https://www.microfocus.com/support/documentation

# Contents

# Preface

## Contacting Customer Support

Visit the Support website to:

- Manage licenses and entitlements
- Create and manage technical assistance requests
- Browse documentation and knowledge articles
- Download software
- Explore the Community

https://www.microfocus.com/support

## For More Information

For more information about Fortify software products:

https://www.microfocus.com/cyberres/application-security

## About the Documentation Set

The Fortify Software documentation set contains installation, user, and deployment guides for all Fortify Software products and components. In addition, you will find technical notes and release notes that describe new features, known issues, and last-minute updates. You can access the latest versions of these documents from the following Product Documentation website:

https://www.microfocus.com/support/documentation

To be notified of documentation updates between releases, subscribe to Fortify Product Announcements on the OpenText Fortify Community:

https://community.microfocus.com/cyberres/fortify/w/announcements

## Fortify Product Feature Videos

You can find videos that highlight Fortify products and features on the Fortify Unplugged YouTube channel:

https://www.youtube.com/c/FortifyUnplugged

# Change log

The following table lists changes made to this document. Revisions to this document are published between software releases only if the changes made affect product functionality.

| Software release / Document version | Changes |
|---|---|
| 24.4.0 | Updated:<br>• Added installer for Linux on ARM (see "Installing Fortify Static Code Analyzer" on page 28)<br>• Scan policies can exclude dataflow issues based on taint flags (see "Applying a scan policy to the analysis" on page 46)<br>• By default, NPM dependencies are excluded from the analysis phase (see "Managing issue detection in NPM dependencies" on page 72)<br>• Support added for Flask and Jinja2 (see "Translating Python code" on page 77)<br>• Added the `-gotags` option to include custom build tags in Fortify Static Code Analyzer translation of Go project (see "Including custom Go build tags" on page 87 and "Go properties" on page 204)<br>• Changes to the command-line options to analyze PL/SQL (see "Analyzing SQL" on page 113)<br>• Added an option to disable build tool name resolution and translate build script files as source files (see "Translation options" on page 134 and "Translation and analysis phase properties" on page 186)<br>• The `-exclude` option is supported in Ant, Bazel, Gradle, and Maven build integrations (see "Integrating with Ant" on page 123, "Integrating with Bazel" on page 123, "Using Gradle integration" on page 125, "Using the Fortify Maven Plugin" on page 132, and "Translation options" on page 134)<br><br>Removed:<br>• Modular analysis was removed from this document. This feature is deprecated and will be removed from the product in the next release. |

| Software release / Document version | Changes |
|---|---|
| 24.2.0 | Added:<br>• "Integrating with Bazel" on page 123<br>• "Integrating with CMake" on page 125<br><br>Updated:<br>• The default scan policy has changed (see "Applying a scan policy to the analysis" on page 46)<br>• Option added to specify a JDK version that is not distributed with Fortify Static Code Analyzer to use for translation (see "Java command-line options" on page 51)<br>• The default Python version has changed (see "Python command-line options" on page 77)<br>• The default PHP version has changed (see "PHP command-line options" on page 102)<br><br>Removed:<br>• The `-apex` option and corresponding configuration property are deprecated and no longer required to translate Apex and Visualforce code |
| 23.2.0 | Added:<br>• "Translating Python in a virtual environment" on page 79<br>• "Analyzing Solidity code" on page 100<br>• "Troubleshooting Gradle integration" on page 126<br>• "Using the Gradle plugin" on page 127<br><br>Updated:<br>• Improved the example Dockerfile to install Fortify Static Code Analyzer (see "Creating a Dockerfile to install Fortify Static Code Analyzer" on page 33)<br>• Added considerations about generated code for the translation phase ("Translation phase" on page 43)<br>• Added instructions for including source code in the MBS file (see "Mobile build sessions" on page 44) |

| Software release / Document version | Changes |
|---|---|
| | • The default JDK version was changed from 1.8 to 11 (see "Java command-line options" on page 51 and "Kotlin command-line options" on page 59)<br>• Improved instructions for translating Java bytecode (see "Translating Java bytecode" on page 56)<br>• Improved instructions for excluding NPM dependencies (see "Managing issue detection in NPM dependencies" on page 72)<br>• Added a property to enable translation of minified JavaScript files (see "JavaScript and TypeScript properties" on page 200)<br>• Added rule properties for Apex and PowerShell (see "fortify-rules.properties" on page 216)<br><br>Removed:<br><br>• Removed all mentions of the `-fcontainer` option for running a Fortify Static Code Analyzer image as a container as it is no longer necessary and has been removed. |
| 23.1.0 | Added:<br><br>• "Applying a scan policy to the analysis" on page 46<br>• "Translating Dart and Flutter code" on page 89<br>• New properties available for rules (see "Properties files" on page 184 and "fortify-rules.properties" on page 216)<br><br>Updated:<br><br>• Installation of Fortify Static Code Analyzer is now separate from the installation of Fortify Applications and Tools (see "Installing Fortify Static Code Analyzer" on page 27)<br>• New command-line syntax for .NET projects (see "Visual Studio Project command-line syntax" on page 62)<br>• New scan policy analysis option (see "Analysis options" on page 136 and "Translation and analysis phase properties" on page 186)<br>• New filter types used for filter files and scan policy files (see "Excluding issues with filter files" on page 178) |

# Chapter 1: Introduction

This guide provides instructions for using Fortify Static Code Analyzer to scan code on most major programming platforms. This guide is intended for people responsible for security audits and secure coding.

This section contains the following topics:



## Fortify Static Code Analyzer

Fortify Static Code Analyzer is a set of software security analyzers that search for violations of security-specific coding rules and guidelines in a variety of languages. Fortify Static Code Analyzer produces analysis information to help you deliver more secure software, and make security code reviews more efficient, consistent, and complete. Its design enables you to incorporate customer-specific security rules.

For a list of supported languages, libraries, compilers, and build tools, see the *Fortify Software System Requirements* document.

At the highest level, using Fortify Static Code Analyzer involves:

1. Running Fortify Static Code Analyzer as a stand-alone process or integrating Fortify Static Code Analyzer in a build tool
2. Translating the source code into an intermediate translated format
3. Scanning the translated code and producing security vulnerability analysis results
4. Auditing the results of the scan, either by opening the results (typically an FPR file) in OpenText™ Fortify Audit Workbench or uploading them to OpenText™ Fortify Software Security Center for analysis, or working directly with the results displayed on screen.

**Note:** For information about how to open and view results in Fortify Audit Workbench or Fortify Software Security Center, see the *OpenText™ Fortify Audit Workbench User Guide* or the *OpenText™ Fortify Software Security Center User Guide*, respectively.

## About the analyzers

Fortify Static Code Analyzer comprises eight vulnerability analyzers: Buffer, Configuration, Content, Control Flow, Dataflow, Null Pointer, Semantic, and Structural. Each analyzer accepts a different type of rule specifically tailored to provide the information necessary for the corresponding type of analysis performed. Rules are definitions that identify elements in the source code that might result in security vulnerabilities or are otherwise unsafe. The following table describes each analyzer.

| Analyzer | Description |
|---|---|
| Dataflow | The Dataflow Analyzer detects potential vulnerabilities that involve tainted data (user-controlled input or private data) put to potentially dangerous use. The Dataflow Analyzer uses interprocedural taint propagation analysis to detect the flow of data between a site of user input (or private data) through the application to a dangerous function call or operation. For example, the Dataflow Analyzer detects whether a user-controlled input string dynamically generates HTML (Cross-Site Scripting) and detects whether a user-controlled string constructs SQL queries (SQL injection). |
| Control Flow | The Control Flow Analyzer detects potentially dangerous sequences of operations. By analyzing control flow paths in a program, the Control Flow Analyzer determines whether a set of operations are executed in a certain order. For example, the Control Flow Analyzer detects time of check/time of use issues and race conditions, and checks whether utilities, such as XML readers, are configured properly before being used. |
| Buffer | The Buffer Analyzer detects buffer overflow vulnerabilities that involve writing or reading more data than a buffer can hold. The buffer can be either stack-allocated or heap-allocated. The Buffer Analyzer uses limited interprocedural analysis to determine whether there is a condition that causes the buffer to overflow. If any execution path to a buffer leads to a buffer overflow, Fortify Static Code Analyzer reports it as a buffer overflow vulnerability and points out the variables that might cause the overflow. If the value of the variable causing the buffer overflow is tainted (user-controlled), then Fortify Static Code Analyzer reports it as well and displays the dataflow trace to show how the variable is tainted. |

| Analyzer | Description |
|---|---|
| Structural | The Structural Analyzer detects potentially dangerous flaws in the structure or definition of the program. By understanding the way programs are structured, the Structural Analyzer identifies violations of secure programming practices and techniques that are often difficult to detect through inspection because they encompass a wide scope involving both the declaration and use of variables and functions. For example, the Structural Analyzer detects hard-coded secrets, cookie misconfiguration in code, and encryption weaknesses. |
| Configuration | The Configuration Analyzer searches for mistakes, weaknesses, and policy violations in application deployment configuration files. For example, the Configuration Analyzer checks for reasonable timeouts in user sessions in a web application. The Configuration Analyzer also performs regular expression analysis (see "Regular expression analysis" on page 47). |
| Semantic | The Semantic Analyzer detects potentially dangerous uses of functions and APIs at the intra-procedural level. |
| Content | The Content Analyzer searches for security issues and policy violations in HTML content. In addition to static HTML pages, the Content Analyzer performs these checks on files that contain dynamic HTML, such as PHP, JSP, and classic ASP files. |
| Null Pointer | The Null Pointer Analyzer detects dereferences of pointer variables that are assigned the null value. The Null Pointer Analyzer detection is performed at the intra-procedural level. Issues are detected only when the null assignment, the dereference, and all the paths between them occur within a single function. |

# Licensing

Fortify Static Code Analyzer requires a license to perform both the translation and analysis (scan) phases of security analysis (for more information about these phases, see "Analysis process" on page 42). For details on how to obtain a license for Fortify Static Code Analyzer, see the *Fortify Software System Requirements* document.

You must have a Fortify license file (`fortify.license`) and optionally you can use the Fortify License and Infrastructure Manager to manage concurrent licenses for Fortify Static Code Analyzer. With a LIM managed concurrent license, multiple installations of Fortify Static Code Analyzer can share a single license. For information about how to set up the LIM with licenses for Fortify Static Code Analyzer, see *OpenText™ Fortify License and Infrastructure Manager Installation and Usage Guide*. For more information about managing your LIM license from Fortify Static Code Analyzer, see "LIM license directives" on page 145.

# Renewing an expired license

The license for Fortify Static Code Analyzer expires annually. For information about how to obtain a Fortify license file, see the *Fortify Software System Requirements* document.

To update an expired license:

- Put the updated Fortify license file in the *<sca_install_dir>* folder.

To update an expired LIM managed concurrent license, see the *OpenText™ Fortify License and Infrastructure Manager Installation and Usage Guide*.

# Fortify Software Security Content

Fortify Static Code Analyzer uses a knowledge base of rules to enforce secure coding standards applicable to the codebase for static analysis. Fortify Software Security Center is required for both translation and analysis. You can download and install security content when you install Fortify Static Code Analyzer (see "Installing Fortify Static Code Analyzer" on page 27). Alternatively, you can download or import previously downloaded Fortify Software Security Content with the fortifyupdate command-line tool as a post-installation task (see "Manually installing Fortify Software Security Content" on page 32).

Fortify Software Security Content (security content) consists of Fortify Secure Coding Rulepacks and external metadata:

- Fortify Secure Coding Rulepacks describe general secure coding idioms for popular languages and public APIs
- External metadata includes mappings from the Fortify categories to alternative categories (such as CWE, OWASP Top 10, and PCI)

OpenText provides the ability to write custom rules that add to the functionality of Fortify Static Code Analyzer and the Fortify Secure Coding Rulepacks. For example, you might need to enforce proprietary security guidelines or analyze a project that uses third-party libraries or other pre-compiled binaries that are not already covered by the Fortify Secure Coding Rulepacks. You can also customize the external metadata to map Fortify issues to different taxonomies, such as internal application security standards or additional compliance obligations. For instructions on how to create your own custom rules or custom external metadata, see the *OpenText™ Fortify Static Code Analyzer Custom Rules Guide*.

OpenText recommends that you periodically update the security content. You can use fortifyupdate to obtain the latest security content. For more information, see "Updating Fortify Software Security Content" on page 149.

# Fortify ScanCentral SAST

You can use OpenText™ Fortify ScanCentral SAST to manage your resources by offloading the Fortify Static Code Analyzer analysis phase from build machines to a collection of machines provisioned for this purpose. For most languages, Fortify ScanCentral SAST can perform both the translation and the analysis (scan) phases. Users of Fortify Software Security Center can direct Fortify ScanCentral SAST to output the FPR file directly to the server. You have the option to install a Fortify ScanCentral SAST client when you install Fortify Static Code Analyzer.

You can analyze your code in one of two ways:

- Perform the translation phase on a local build machine and generate a mobile build session (MBS). Start the scan with Fortify ScanCentral SAST using the MBS file. In addition to freeing up the build machines, this process gives you the ability to expand the system by adding more resources as needed, without having to interrupt the build process. For more information about MBS, see "Mobile build sessions" on page 44.
- If your application is written in a language supported for Fortify ScanCentral SAST translation, you can offload the translation and analysis (scan) phase of the analysis to Fortify ScanCentral SAST. For information about the specific supported languages, see the *Fortify Software System Requirements* document.

For detailed information about how to configure and use Fortify ScanCentral SAST, see the *OpenText™ Fortify ScanCentral SAST Installation, Configuration, and Usage Guide*.

# Fortify Static Code Analyzer applications and tools

OpenText provides applications and tools (including Fortify Secure Code Plugins) that integrate with Fortify Static Code Analyzer, Fortify ScanCentral SAST, and Fortify Software Security Center. The following table describes the applications that are available for installation with the Fortify Applications and Tools installer. For instructions about installing the Fortify Applications and Tools, see the *OpenText™ Fortify Static Code Analyzer Applications and Tools Guide*.

| Application | Description |
|---|---|
| Fortify Audit Workbench | An application that provides a graphical user interface to help you organize, investigate, and prioritize analysis results so that developers can fix security flaws quickly. |
| OpenText™ Fortify Plugin for Eclipse | Adds the ability to scan and analyze the entire codebase of a project and apply software security rules that identify the vulnerabilities in your Java code from the Eclipse IDE. The results are displayed, along with descriptions of each of the security issues and suggestions for their elimination. |

| Application | Description |
|---|---|
| OpenText™ Fortify Analysis Plugin for IntelliJ IDEA and Android Studio | Adds the ability to run scans on the entire codebase of a project and apply software security rules that identify the vulnerabilities in your code from IntelliJ IDEA and Android Studio. |
| OpenText™ Fortify Extension for Visual Studio | Adds the ability to scan and locate security vulnerabilities in your solutions and projects and displays the scan results in Visual Studio. The results include a list of issues uncovered, descriptions of the type of vulnerability each issue represents, and suggestions on how to fix them. This extension also includes remediation functionality that works with audit results stored on a Fortify Software Security Center server. |
| OpenText™ Fortify Custom Rules Editor | An application to create and edit custom rules. |
| Fortify Scan Wizard | Provides a graphical user interface that enables you to prepare a script to scan your code (either locally or remotely using Fortify ScanCentral SAST) and then optionally upload the results to Fortify Software Security Center. |
| BIRTReportGenerator<br>ReportGenerator | Command-line tools to generate issue reports (BIRT) and legacy reports from FPR files. |

# Sample projects

OpenText provides sample projects available as a separate download in the `Fortify_SCA_Samples_<version>.zip` archive.

The ZIP file contains two directories: `basic` and `advanced`. Each code sample includes a `README.txt` file that provides instructions on how to scan the code with Fortify Static Code Analyzer and view the results in Fortify Audit Workbench.

The `basic` directory includes an assortment of simple language-specific code samples. The `advanced` directory includes more advanced samples.

# Related Documents

This topic describes documents that provide information about Fortify Software products.

**Note:** You can find the Fortify Product Documentation at https://www.microfocus.com/support/documentation. Most guides are available in both PDF and HTML formats.

## All Products

The following documents provide general information for all products. Unless otherwise noted, these documents are available on the Product Documentation website.

| Document / File Name | Description |
|---|---|
| *About Fortify Software Documentation*<br><br>About_Fortify_Docs_<*version*>.pdf | This paper provides information about how to access Fortify Software product documentation.<br><br>**Note:** This document is included only with the product download. |
| *Fortify Software System Requirements*<br><br>Fortify_Sys_Reqs_<*version*>.pdf | This document provides the details about the environments and products supported for this version of Fortify Software. |
| *Fortify Software Release Notes*<br><br>FortifySW_RN_<*version*>.pdf | This document provides an overview of the changes made to Fortify Software for this release and important information not included elsewhere in the product documentation. |
| *What's New in Fortify Software <version>*<br><br>Fortify_Whats_New_<*version*>.pdf | This document describes the new features in Fortify Software products. |

## Fortify ScanCentral SAST

The following document provides information about Fortify ScanCentral SAST. This document is available on the Product Documentation website at https://www.microfocus.com/documentation/fortify-software-security-center.

| Document / File Name | Description |
|---|---|
| *OpenText™ Fortify ScanCentral SAST Installation, Configuration, and Usage Guide* | This document provides information about how to install, configure, and use Fortify ScanCentral SAST to streamline the static code analysis process. It is written for anyone |

| Document / File Name | Description |
|---|---|
| SC_SAST_Guide_<*version*>.pdf | who intends to install, configure, or use Fortify ScanCentral SAST to offload the resource-intensive translation and scanning phases of their Fortify Static Code Analyzer process. |

## Fortify Software Security Center

The following document provides information about Fortify Software Security Center. This document is available on the Product Documentation website at https://www.microfocus.com/documentation/fortify-software-security-center.

| Document / File Name | Description |
|---|---|
| *OpenText™ Fortify Software Security Center User Guide*<br><br>SSC_Guide_<*version*>.pdf | This document provides Fortify Software Security Center users with detailed information about how to deploy and use Fortify Software Security Center. It provides all the information you need to acquire, install, configure, and use Fortify Software Security Center.<br><br>It is intended for use by system and instance administrators, database administrators (DBAs), enterprise security leads, development team managers, and developers. Fortify Software Security Center provides security team leads with a high-level overview of the history and status of a project. |

## Fortify Static Code Analyzer

The following documents provide information about Fortify Static Code Analyzer. Unless otherwise noted, these documents are available on the Product Documentation website at https://www.microfocus.com/documentation/fortify-static-code.

| Document / File Name | Description |
|---|---|
| *OpenText™ Fortify Static Code Analyzer User Guide*<br><br>SCA_Guide_<*version*>.pdf | This document describes how to install and use Fortify Static Code Analyzer to scan code on many of the major programming platforms. It is intended for people responsible for security audits and secure coding. |

| Document / File Name | Description |
|---|---|
| *OpenText™ Fortify Static Code Analyzer Custom Rules Guide*<br><br>SCA_Cust_Rules_Guide_<*version*>.zip | This document provides the information that you need to create custom rules for Fortify Static Code Analyzer. This guide includes examples that apply rule-writing concepts to real-world security issues.<br><br>**Note:** This document is included only with the product download. |
| *OpenText™ Fortify License and Infrastructure Manager Installation and Usage Guide*<br><br>LIM_Guide_<*version*>.pdf | This document describes how to install, configure, and use the Fortify License and Infrastructure Manager (LIM), which is available for installation on a local Windows server and as a container image on the Docker platform. |

## Fortify Static Code Analyzer Applications and Tools

The following documents provide information about Fortify Static Code Analyzer applications and tools. These documents are available on the Product Documentation website at https://www.microfocus.com/documentation/fortify-static-code-analyzer-and-tools.

| Document / File Name | Description |
|---|---|
| *OpenText™ Fortify Static Code Analyzer Applications and Tools Guide*<br><br>SCA_Apps_Tools_<*version*>.pdf | This document describes how to install Fortify Static Code Analyzer applications and tools. It provides an overview of the applications and command-line tools that enable you to scan your code with Fortify Static Code Analyzer, review analysis results, work with analysis results files, and more. |
| *OpenText™ Fortify Audit Workbench User Guide*<br><br>AWB_Guide_<*version*>.pdf | This document describes how to use Fortify Audit Workbench to scan software projects and audit analysis results. This guide also includes how to integrate with bug trackers, produce reports, and perform collaborative auditing. |
| *OpenText™ Fortify Plugin for Eclipse User Guide*<br><br>Eclipse_Plugin_Guide_<*version*>.pdf | This document provides information about how to install and use the Fortify Plugin for Eclipse to analyze and audit your code. |
| *OpenText™ Fortify Analysis Plugin for* | This document describes how to install and use the Fortify |

| Document / File Name | Description |
| --- | --- |
| *IntelliJ IDEA and Android Studio User Guide*<br><br>IntelliJ_AnalysisPlugin_Guide_<*version*>.pdf | Analysis Plugin for IntelliJ IDEA and Android Studio to analyze your code and optionally upload the results to Fortify Software Security Center. |
| *OpenText™ Fortify Extension for Visual Studio User Guide*<br><br>VS_Ext_Guide_<*version*>.pdf | This document provides information about how to install and use the Fortify Extension for Visual Studio to analyze, audit, and remediate your code to resolve security-related issues in solutions and projects. |

# Chapter 2: Installing Fortify Static Code Analyzer

This chapter describes how to install and uninstall Fortify Static Code Analyzer. This chapter also describes basic post-installation tasks. See the *Fortify Software System Requirements* document to be sure that your system meets the minimum requirements for each software component installation.

This section contains the following topics:



## About installing Fortify Static Code Analyzer

This section describes how to install Fortify Static Code Analyzer. Several command-line tools are installed automatically with Fortify Static Code Analyzer (see "Command-line tools" on page 148). You can optionally include a Fortify ScanCentral SAST client and the Fortify Software Security Center fortifyclient utility with the Fortify Static Code Analyzer installation. For information about Fortify ScanCentral SAST, see the *OpenText™ Fortify ScanCentral SAST Installation, Configuration, and Usage Guide*.

You must provide a Fortify license file and optionally LIM license pool credentials during the installation. The following table lists the different ways to install Fortify Static Code Analyzer.

| Installation method | Instructions |
|---|---|
| Perform the installation using a standard install wizard | "Installing Fortify Static Code Analyzer" on the next page |
| Perform the installation silently (unattended) | "Installing Fortify Static Code Analyzer silently" on page 30 |
| Perform a text-based installation on non-Windows systems | "Installing Fortify Static Code Analyzer in text-based mode on non-Windows platforms" on page 32 |
| Perform the installation using Docker | "Using Docker to install and run Fortify Static Code Analyzer" on page 33 |

For best performance, install Fortify Static Code Analyzer on the same local file system where the code that you want to scan resides.

**Note:** On non-Windows systems, you must install Fortify Static Code Analyzer as a user that has a home directory with write permission. Do not install Fortify Static Code Analyzer as a non-root user that has no home directory.

After you complete the installation, see "Post-installation tasks" on page 37 for additional steps you can perform to complete your system setup. You can also configure settings for runtime analysis, output, and performance of Fortify Static Code Analyzer by updating the installed configuration files. For information about the configuration options for Fortify Static Code Analyzer, see "Configuration options" on page 184.

## Installing Fortify Static Code Analyzer

To install Fortify Static Code Analyzer:

1. Run the installer file for your operating system to start the Fortify Static Code Analyzer Setup wizard:
   - Windows: `Fortify_SCA_<version>_windows_x64.exe`
   - Linux: `Fortify_SCA_<version>_linux_x64.run` or `Fortify_SCA_<version>_linux_arm64.run`
   - macOS: `Fortify_SCA_<version>_osx_x64.app.zip`
     Uncompress the ZIP file before you run the APP installer file.
   - AIX: `Fortify_SCA_<version>_aix.run`

   where *<version>* is the software release version, and then click **Next**.
2. Review and accept the license agreement, and then click **Next**.
3. (Optional) Select components to install, and then click **Next**.
4. If the installer detects that the system does not include the minimum software required to analyze some types of projects, a System Requirements page displays any missing requirements and which projects require them. Click **Next**.

   See the *Fortify Software System Requirements* document for all software requirements.
5. Choose where to install Fortify Static Code Analyzer, and then click **Next**.

   If you selected to include Fortify ScanCentral SAST client with the installation in step 3, then you must specify a location that does not include spaces in the path.

   **Important!** Do not install Fortify Static Code Analyzer in the same directory where Fortify Applications and Tools is installed.
6. Specify the path to the `fortify.license` file, and then click **Next**.
7. (Optional) On the LIM License page, select **Yes** to manage your concurrent licenses with Fortify License and Infrastructure Manager (LIM), and then click **Next**.

**Note:** When Fortify Static Code Analyzer performs a task that requires a license, the application will attempt to acquire a LIM lease from the license pool. If Fortify Static Code Analyzer fails to acquire a license due to a communication issue with the LIM server, it will use the Fortify license file. To change this behavior, use the `com.fortify.sca.lim.WaitForInitialLicense` in the `fortify-sca.properties` file (see "LIM license properties" on page 194).

a. Type the LIM API URL, the license pool name, and the pool password.
b. Click **Next**. The **LIM Proxy Settings** page opens.
c. If connection to the LIM server requires a proxy server, type the proxy host (hostname or IP address of your proxy server) and optionally a port number.
d. Click **Next**.

8. To update the security content for your installation:

**Note:** For deployment environments that do not have access to the Internet during installation, you can update the security content using the fortifyupdate command-line tool. See "Manually installing Fortify Software Security Content" on page 32.

a. Type the web address of the update server. To use the Fortify Rulepack update server for security content updates, keep the web address `https://update.fortify.com`. You can also use Fortify Software Security Center as the update server.
b. (Optional) If connection to the update server requires a proxy server, type the proxy host and port number.
c. If you want to update the security content manually, clear the **Update security content after installation** check box.
d. Click **Next**.

9. Specify if you want to migrate from a previous installation on your system.

Migrating from a previous installation preserves Fortify Static Code Analyzer artifact files. For more information, see "About upgrading Fortify Static Code Analyzer" on page 35.

**Note:** You can also migrate artifacts using the `scapostinstall` command-line tool. For information on how to use the post-install tool to migrate from a previous installation, see "Migrating properties files" on page 38.

To migrate artifacts from a previous installation:

a. In the Fortify Static Code Analyzer Migration page, select **Yes**, and then click **Next**.
b. Specify the location of the existing installation on your system, and then click **Next**.

To skip migration of artifacts from a previous release, leave the migration selection set to **No**, and then click **Next**.

10. Click **Next** on the Ready to Install page to install Fortify Static Code Analyzer, any selected components, and Fortify security content.

If you selected to update security content, the Security Content Update Result window displays the security content update results.

11. Click **Finish** to close the Setup wizard.

# Installing Fortify Static Code Analyzer silently

A silent installation enables you to complete the installation without any user prompts. To install silently, you need to create an option file to provide the necessary information to the installer. Using the silent installation, you can replicate the installation parameters on multiple machines.

**Important!** Do not install Fortify Static Code Analyzer in the same directory where Fortify Applications and Tools is installed.

When you install Fortify Static Code Analyzer silently, the installer does not download the Fortify Software Security Center by default. You can enable download of the Fortify security content in the options file or you can install the Fortify security content manually (see "Manually installing Fortify Software Security Content" on page 32).

To install Fortify Static Code Analyzer silently:

1. Create an options file.
   a. Create a text file that contains the following line:

      ```
      fortify_license_path=<license_file_location>
      ```

      where *<license_file_location>* is the full path to your `fortify.license` file.
   b. To use a LIM license server, add the following lines with your LIM license pool credentials to the options file:

      ```
      lim_url=<lim_url>
      lim_pool_name=<license_pool_name>
      lim_pool_password=<license_pool_pwd>
      ```

   c. To use a location for Fortify Software Security Content updates that is different than the default of `https://update.fortify.com`, add the following line:

      ```
      update_server=<update_server_url>
      ```

   d. If you require a proxy server for the Fortify security content download, add the following lines:

      ```
      update_proxy_server=<proxy_server>
      update_proxy_port=<port_number>
      ```

   e. To enable download of Fortify security content, add the following line:

      ```
      update_security_content=1
      ```

   f. Add more installation instructions, as needed, to the options file.

      To obtain a list of installation options that you can add to your options file, open a command prompt, and then type the installer file name and the `--help` option. This command displays

each available command-line option preceded with a double dash and the available parameters enclosed in angle brackets. For example, if you want to see the progress of the install displayed at the command line, add `unattendedmodeui=minimal` to your options file.

**Notes:**

- The command-line options are case-sensitive.
- The installation options are not the same on all supported operating systems. Run the installer with `--help` to see the options available for your operating system.

The following example Windows options file specifies the location of the license file, the location of a Fortify Software Security Center server and proxy information to obtain Fortify Software Security Content, a request to migrate from a previous release, and the location of the Fortify Static Code Analyzer installation directory:

```
fortify_license_path=C:\Users\admin\Desktop\fortify.license
update_server=https://my_ssc_host:8080/ssc
update_proxy_server=webproxy.abc.company.com
update_proxy_port=8080
migrate_sca=1
install_dir=C:\Fortify
```

The following options file example is for Linux and macOS:

```
fortify_license_path=/opt/Fortify/fortify.license
update_server=https://my_ssc_host:8080/ssc
update_proxy_server=webproxy.abc.company.com
update_proxy_port=8080
migrate_sca=1
install_dir=/opt/Fortify
```

2. Save the options file.
3. Run the silent install command for your operating system.

**Note:** You might need to run the command prompt as an administrator before you run the installer.

| | |
|---|---|
| **Windows** | `Fortify_SCA_<version>_windows_x64.exe --mode unattended --optionfile <full_path_to_options_file>` |
| **Linux** | `./Fortify_SCA_<version>_linux_x64.run --mode unattended --optionfile <full_path_to_options_file>` |

| | |
|---|---|
| **macOS** | You must uncompress the ZIP file before you run the command.<br>`Fortify_SCA_<version>_osx_x64.app/Contents/MacOS/installbuilder.sh --mode unattended --optionfile <full_path_to_options_file>` |
| **AIX** | `./Fortify_SCA_<version>_aix.run --mode unattended --optionfile <full_path_to_options_file>` |

The installer creates an installer log file when the installation is complete. This log file is in the following location, which depends on your operating system.

| | |
|---|---|
| **Windows** | `C:\Users\<username>\AppData\Local\Temp\FortifySCA-<version>-install.log` |
| **Non-Windows** | `/tmp/FortifySCA-<version>-install.log` |

## Installing Fortify Static Code Analyzer in text-based mode on non-Windows platforms

You perform a text-based installation on the command line. During the installation, you are prompted for information required to complete the installation. Text-based installations are not supported on Windows systems.

**Important!** Do not install Fortify Static Code Analyzer in the same directory where Fortify Applications and Tools is installed.

To perform a text-based installation of Fortify Static Code Analyzer, run the text-based install command for your operating system as listed in the following table.

| | |
|---|---|
| **Linux** | `./Fortify_SCA_<version>_linux_x64.run --mode text` |
| **macOS** | You must uncompress the provided ZIP file before you run the command.<br>`Fortify_SCA_<version>_osx_x64.app/Contents/MacOS/installbuilder.sh --mode text` |
| **AIX** | `./Fortify_SCA_<version>_aix.run --mode text` |

## Manually installing Fortify Software Security Content

You can install Fortify Software Security Content (Fortify Secure Coding Rulepacks and metadata) automatically during the installation. However, you can also download Fortify Software Security Content from the Fortify Rulepack update server, and then use the fortifyupdate command-

line tool to install it. This option is provided for deployment environments that do not have access to the Internet during installation.

Use fortifyupdate to install Fortify security content from either a remote server or a locally downloaded file.

To install security content:

1. Open a command window.
2. Navigate to the *<sca_install_dir>*/bin directory.
3. At the command prompt, type `fortifyupdate`.

   If you have previously downloaded the Fortify Software Security Content from the Fortify Rulepack update server, run `fortifyupdate` with the `-import` option and the path to the directory where you downloaded the ZIP file.

You can also use this same tool to update your Fortify Software Security Content. For more information about the fortifyupdate command-line tool, see "Updating Fortify Software Security Content" on page 149.

# Using Docker to install and run Fortify Static Code Analyzer

You can install Fortify Static Code Analyzer in a Docker image and then run Fortify Static Code Analyzer as a Docker container.

**Note:** You can only run Fortify Static Code Analyzer in Docker on supported Linux platforms.

## Creating a Dockerfile to install Fortify Static Code Analyzer

This topic describes how to create a Dockerfile to install Fortify Static Code Analyzer in a Docker image.

The Dockerfile must include the following instructions:

1. Set a Linux system to use for the base image.

   **Note:** If you intend to use build tools when you run Fortify Static Code Analyzer, make sure that the required build tools are installed in the image. For information about using the supported build tools, see "Build integration" on page 121.

2. Copy the Fortify Static Code Analyzer installer, the Fortify license file, and installation options file to the Docker image using the COPY instruction.

   For instructions on how to create an installation options file, see "Installing Fortify Static Code Analyzer silently" on page 30.
3. Run the Fortify Static Code Analyzer installer using the RUN instruction.

You must run the installer in unattended mode. For more information, see "Installing Fortify Static Code Analyzer silently" on page 30.

4. Run fortifyupdate to install the Fortify security content using the RUN instruction.

   **Important!** Fortify Static Code Analyzer requires installation of the Fortify Software Security Content to perform analysis of projects. The following example installs Fortify Software Security Content from a previously downloaded local file during the build of the image. For more information about downloading and installing Fortify Software Security Content using the fortifyupdate tool, see "Manually installing Fortify Software Security Content" on page 32.

5. To configure the image so you can run Fortify Static Code Analyzer, set the entry point to the location of the installed sourceanalyzer executable using the ENTRYPOINT instruction.

   The default sourceanalyzer installation path is: `/opt/Fortify/Fortify_SCA_<version>/bin/sourceanalyzer`.

The following is an example of a Dockerfile to install Fortify Static Code Analyzer:

```
FROM ubuntu:18.04
WORKDIR /app
ENV APP_HOME="/app"
ENV RULEPACK="MyRulepack.zip"

COPY fortify.license ${APP_HOME}
COPY Fortify_SCA_24.4.0_linux_x64.run ${APP_HOME}
COPY optionFile ${APP_HOME}
COPY ${RULEPACK} ${APP_HOME}

RUN ./Fortify_SCA_24.4.0_linux_x64.run --mode unattended \
  --optionfile "${APP_HOME}/optionFile" && \
  /opt/Fortify/Fortify_SCA_24.4.0/bin/fortifyupdate -import ${RULEPACK} && \
  rm Fortify_SCA_24.4.0_linux_x64.run optionFile

ENTRYPOINT ["/opt/Fortify/Fortify_SCA_24.4.0/bin/sourceanalyzer"]
```

To create the docker image using the Dockerfile from the current directory, you must use the docker build command. For example:

```
docker buildx build -f <docker_file> -t <image_name> "."
```

## Running the container

This topic describes how to run the Fortify Static Code Analyzer image as a container and provides example Docker run commands for translation and scan.

**Note:** When you run Fortify Static Code Analyzer in a container and especially if you also leverage runtime container protections, make sure that Fortify Static Code Analyzer has the appropriate permission to run build commands (for example, javac).

To run the Fortify Static Code Analyzer image as a container, you must mount two directories from the host file system to the container:

- The directory that contains the source files you want to analyze.
- A temporary directory to store the Fortify Static Code Analyzer build session between the translate and scan phases and to share the output files (logs and FPR file) with the host.

  Specify this directory using the `–project-root` command-line option in both the Fortify Static Code Analyzer translate and scan commands.

The following example commands mount the input directory `/sources` in `/src` and the temporary directory in `/scratch_docker`. The image name in the example is `fortify-sca`.

### Example Docker run commands for translation and scan

The following example mounts the temporary directory and the sources directory, and then runs Fortify Static Code Analyzer from the container for the translation phase:

```
docker run -v /scratch_local/:/scratch_docker -v /sources/:/src
-it fortify-sca –b MyProject -project-root /scratch_docker  [<sca_options>]
/src
```

The following example mounts the temporary directory, and then runs Fortify Static Code Analyzer from the container for the analysis phase:

```
docker run -v /scratch_local/:/scratch_docker
-it fortify-sca –b MyProject -project-root /scratch_docker –scan  [<sca_
options>] –f /scratch_docker/MyResults.fpr
```

The `MyResults.fpr` output file is created in the host's `/scratch_local` directory.

# About upgrading Fortify Static Code Analyzer

To upgrade Fortify Static Code Analyzer, install the new version in a different location than where your current version is installed and choose to migrate settings from the previous installation. This migration preserves and updates the Fortify Static Code Analyzer artifact files located in the *<sca_install_dir>*`/Core/config` directory.

If you choose not to migrate any settings from a previous release, OpenText recommends that you save a backup of the following data if it has been modified:

- *<sca_install_dir>*`/Core/config/rules` folder
- *<sca_install_dir>*`/Core/config/customrules` folder
- *<sca_install_dir>*`/Core/config/ExternalMetadata` folder
- *<sca_install_dir>*`/Core/config/CustomExternalMetadata` folder
- *<sca_install_dir>*`/Core/config/server.properties` file
- *<sca_install_dir>*`/Core/config/scales` folder

After you install the new version, you can uninstall the previous version. For more information, see "About uninstalling Fortify Static Code Analyzer" below.

**Note:** You can leave the previous version installed. If you have multiple versions installed on the same system, the most recently installed version is used when you run the command from the command line.

# About uninstalling Fortify Static Code Analyzer

This section describes how to uninstall Fortify Static Code Analyzer. You can use the standard install wizard, or you can silently install Fortify Static Code Analyzer. You can also perform a text-based uninstallation on non-Windows systems.

## Uninstalling Fortify Static Code Analyzer

To uninstall Fortify Static Code Analyzer:

1. Run the uninstall command located in the *<sca_install_dir>* for your operating system:

| OS | Uninstall command |
|---|---|
| Windows | `Uninstall_FortifySCA.exe`<br>Alternatively, you can uninstall the application from the Windows interface. See the Microsoft Windows documentation for instructions. |
| Linux<br>AIX | `./Uninstall_FortifySCA` |
| macOS | `Uninstall_FortifySCA.app` |

2. You are prompted to indicate whether to remove the entire application or individual components. Make your selection, and then click **Next**.

   If you are uninstalling specific components, select the components to remove on the Select Components to Uninstall page, and then click **Next**.
3. You are prompted to indicate whether to remove all application settings. Do one of the following:
   - Click **Yes** to remove the application settings for the components installed with the version of Fortify Static Code Analyzer that you are uninstalling.

     The Fortify Static Code Analyzer (`sca<version>`) folder is not removed.
   - Click **No** to retain the application settings on your system.

## Uninstalling Fortify Static Code Analyzer silently

To uninstall Fortify Static Code Analyzer silently:

1. Navigate to the installation directory.
2. Type one of the following commands based on your operating system:

| | |
|---|---|
| **Windows** | `Uninstall_FortifySCA_<version>.exe --mode unattended` |
| **Linux**<br>**AIX** | `./Uninstall_FortifySCA_<version> --mode unattended` |
| **macOS** | `Uninstall_FortifySCA_<version>.app/Contents/MacOS/installbuilder.sh --mode unattended` |

**Note:** For Windows, Linux, and macOS, the uninstaller removes the application settings for the components installed with the version of Fortify Static Code Analyzer that you are uninstalling.

## Uninstalling Fortify Static Code Analyzer in text-based mode on non-Windows platforms

To uninstall Fortify Static Code Analyzer in text-based mode, run the text-based install command for your operating system, as follows:

1. Navigate to the installation directory.
2. Type one of the following commands based on your operating system:

| | |
|---|---|
| **Linux**<br>**AIX** | `./Uninstall_Fortify_SCA --mode text` |
| **macOS** | `Uninstall_Fortify_SCA.app/Contents/MacOS/installbuilder.sh --mode text` |

# Post-installation tasks

Post-installation tasks prepare you to start using Fortify Static Code Analyzer.

## Running the post-install tool

You can use the post-install tool to migrate properties files from a previous version of Fortify Static Code Analyzer, configure Fortify security content updates, and configure settings to connect to

Fortify Software Security Center.

To run the Fortify Static Code Analyzer post-install tool:

1. Navigate to the *<sca_install_dir>*/bin directory from the command line.
2. At the command prompt, type scapostinstall.
3. Type one of the following:
   - To display settings, type s.
   - To return to the previous prompt, type r.
   - To exit the tool, type q.

## Migrating properties files

To migrate properties files from a previous version of Fortify Static Code Analyzer to the current version of Fortify Static Code Analyzer installed on your system:

1. Navigate to the *<sca_install_dir>*/bin directory from the command line.
2. At the command prompt, type scapostinstall.
3. Type 1 to select Migration.
4. Type 1 to select Static Code Analyzer Migration.
5. Type 1 to select Migrate from an existing Fortify installation.
6. Type 1 to select Set previous Fortify installation directory.
7. Type the previous install directory.
8. Type s to confirm the settings.
9. Type 2 to perform the migration.
10. Type y to confirm.

## Specifying a locale

English is the default locale for a Fortify Static Code Analyzer installation.

To change the locale for your Fortify Static Code Analyzer installation:

1. Navigate to the bin directory from the command line.
2. At the command prompt, type scapostinstall.
3. Type 2 to select Settings.
4. Type 1 to select General.
5. Type 1 to select Locale.
6. Type one of the following locale codes:
   - en (English)
   - es (Spanish)

- `ja` (Japanese)
- `ko` (Korean)
- `pt_BR` (Brazilian Portuguese)
- `zh_CN` (Simplified Chinese)
- `zh_TW` (Traditional Chinese)

## Configuring Fortify Security Content updates

Specify how you want to obtain Fortify security content. You must also specify proxy information if it is required to reach the server.

To specify settings for Fortify Security Content updates:

1. Navigate to the `bin` directory from the command line.
2. At the command prompt, type `scapostinstall`.
3. Type `2` to select `Settings`.
4. Type `2` to select `Fortify Update`.
5. To change the Fortify Rulepack update server URL, type `1`, and then type the URL.
   The default Fortify Rulepack update server URL is `https://update.fortify.com`.
6. To specify a proxy for Fortify security content updates, do the following:
   a. Type `2` to select `Proxy Server`, and then type the name of the proxy server.
   Exclude the protocol and port number (for example, some.secureproxy.com).
   b. Type `3` to select `Proxy Server Port`, and then type the proxy server port number.
   c. (Optional) You can also specify a proxy server user name (option `4`) and password (option `5`).

## Configuring the connection to Fortify Software Security Center

Specify how to connect to Fortify Software Security Center. If your network uses a proxy server to reach the Fortify Software Security Center server, you must specify the proxy information.

To specify settings for connecting to Fortify Software Security Center:

1. Navigate to the `bin` directory from the command line.
2. At the command prompt, type `scapostinstall`.
3. Type `2` to select `Settings`.
4. Type `3` to select `Software Security Center Settings`.
5. Type `1` to select `Server URL`, and then type the Fortify Software Security Center server URL.

6. To specify proxy settings for the connection, do the following:
   a. Type `2` to select `Proxy Server`, and then type the name of the proxy server. Exclude the protocol and port number (for example, some.secureproxy.com).
   b. Type `3` to select `Proxy Server Port`, and then type the proxy server port number.
   c. To specify a proxy server user name and password, use option `4` for the username and option `5` for the password.
7. (Optional) You can also specify the following:
   - Whether to update Fortify Software Security Content from your Fortify Software Security Center server (option `6`)
   - The Fortify Software Security Center user name (option `7`)

## Removing proxy server settings

If you previously specified proxy server settings for the Fortify Rulepack update server or Fortify Software Security Center and it is no longer required, you can remove these settings.

To remove the proxy settings for obtaining Fortify Software Security Content updates or connecting to Fortify Software Security Center:

1. Navigate to the `bin` directory from the command line.
2. At the command prompt, type `scapostinstall`.
3. Type `2` to select `Settings`.
4. Type `2` to select `Fortify Update` or type `3` to select `Software Security Center Settings`.
5. Type the number that corresponds to the proxy setting you want to remove, and then type a minus sign (`-`) to remove the setting.
6. Repeat step 5 for each proxy setting you want to remove.

## Adding trusted certificates

Connection from Fortify Static Code Analyzer to other Fortify Software products and external systems might require communication over HTTPS. Some examples include:

- Fortify Static Code Analyzer by default requires an HTTPS connection to communicate with the LIM server for license management.

  The property `com.fortify.sca.lim.RequireTrustedSSLCert` determines whether the connection with the LIM server requires a trusted SSL certificate. For more information about this property, see "LIM license properties" on page 194.
- The fortifyupdate command-line tool uses an HTTPS connection either automatically during a Windows system installation or manually (see "Manually installing Fortify Software Security Content" on page 32) to update Fortify security content.
- Fortify Static Code Analyzer configured as a Fortify ScanCentral SAST sensor uses an HTTPS connection to communicate with the Controller.

When using HTTPS, Fortify Static Code Analyzer and its applications will by default apply standard checks to the presented SSL server certificate, including a check to determine if the certificate is trusted. If your organization runs its own certificate authority (CA) and Fortify Static Code Analyzer needs to trust connections where the server presents a certificate issued by this CA, you must configure Fortify Static Code Analyzer to trust the CA. Otherwise, the use of HTTPS connections might fail.

You must add the trusted certificate of the CA to the Fortify Static Code Analyzer keystore. The Fortify Static Code Analyzer keystore is in the *<sca_install_dir>*/jre/lib/security/cacerts file. You can use the keytool command to add the trusted certificate to the keystore.

To add a trusted certificate to the Fortify Static Code Analyzer keystore:

1. Open a command prompt, and then run the following command:

   ```
   <sca_install_dir>/jre/bin/keytool -importcert -alias <alias_name> -
   cacerts -file <cert_file>
   ```

   where:

   - *<alias_name>* is a unique name for the certificate you are adding.
   - *<cert_file>* is the name of the file that contains the trusted root certificate in PEM or DER format.

2. Enter the keystore password.

   **Note:** The default password is `changeit`.

3. When prompted to trust this certificate, select **yes**.

# Chapter 3: Analysis process overview

This section contains the following topics:



## Analysis process

There are four distinct phases that make up the analysis process:

1. **Build Integration**—Choose whether to integrate Fortify Static Code Analyzer into your build tool. For descriptions of build integration options, see "Integrating the analysis into a build" on page 121.
2. **Translation**—Gathers source code using a series of commands and translates it into an intermediate format associated with a build ID. The build ID is usually the name of the project you are translating. For more information, see "Translation phase" on the next page.
3. **Analysis**—Scans source files identified in the translation phase and generates an analysis result file (typically in the Fortify Project Results (FPR) format). FPR files have the `.fpr` extension. For more information, see "Analysis phase" on page 46.
4. **Verification of translation and analysis**—Verifies that the source files were scanned using the correct Rulepacks and that no errors were reported. For more information, see "Translation and analysis phase verification" on page 48.

OpenText recommends that you perform translation and analysis commands from a user account with least privilege access. OpenText does not recommend that you run Fortify Static Code Analyzer as a root user or translate a project that requires root access, because it might not work properly.

The following is the fundamental sequence of commands to translate and analyze code:

1. Remove all existing Fortify Static Code Analyzer temporary files for the specified build ID.

   ```
   sourceanalyzer -b MyProject -clean
   ```

   Always begin an analysis with this step to analyze a project with a previously used build ID.
2. Translate the project code.

   ```
   sourceanalyzer -b MyProject <files_to_analyze>
   ```

For most languages, this step can consist of multiple calls to sourceanalyzer with the same build ID. For more details, see "Translation phase" below.

3. Analyze the project code and save the results in a Fortify Project Results(FPR) file.

```
sourceanalyzer -b MyProject -scan -f MyResults.fpr
```

For more information, see "Analysis phase" on page 46.

## Parallel processing

Fortify Static Code Analyzer runs in parallel analysis mode to reduce the scan time of large projects. This takes advantage of all CPU cores available on your system. When you run Fortify Static Code Analyzer, avoid running other CPU intensive processes during the Fortify Static Code Analyzer execution because it expects to have the full resources of your hardware available for the scan.

# Translation phase

To successfully translate a project that is normally compiled, make sure that you have any dependencies required to build the project available. For languages that have any specific requirements, see the chapters for the specific source code type.

The basic command-line syntax to perform the first step of the analysis process, file translation, is:

```
sourceanalyzer -b <build_id> ... <files>
```

or

```
sourceanalyzer -b <build_id> ... <compiler_command>
```

The translation phase consists of one or more invocations of Fortify Static Code Analyzer using the `sourceanalyzer` command. Fortify Static Code Analyzer uses a build ID (`-b` option) to tie the invocations together. Subsequent invocations of `sourceanalyzer` add any newly specified source or configuration files to the file list associated with the build ID.

After translation, you can use the `-show-build-warnings` directive to list any warnings and errors that occurred in the translation phase:

```
sourceanalyzer -b <build_id> -show-build-warnings
```

To view the files associated with a build ID, use the `-show-files` directive:

```
sourceanalyzer -b <build_id> -show-files
```

## Special considerations for the translation phase

Consider the following special considerations before you perform the translation phase on your project:

- When you translate dynamic languages (JavaScript/TypeScript, PHP, Python, and Ruby), you must specify all source files together in one invocation. Fortify Static Code Analyzer does not support adding new files to the file list associated with the build ID on subsequent invocations.
- Generated code is automatically generated by a script or a tool such as a parsing tool. This code can be optimized, minimized, or large and complex. Therefore, OpenText recommends that you exclude it from translation because it would be challenging to fix any vulnerabilities Fortify Static Code Analyzer might report in this code. Use the `-exclude` command-line option to exclude this type of code from translation.

The following chapters describe how to translate different types of source code:

- "Translating Java code" on page 50
- "Translating Kotlin code" on page 58
- "Translating Visual Studio projects" on page 62
- "Translating C and C++ code" on page 68
- "Translating JavaScript and TypeScript code" on page 71
- "Translating Python code" on page 77
- "Translating code for mobile platforms" on page 82
- "Translating Go code" on page 85
- "Translating Dart and Flutter code" on page 89
- "Translating Ruby code" on page 91
- "Translating COBOL code" on page 93
- "Translating Salesforce Apex and Visualforce code" on page 98
- "Translating other languages and configurations" on page 100

# Mobile build sessions

With a Fortify Static Code Analyzer mobile build session (MBS), you can translate a project on one machine and scan it on another. A mobile build session (MBS file) includes all the files needed for the analysis phase. To improve scan time, you can perform the translation on the build computer, and then move the build session (MBS file) to a better equipped computer for the scan. The developers can run translations on their own computers and use only one powerful computer to run large scans.

To include regular expression analysis (see "Regular expression analysis" on page 47) for your project, OpenText recommends that you include `-Dcom.fortify.sca.MobileBuildSessions=true` in the command to create the MBS file so that the source code is included in the MBS. This enables regular expression analysis to work for the scan on a different computer.

You must have the same version of Fortify Software Security Content (Rulepacks) installed on both the system where you perform the translation and the system where you perform the analysis.

## Mobile build session version compatibility

The Fortify Static Code Analyzer version on the translate machine must be compatible with the Fortify Static Code Analyzer version on the analysis machine. The version number format is *<major>.<minor>.<patch>.<build_number>* (for example, 24.4.0.0140). The *<major>* and *<minor>* portions of the Fortify Static Code Analyzer version numbers on both the translation and the analysis machines must match. For example, 24.4.0 and 24.4.x are compatible. To determine the Fortify Static Code Analyzer version number, type `sourceanalyzer -v` on the command line.

You can obtain the build ID and the Fortify Static Code Analyzer version from an MBS file with the following command:

```
sourceanalyzer -import-build-session <file>.mbs
-Dcom.fortify.sca.ExtractMobileInfo=true
```

## Creating a mobile build session

On the machine where you performed the translation, issue the following command to generate a mobile build session:

```
sourceanalyzer -b <build_id> -export-build-session <file>.mbs
```

where *<file>*`.mbs` is the file name you provide for the Fortify Static Code Analyzer mobile build session.

To include source code in the MBS file, run the following command:

```
sourceanalyzer -b <build_id> -Dcom.fortify.sca.MobileBuildSessions=true -
export-build-session <file>.mbs
```

## Importing a mobile build session

After you move the *<file>*`.mbs` file to the machine where you want to perform the scan, import the mobile build session into the Fortify Static Code Analyzer project root directory.

To import the mobile build session, type the following command:

```
sourceanalyzer -import-build-session <file>.mbs
```

After you import your Fortify Static Code Analyzer mobile build session, you can proceed to the analysis phase. Perform a scan with the same build ID that was used in the translation.

You cannot merge multiple mobile build sessions into a single MBS file. Each exported build session must have a unique build ID. However, after all the build IDs are imported on the same Fortify Static

Code Analyzer installation, you can scan multiple build IDs in one scan with the `-b` option (see "Analysis phase" below).

# Analysis phase

The analysis phase scans the intermediate files created during translation and creates the vulnerability results file (FPR).

The analysis phase consists of one invocation of `sourceanalyzer`. You specify the build ID and include the `-scan` directive with any other required analysis or output options (see "Analysis options" on page 136 and "Output options" on page 139).

The following example shows the command-line syntax to perform the analysis phase and save the results in an FPR file:

```
sourceanalyzer -b MyProject -scan -f MyResults.fpr
```

**Note:** By default, Fortify Static Code Analyzer includes the source code in the FPR file.

To combine multiple builds into a single scan command, add the additional builds to the command line:

```
sourceanalyzer -b MyProject1 -b MyProject2 -b MyProject3 -scan -f
MyResults.fpr
```

## Applying a scan policy to the analysis

For the analysis (scan) phase, you can specify a scan policy to help you identify the most serious vulnerabilities so you can remediate the code quickly. The following table describes the three available scan policies.

| Policy name | Description |
|---|---|
| security | This is the default scan policy, which excludes issues related to code quality and dataflow that involves typically trusted locations from the analysis results. Use this policy to focus code remediation on the security issues. |
| devops | This scan policy excludes issues that are also excluded by the security policy and reduces the number of reported low-priority issues. Use this scan policy when scan speed is a priority, and developers review results directly (without any intermediate auditing). Issues that remain after you apply this scan policy are probably serious security issues that require remediation.<br><br>**Note:** This devops scan policy does not automatically include any |

| Policy name | Description |
|---|---|
| | customization made to the local security scan policy. |
| classic | This scan policy does not exclude any issues. Use this scan policy to see all issues, including those that are code quality related. |

To specify a scan policy for your analysis, include the `-scan-policy` (or `-sc`) option in the analysis phase as shown in the following example:

```
sourceanalyzer -b MyProject -scan -scan-policy devops -f MyResults.fpr
```

Alternatively, you can specify the scan policy with the `com.fortify.sca.ScanPolicy` property in the `fortify-sca.properties` file. For example:

```
com.fortify.sca.ScanPolicy=devops
```

**Note:** You can apply a filter file (see "Excluding issues with filter files" on page 178) with a scan policy setting for an analysis. In this case, Fortify Static Code Analyzer applies both the scan policy and the filter file to the analysis.

The policy files are in the *<sca_install_dir>*`/Core/config/scales` directory. There is one file for each scan policy. You can change the settings in these policy files to customize your scan policies. For information about the syntax used for the policy files, see "Excluding issues with filter files" on page 178.

**See also**

"Translation and analysis phase properties" on page 186

## Regular expression analysis

Regular expression (regex) analysis provides the ability for using regular expression rules to detect vulnerabilities in both file content and file names. This analysis can detect vulnerable secrets such as passwords, keys, and credentials in project files. The Configuration Analyzer includes the regex analysis capability.

**Important!** Regex analysis is language agnostic and therefore it might detect vulnerabilities in file types that Fortify Static Code Analyzer does not officially support.

Regex analysis recursively examines all file paths and path patterns included in the translation phase. Every file, for each directory found is analyzed unless it is specifically excluded from the translation. To manage the files that are included in regex analysis, the following options are available:

- Exclude any file or directory with the `-exclude` option in the translation phase.
  For more information about this option, see "Translation options" on page 134.

- By default, regex analysis excludes all detectible binary files. To include binary files in the analysis, add the following property to the `fortify-sca.properties` file (or include this property on the command line using the `-D` option):

```
com.fortify.sca.regex.ExcludeBinaries = false
```

- By default, regex analysis excludes files larger than 10 MB to ensure that the scan time is acceptable. You can change the maximum file size (in megabytes) with the following property:

```
com.fortify.sca.regex.MaxSize = <max_file_size_mb>
```

To disable regex analysis, add the following property to the `fortify-sca.properties` file or include it on the command line:

```
com.fortify.sca.regex.Enable = false
```

**See also**

"Mobile build sessions" on page 44

"Regex analysis properties " on page 193

## Higher-Order Analysis

Higher-Order Analysis (HOA) improves the ability to track dataflow through higher-order code. Higher-order code manipulates functions as values, generates them with anonymous function expressions (lambda expressions), passes them as arguments, returns them as values, and assigns them to variables and to fields of objects. These code patterns are common in modern dynamic languages such as JavaScript, TypeScript, Python, Ruby, and Swift.

By default, Fortify Static Code Analyzer performs Higher-Order Analysis when you scan JavaScript, TypeScript, Python, Ruby, and Swift code. For a description of the Higher-Order Analysis properties, see "Translation and analysis phase properties" on page 186.

# Translation and analysis phase verification

Fortify Audit Workbench certification indicates whether the code analysis from a scan is complete and valid. The project summary in Fortify Audit Workbench shows the following specific information about Fortify Static Code Analyzer scanned code:

- List of files scanned, with file sizes and timestamps
- Java class path used for the translation (if applicable)
- Rulepacks used for the analysis
- Fortify Static Code Analyzer runtime settings and command-line options
- Any errors or warnings encountered during translation or analysis
- Machine and platform information

**Note:** To obtain result certification, you must specify FPR for the analysis phase output format.

To view result certification information, open the FPR file in Fortify Audit Workbench and select **Tools > Project Summary > Certification**. For more information, see the *OpenText™ Fortify Audit Workbench User Guide*.

# Chapter 4: Translating Java code

This section describes how to translate Java code.

Fortify Static Code Analyzer supports analysis of Jakarta EE (Java EE) applications (including JSP files, configuration files, and deployment descriptors), Java Bytecode, and Java code with Lombok annotations.

This section contains the following topics:



## Java translation command-line syntax

To translate Java code, all types defined in a library that are referenced in the code must have a corresponding definition in the source code, a class file, or a JAR file. Include all source files on the Fortify Static Code Analyzer command line.

If your project contains Java code that refers to Kotlin code, make sure that the Java and Kotlin code are translated in the same Fortify Static Code Analyzer instance so that the Java references to Kotlin elements are resolved correctly. Kotlin to Java interoperability does not support Kotlin files provided by the `-sourcepath` option. For more information about the `-sourcepath` option, see "Java command-line options" on the next page

The basic command-line syntax to translate Java code is shown in the following example:

```
sourceanalyzer -b <build_id> -cp <classpath> <files>
```

With Java code, Fortify Static Code Analyzer can either:

- Emulate the compiler, which might be convenient for build integration
- Accept source files directly, which is convenient for command-line scans

For information about how to integrate Fortify Static Code Analyzer with Ant, see "Integrating with Ant" on page 123.

To have Fortify Static Code Analyzer emulate the compiler, type:

```
sourceanalyzer -b <build_id> javac [<translation_options>]
```

To pass files directly to Fortify Static Code Analyzer, type:

```
sourceanalyzer -b <build_id> -cp <classpath> [<translation_options>]
<files> | <file_specifiers>
```

where:

- *<translation_options>* are options passed to the compiler.
- `-cp` *<classpath>* specifies the class path to use for the Java source code.
  Include all JAR dependencies normally used to build the project. Separate multiple paths with semicolons (Windows) or colons (non-Windows).

  Similar to javac, Fortify Static Code Analyzer loads classes in the order they appear in the class path. If there are multiple classes with the same name in the list, Fortify Static Code Analyzer uses the first loaded class. In the following example, if both `A.jar` and `B.jar` include a class called `MyData.class`, Fortify Static Code Analyzer uses the `MyData.class` from `A.jar`.

  ```
  sourceanalyzer -cp A.jar:B.jar myfile.java
  ```

  OpenText strongly recommends that you avoid using duplicate classes with the `-cp` option.
  Fortify Static Code Analyzer loads JAR files in the following order:

  a. From the `-cp` option
  b. From `jre/lib`
  c. From *<sca_install_dir>*`/Core/default_jars`

  This enables you to override a library class by including the similarly-named class in a JAR specified with the `-cp` option.

For descriptions of all the available Java-specific command-line options, see "Java command-line options" below.

## Java command-line options

The following table describes the Java command-line options (for Java SE and Jakarta EE).

| Java or Jakarta EE option | Description |
|---|---|
| `-appserver weblogic \| websphere` | Specifies the application server to process JSP files.<br>**Equivalent property name:**<br>`com.fortify.sca.AppServer` |
| `-appserver-home` *<dir>* | Specifies the application server's home.<br>• For WebLogic, this is the path to the directory that contains the `server/lib` directory.<br>• For WebSphere, this is the path to the directory that |

| Java or Jakarta EE option | Description |
|---|---|
| | contains the `JspBatchCompiler` script.<br><br>**Equivalent property name:**<br>`com.fortify.sca.AppServerHome` |
| `-appserver-version <version>` | Specifies the version of the application server.<br><br>**Equivalent property name:**<br>`com.fortify.sca.AppServerVersion` |
| `-cp <paths>`\|<br>`-classpath <paths>` | Specifies the class path used to analyze Java source code. The format is the same as javac: a semicolon- or colon-separated list of directories. You can use Fortify Static Code Analyzer file specifiers as shown in the following example:<br><br>`-cp "build/classes:lib/*.jar"`<br><br>For information about file specifiers, see "Specifying files and directories" on page 146.<br><br>**Equivalent property name:**<br>`com.fortify.sca.JavaClasspath` |
| `-extdirs <dirs>` | Similar to the javac `extdirs` option, accepts a semicolon- or colon-separated list of directories. Any JAR files found in these directories are included implicitly on the class path.<br><br>**Equivalent property name:**<br>`com.fortify.sca.JavaExtdirs` |
| `-java-build-dir <dirs>` | Specifies one or more directories that contain compiled Java sources. |
| `-source <version>`\|<br>`-jdk <version>` | Indicates the JDK version for which the Java code is written. See the *Fortify Software System Requirements* document for supported versions. The default is version 11.<br><br>**Equivalent property name:**<br>`com.fortify.sca.JdkVersion` |
| `-custom-jdk-dir` | Specifies a directory that contains a JDK. Use this option to specify a version that is not included in the Fortify Static Code Analyzer installation (*<sca_install_dir>*`/Core/bootcp/`). See the *Fortify Software System* |

| Java or Jakarta EE option | Description |
|---|---|
| | *Requirements* document for supported versions.<br>**Equivalent property name:**<br>`com.fortify.sca.CustomJdkDir` |
| `-show-unresolved-symbols` | Displays any unresolved types, fields, and functions referenced in translated Java source files at the end of the translation. It lists only field and function references for which the receiver type is a resolved Java type. Displays each class, field, and function with the source information of the first translated occurrence in the code. This information is also written in the log file.<br>**Equivalent property name:**<br>`com.fortify.sca.ShowUnresolvedSymbols` |
| `-sourcepath <dirs>` | Specifies a semicolon- or colon-separated list of directories that contain source code that is not included in the scan but is used for name resolution. The source path is similar to class path, except it uses source files instead of class files for resolution. Only source files that are referenced by the target file list are translated.<br>**Equivalent property name:**<br>`com.fortify.sca.JavaSourcePath` |

**See also**

"Java and Kotlin properties" on page 197

## Java command-line examples

To translate a single file named `MyServlet.java` with `javaee.jar` as the class path, type:

```
sourceanalyzer -b MyServlet -cp lib/javaee.jar MyServlet.java
```

To translate all `.java` files in the `src` directory using all JAR files in the `lib` directory as a class path, type:

```
sourceanalyzer -b MyProject -cp "lib/*.jar" "src/**/*.java"
```

To translate and compile the `MyCode.java` file with the javac compiler, type:

```
sourceanalyzer -b MyProject javac -classpath libs.jar MyCode.java
```

# Handling Java warnings

To see all warnings that were generated during translation, type the following command before you start the scan phase:

```
sourceanalyzer -b <build_id> -show-build-warnings
```

## Java translation warnings

You might see the following warnings in the Java code translation.

| Warning | Resolution |
| --- | --- |
| `Unable to resolve type...`<br>`Unable to resolve function...`<br>`Unable to resolve field...`<br>`Unable to locate import...`<br>`Unable to resolve symbol...` | These warnings are typically caused by missing resources. For example, some of the `.jar` and `.class` files required to build the application might not have been specified.<br>To resolve these warnings, make sure that you include all the required files that your application uses. |
| `Multiple definitions found for class...` | This warning is typically caused by duplicate classes in the Java files.<br>To resolve these warnings, make sure that the source files displayed in the warning are not duplicates of the same file included several times in the sources to translate (for example if it contains two versions of the same project). If a duplicate exists, remove one of them from the files to translate. Then Fortify Static Code Analyzer can determine which version of the class to use. |

# Translating Jakarta EE (Java EE) applications

To translate Jakarta EE applications, Fortify Static Code Analyzer processes Java source files and Jakarta EE components such as JSP files, deployment descriptors, and configuration files. While you can process all the pertinent files in a Jakarta EE application in one step, your project might require that you break the procedure into its components for integration in a build process or to meet the needs of various stakeholders in your organization.

## Translating Java files

To translate Jakarta EE applications, use the same procedure used to translate Java files. For examples, see "Java command-line examples" on page 53.

## Translating JSP projects, configuration files, and deployment descriptors

In addition to translating the Java files in your Jakarta EE (Java EE) application, you might also need to translate JSP files, configuration files, and deployment descriptors. Your JSP files must be part of a Web Application Archive (WAR). If your source directory is already organized in a WAR file format, you can translate the JSP files directly from the source directory. If not, you might need to deploy your application and translate the JSP files from the deployment directory.

For example:

```
sourceanalyzer -b MyJavaApp "/**/*.jsp" "/**/*.xml"
```

where `/**/*.jsp` refers to the location of your JSP project files and `/**/*.xml` refers to the location of your configuration and deployment descriptor files.

## Jakarta EE (Java EE) translation warnings

You might see the following warning in the translation of Jakarta EE applications:

```
Could not locate the root (WEB-INF) of the web application. Please build
your web application and try again. Failed to parse the following jsp
files:

<list_of_jsp_files>
```

This warning indicates that your web application is not deployed in the standard WAR directory format or does not contain the full set of required libraries. To resolve the warning, make sure that your web application is in an exploded WAR directory format with the correct `WEB-INF/lib` and `WEB-INF/classes` directories that contain all the `.jar` and `.class` files required for your

application. Also verify that you have all the `TLD` files for all your tags and the corresponding JAR files with their tag implementations.

# Translating Java bytecode

OpenText recommends that you do not translate Java bytecode and JSP/Java code in the same call to `sourceanalyzer`. Use multiple invocations of `sourceanalyzer` with the same build ID to translate a project that contains both bytecode and JSP/Java code.

To translate bytecode:

1. Add the following properties to the `fortify-sca.properties` file (or include these properties on the command line using the `-D` option):

   ```
   com.fortify.sca.fileextensions.class=BYTECODE
   com.fortify.sca.fileextensions.jar=ARCHIVE
   ```

   This specifies how Fortify Static Code Analyzer processes `.class` and `.jar` files.
2. Do one of the following:
   - Request that Fortify Static Code Analyzer decompile the bytecode classes to regular Java files for inclusion in the translation.

     Add the following property to the `fortify-sca.properties` file:

     ```
     com.fortify.sca.DecompileBytecode=true
     ```

     or include this property on the command line for the translation phase with the `-D` option:

     ```
     sourceanalyzer -b MyProject -Dcom.fortify.sca.DecompileBytecode=true
     -cp "lib/*.jar" "src/**/*.class"
     ```
   - Request that Fortify Static Code Analyzer translate bytecode without decompilation.

     For best results, OpenText recommends that the bytecode be compiled with full debug information (`javac -g`).
     Include bytecode in the translation phase by specifying the Java bytecode files that you want to translate. For best performance, specify only the `.jar` or `.class` files that require scanning. In the following example, the `.class` files are translated:

     ```
     sourceanalyzer -b MyProject -cp "lib/*.jar" "src/**/*.class"
     ```

# Troubleshooting JSP translation and analysis issues

The following sections provide troubleshooting information for JSP analysis.

## Unable to translate some JSPs

Fortify Static Code Analyzer uses either the built-in compiler or your specific application server JSP compiler to translate JSP files into Java files for analysis. If the JSP parser encounters problems when Fortify Static Code Analyzer converts JSP files to Java files, you will see a message similar to the following:

```
Failed to translate the following jsps into analysis model. Please see the
log file for any errors from the jsp parser and the user manual for hints
on fixing those
<list_of_jsp_files>
```

This typically happens for one or more of the following reasons:

- The web application is not laid out in a proper deployable WAR directory format
- Some JAR files or classes required for the application are missing
- Some tag libraries or their definitions (TLD) for the application are missing

To obtain more information about the problem, perform the following steps:

1. Open the Fortify Static Code Analyzer log file in an editor.
2. Search for the following strings:
   - `Jsp parser stdout:`
   - `Jsp parser stderr:`

The JSP parser generates these errors. Resolve the errors and rerun Fortify Static Code Analyzer.

For more information about how to analyze Jakarta EE applications, see "Translating Jakarta EE (Java EE) applications" on page 55.

## Increased issues count in JSP-related categories

If the analysis results contain a considerable increase in the number of vulnerabilities in JSP-related categories such as cross-site scripting compared with earlier Fortify Static Code Analyzer versions, you can specify the `-legacy-jsp-dataflow` option in the analysis phase (with the `-scan` option). This option enables additional filtering on JSP-related dataflow to reduce the number of spurious false positives detected.

The equivalent property for this option that you can specify in the `fortify-sca.properties` file is `com.fortify.sca.jsp.LegacyDataflow`.

# Chapter 5: Translating Kotlin code

This section describes how to translate Kotlin code.

This section contains the following topics:



## Kotlin command-line syntax

The translation of Kotlin code is similar to the translation of Java code. To translate Kotlin code, all types defined in a library that are referenced in the code must have a corresponding definition in the source code, a class file, or a JAR file. Include all source files on the Fortify Static Code Analyzer command line.

The basic command-line syntax to translate Kotlin code is shown in the following example:

```
sourceanalyzer -b <build_id> -cp <classpath> [<translation_options>]
<files>
```

where

- `-cp <classpath>` specifies the class path to use for the Kotlin source code.
  Include all JAR dependencies normally used to build the project. Separate multiple paths with semicolons (Windows) or colons (non-Windows).

  Fortify Static Code Analyzer loads classes in the order they appear in the class path. If there are multiple classes with the same name in the list, Fortify Static Code Analyzer uses the first loaded class. In the following example, if both `A.jar` and `B.jar` include a class called `MyData.class`, Fortify Static Code Analyzer uses the `MyData.class` from `A.jar`.

  ```
  sourceanalyzer -cp "A.jar:B.jar" myfile.kt
  ```

  OpenText strongly recommends that you avoid using duplicate classes with the `-cp` option.

For descriptions of all the available Kotlin-specific command-line options, see "Kotlin command-line options" on the next page.

## Kotlin command-line options

The following table describes the Kotlin-specific command-line options.

| Kotlin option | Description |
|---|---|
| `-cp <paths>`\|<br>`-classpath <paths>` | Specifies the class path used to analyze Kotlin source code, which is a semicolon- or colon-separated list of directories. You can use Fortify Static Code Analyzer file specifiers as shown in the following example:<br>`-cp "build/classes:lib/*.jar"`<br>For information about file specifiers, see "Specifying files and directories" on page 146.<br>**Equivalent property name:**<br>`com.fortify.sca.JavaClasspath` |
| `-source <version>`\|<br>`-jdk <version>` | Indicates the JDK version for which the Kotlin code is written. See the *Fortify Software System Requirements* document for supported versions. The default is version 11.<br>**Equivalent property name:**<br>`com.fortify.sca.JdkVersion` |
| `-sourcepath <dirs>` | Specifies a semicolon- or colon-separated list of directories that contain Java source code that is not included in the scan but is used for name resolution. The source path is similar to class path, except it uses source files instead of class files for resolution. Only source files that are referenced by the target file list are translated.<br>**Equivalent property name:**<br>`com.fortify.sca.JavaSourcePath` |
| `-jvm-default <mode>` | Specifies the generation of the `DefaultImpls` class for methods with bodies in Kotlin interfaces. The valid values for *<mode>* are:<br>• `disable`—Specifies to generate the `DefaultImpls` class for each interface that contains methods with bodies.<br>• `all`—Specifies to generate the `DefaultImpls` class if an interface is annotated with |

| Kotlin option | Description |
|---|---|
| | `@JvmDefaultWithCompatibility`.<br>• `all-compatibility`—Specifies to generate the `DefaultImpls` class unless an interface is annotated with `@JvmDefaultWithoutCompatibility`.<br>**Equivalent property name:**<br>`com.fortify.sca.KotlinJvmDefault` |

**See also**

"Java and Kotlin properties" on page 197

## Kotlin command-line examples

To translate a single file named `MyKotlin.kt` with `A.jar` as the class path, type:

```
sourceanalyzer -b MyProject -cp lib/A.jar MyKotlin.kt
```

To translate all `.kt` files in the `src` directory using all JAR files in the `lib` directory as a class path, type:

```
sourceanalyzer -b MyProject -cp "lib/**/*.jar" "src/**/*.kt"
```

To translate a gradle project using gradlew, type:

```
sourceanalyzer -b MyProject gradlew clean assemble
```

To translate all files in the `src` directory using Java dependencies from `src/java` and all JAR files in the `lib` directory and subdirectories as a class path, type:

```
sourceanalyzer -b MyProject -cp "lib/**/*.jar" -sourcepath "src/java" "src"
```

# Kotlin and Java translation interoperability

If your project contains Kotlin code that refers to Java code, you can provide Java files to the translator the same way as Kotlin files that refers to another Kotlin file. You can provide them as part of the translated project source or as `-sourcepath` parameters.

If your project contains Java code that refers to Kotlin code, make sure that the Java and Kotlin code are translated in the same Fortify Static Code Analyzer instance so that the Java references to Kotlin elements are resolved correctly. Kotlin to Java interoperability does not support Kotlin files provided by the `-sourcepath` option. For more information about the `-sourcepath` option, see "Kotlin command-line options" on the previous page

# Translating Kotlin scripts

Fortify Static Code Analyzer supports translation of Kotlin scripts excluding experimental script customization. Script customization includes adding external properties, providing static or dynamic dependencies, and so on. Script definitions (templates) are used to create custom scripts and the template is applied to the script based on the `*.kts` extension. Fortify Static Code Analyzer translates `*.kts` files but does not apply these templates.

# Chapter 6: Translating Visual Studio projects

Fortify Static Code Analyzer provides a build integration to support translation of the following Visual Studio project types:

- C/C++ projects
- C# projects that target .NET Framework and .NET Core
- ASP.NET applications that target ASP.NET framework and ASP.NET Core
- Xamarin applications that target Android and iOS platforms

For a list of supported versions of relevant programming languages and frameworks, as well as Visual Studio and MSBuild, see the *Fortify Software System Requirements* document.

This section contains the following topics:



## Visual Studio Project translation prerequisites

OpenText recommends that each project you translate is complete and that you perform the translation in an environment where you can build it without errors. For a list of software environment requirements, see the *Fortify Software System Requirements* document. A complete project contains the following:

- All necessary source code files (C/C++, C#, or VB.NET).
- All required reference libraries.

  This includes those from relevant frameworks, NuGet packages, and third-party libraries.
- For C/C++ projects, include all necessary header files that do not belong to the Visual Studio or MSBuild installation.
- For ASP.NET and ASP.NET Core projects, include all the necessary ASP.NET page files.

  The supported ASP.NET page types are ASAX, ASCX, ASHX, ASMX, ASPX, AXML, BAML, CSHTML, Master, RAZOR, VBHTML, and XAML.

## Visual Studio Project command-line syntax

The basic syntax to translate a Visual Studio solution or project is to specify the corresponding build option for your project as part of the Fortify Static Code Analyzer translation command. This starts a

build integration that analyzes your solution and project files and automatically executes the appropriate translation steps.

**Important!** To ensure that the build integration correctly pulls in all of the appropriate project dependencies and resources, you must run the Fortify Static Code Analyzer command from a command prompt with access to your build environment configuration. OpenText strongly recommends you run this command from the Developer Command Prompt for Visual Studio to ensure an optimal environment for the translation.

In the following examples, Fortify Static Code Analyzer translates all the projects contained in the Visual Studio solution `Sample.sln`. You can also translate one or more specific projects by providing a semicolon-separated list of projects.

- For a .NET 6.0 or later solution on Windows or Linux, use the following commands to translate the solution:
  a. Optionally, run the following command to remove any intermediate files from previous project builds:

     ```
     dotnet clean Sample.sln
     ```

  b. Optionally, run the following command to ensure that all required reference libraries are downloaded and installed in the project. Run this command from the top-level folder of the project:

     ```
     dotnet restore Sample.sln
     ```

  c. Run one of the following Fortify Static Code Analyzer commands depending on how your project build is implemented. You can include any additional build parameters in this command:

     ```
     sourceanalyzer –b MyProject dotnet msbuild Sample.sln
     ```

     or

     ```
     sourceanalyzer –b MyProject dotnet build Sample.sln
     ```

- For a C, C++, and .NET Framework solution (4.8.x or earlier) on Windows, use the following command to translate the solution:

  ```
  sourceanalyzer –b MyProject msbuild /t:rebuild [<msbuild_options>]
  Sample.sln
  ```

  **Note:** If you run Fortify Static Code Analyzer from a Windows Command Prompt instead of the Visual Studio Developer Command Prompt, you must set up the environment and make sure the path to the MSBuild executable required to build your project is included in the PATH environment variable.

After the translation is complete, perform the analysis phase and save the results in an FPR file as shown in the following example:

```
sourceanalyzer –b MyProject -scan -f MyResults.fpr
```

# Handling special cases for translating Visual Studio projects

## Running translation from a script

To perform the translation in a non-interactive mode such as with a script, establish an optimal environment for translation by executing the following command before you run the Fortify Static Code Analyzer translation:

```
cmd.exe /k <vs_install_dir>/Common7/Tools/VSDevCmd.bat
```

where *<vs_install_dir>* is the directory where you installed Visual Studio.

## Translating plain .NET and ASP.NET projects

You can translate plain .NET and ASP.NET projects from the Windows Command Prompt as well as from a Visual Studio environment. When you translate from the Windows Command Prompt, make sure the path to the MSBuild executable required to build your project is included in the PATH environment variable.

## Translating C/C++ and Xamarin projects

You must translate C/C++ and Xamarin projects either from a Developer Command Prompt for Visual Studio or from the Fortify Extension for Visual Studio.

**Note:** For Xamarin projects, there is no need to use a custom rule for the Xamarin.Android API if a rule for the corresponding native Android API exists in the Fortify Secure Coding Rulepacks. Doing so can cause duplicate issues to be reported.

## Translating projects with settings containing spaces

If your project is built with a configuration or other settings file that contains spaces, make sure to enclose the setting values in quotes. For example, to translate a Visual Studio solution `Sample.sln` that is built with configuration `My Configuration`, use the following command:

```
sourceanalyzer –b MySampleProj msbuild /t:rebuild
/p:Configuration="My Configuration" Sample.sln
```

## Translating a single project from a Visual Studio solution

If your Visual Studio solution contains multiple projects, you have the option to translate a single project instead of the entire solution. Project files have a file name extension that ends with `proj` such as `.vcxproj` and `.csproj`. To translate a single project, specify the project file instead of the solution as the parameter for the MSBuild command.

The following example translates the `Sample.vcxproj` project file:

```
sourceanalyzer –b MySampleProj msbuild /t:rebuild Sample.vcxproj
```

## Analyzing projects that build multiple executable files

If your Visual Studio or MSBuild project builds multiple executable files (such as files with the file name extension `*.exe`), OpenText strongly recommends that you run the analysis phase separately for each executable file to avoid false positive issues in the analysis results. To do this, use the `–binary-name` option when you run the analysis phase and specify the executable file name or .NET assembly name as the parameter.

The following example shows how to translate and analyze a Visual Studio solution `Sample.sln` that consists of two projects, Sample1 (a C++ project with no associated .NET assembly name) and Sample2 (a .NET project with .NET assembly name `Sample2`). Each project builds a separate executable file, `Sample1.exe` and `Sample2.exe`, respectively. The analysis results are saved in `Sample1.fpr` and `Sample2.fpr` files.

```
sourceanalyzer -b MySampleProj msbuild /t:rebuild Sample.sln
sourceanalyzer -b MySampleProj -scan -binary-name Sample1.exe -f
Sample1.fpr
sourceanalyzer -b MySampleProj -scan -binary-name Sample2.exe -f
Sample2.fpr
```

For more information about the `-binary-name` option, see "Analysis options" on page 136.

# Alternative ways to translate Visual Studio projects

This section describes alternative methods of translating Visual Studio projects.

## Alternative translation options for Visual Studio solutions

The following are two alternative ways of translation available only for Visual Studio solutions:

- Use the Fortify Extension for Visual Studio

  The Fortify Extension for Visual Studio runs the translation and analysis (scan) phases together in one step.
- Append a devenv command to the Fortify Static Code Analyzer command

  The following command translates the Visual Studio solution `Sample.sln`:

  ```
  sourceanalyzer –b MySampleProj devenv Sample.sln /rebuild
  ```

  Note that Fortify Static Code Analyzer converts a devenv invocation to the equivalent MSBuild invocation, therefore in this case, the solution with this command is built by MSBuild instead of the devenv tool.

## Translating without explicitly running Fortify Static Code Analyzer

You have the option to translate your Visual Studio project without invoking Fortify Static Code Analyzer directly. This requires the `Fortify.targets` file, which is located in *<sca_install_dir>*`\Core\private-bin\sca\MSBuildPlugin` in the `DotNet` and `Framework` directory. You can specify the file using an absolute or relative path in the build command line that builds your project. Use the path with the `Dotnet` or `Framework` directory depending on the build command you are using: `dotnet.exe` or `MSBuild.exe` respectively. For example:

```
dotnet.exe msbuild /t:rebuild /p:CustomAfterMicrosoftCommonTargets=<sca_
install_dir>\Core\private-bin\sca\MSBuildPlugin\Dotnet\Fortify.targets
Sample.sln
```

or

```
msbuild.exe /t:rebuild
/p:CustomAfterMicrosoftCommonTargets=<sca_install_dir>\Core\private-
bin\sca\MSBuildPlugin\Framework\Fortify.targets Sample.sln
```

There are several environment variables that you can set to configure the translation of your project. Most of them have default values, which Fortify Static Code Analyzer uses if the variable is not set. These variables are listed in the following table.

| Environment variable | Description | Default value |
|---|---|---|
| FORTIFY_ MSBUILD_ BUILDID | Specifies the Fortify Static Code Analyzer build ID for translation. Make sure that you set this value.<br>This is equivalent to the Fortify Static Code Analyzer `-b` option. | None |
| FORTIFY_ MSBUILD_ DEBUG | Enables debug mode. This is equivalent to the Fortify Static Code Analyzer `–debug` option. | False |
| FORTIFY_ MSBUILD_ DEBUG_ VERBOSE | Enables verbose debug mode. This is equivalent to the Fortify Static Code Analyzer `–debug-verbose` option. Takes precedence over FORTIFY_MSBUILD_ DEBUG variable if both are set to true. | False |
| FORTIFY_ MSBUILD_MEM | Specifies the memory requirements for translation in the form of the JVM `-Xmx` option. For example, `-Xmx2G`. | Automatic allocation based on physical memory available on the system |
| FORTIFY_ MSBUILD_ SCALOG | Specifies the location (absolute path) of the Fortify Static Code Analyzer log file.<br>This is equivalent to the Fortify Static Code Analyzer `-logfile` option. | `%LOCALAPPDATA%/Fortify/ sca/log/sca.log` |

# Chapter 7: Translating C and C++ code

This section describes how to translate C and C++ code.

**Important!** The chapter describes how to translate C and C++ code that is ***not*** a part of a Visual Studio or MSBuild project. For instructions on how to translate Visual Studio or MSBuild projects, see "Translating Visual Studio projects" on page 62.

This section contains the following topics:



## C and C++ Code translation prerequisites

Make sure that you have any dependencies required to build the project available, including headers for third-party libraries. Fortify Static Code Analyzer translation does not require object files and static/dynamic library files.

**Note:** Fortify Static Code Analyzer might not support all non-standard C++ constructs.

If you use Gradle to build your C++ project, make sure that the C++ Application Plugin is added to your Gradle file in one of the following formats:

```
apply plugin: 'cpp'
```

```
plugins {
  id 'cpp-application'
}
```

**See also**

"Using Gradle integration" on page 125

## C and C++ command-line syntax

Command-line options passed to the compiler affect preprocessor execution and can enable or disable language features and extensions. For Fortify Static Code Analyzer to interpret your source code in the same way as the compiler, the translation phase for C/C++ source code requires the

complete compiler command line. Prefix your original compiler command with the `sourceanalyzer` command and options.

The basic command-line syntax for translating a single file is:

```
sourceanalyzer -b <build_id> [<sca_options>] <compiler> [<compiler_
options>] <file>.c
```

where:

- *<sca_options>* are options passed to Fortify Static Code Analyzer.
- *<compiler>* is the name of the C/C++ compiler you use, such as `gcc`, `g++`, or `cl`. See the *Fortify Software System Requirements* document for a list of supported C/C++ compilers.
- *<compiler_options>* are options passed to the C/C++ compiler.
- *<file>*`.c` must be in ASCII or UTF-8 encoding.

**Note:** All Fortify Static Code Analyzer options must precede the compiler options.

The compiler command must successfully complete when executed on its own. If the compiler command fails, then the Fortify Static Code Analyzer command prefixed to the compiler command also fails.

For example, if you compile a file with the following command:

```
gcc -I. -o hello.o -c helloworld.c
```

then you can translate this file with the following command:

```
sourceanalyzer -b MyProject gcc -I. -o hello.o -c helloworld.c
```

Fortify Static Code Analyzer executes the original compiler command as part of the translation phase. In the previous example, the command produces both the translated source suitable for scanning, and the object file `hello.o` from the `gcc` execution. You can use the Fortify Static Code Analyzer `-nc` option to disable the compiler execution.

# Scanning pre-processed C and C++ code

If, before compilation, your C/C++ build executes a third-party C preprocessor that Fortify Static Code Analyzer does not support, you must start the Fortify Static Code Analyzer translation on the intermediate file. Fortify Static Code Analyzer touchless build integration automatically translates the intermediate file provided that your build executes the unsupported preprocessor and supported compiler as two commands connected by a temporary file rather than a pipe chain.

# C/C++ Precompiled Header Files

Some C/C++ compilers support Precompiled Header Files, which can improve compilation performance. Some compilers' implementations of this feature have subtle side-effects. When the feature is enabled, the compiler might accept erroneous source code without warnings or errors. This can result in a discrepancy where Fortify Static Code Analyzer reports translation errors even when your compiler does not.

If you use your compiler's Precompiled Header feature, disable Precompiled Headers, and then perform a full build to make sure that your source code compiles cleanly.

# Chapter 8: Translating JavaScript and TypeScript code

You can analyze JavaScript projects that contain JavaScript, TypeScript, JSX, and TSX source files, as well as JavaScript embedded in HTML files.

Some JavaScript frameworks are transpiled (source-to-source compilation) to plain JavaScript, which is generated code. Use the `-exclude` command-line option to exclude this type of code.

When you translate JavaScript and TypeScript code, make sure that you specify all source files together in one invocation. Fortify Static Code Analyzer does not support adding new files to the file list associated with the build ID on subsequent invocations.

Fortify Static Code Analyzer does not translate minified JavaScript (*.min.js).

**Note:** There are some types of minified JavaScript files that Fortify Static Code Analyzer cannot automatically detect for exclusion from the translation. Use the `-exclude` command-line option to exclude these files directly.

This section contains the following topics:



## Translating pure JavaScript projects

The basic command-line syntax to translate JavaScript is:

```
sourceanalyzer -b <build_id> <js_file_or_dir>
```

where *<js_file_or_dir>* is either the name of the JavaScript file to be translated or a directory that contains multiple JavaScript files. You can also translate multiple files by specifying `*.js` for the *<js_file_or_dir>*.

# Excluding dependencies

You can avoid translating specific dependencies by adding them to the appropriate property setting in the `fortify-sca.properties` file. Files specified in the following properties are *not* translated:

- `com.fortify.sca.skip.libraries.ES6`
- `com.fortify.sca.skip.libraries.jQuery`
- `com.fortify.sca.skip.libraries.javascript`
- `com.fortify.sca.skip.libraries.typescript`

Each property specifies a list of comma- or colon-separated file names (without path information).

The files specified in these properties apply to both local files and files on the internet. Suppose, for example, that the JavaScript code includes the following local file reference:

```
<script src="js/jquery-ui.js" type="text/javascript" charset="utf-
8"></script>
```

By default, the `com.fortify.sca.skip.libraries.jQuery` property in the `fortify-sca.properties` file includes `jquery-us.js`, and therefore Fortify Static Code Analyzer does not translate the file shown in the previous example.

You can use regular expressions for the file names. Note that Fortify Static Code Analyzer automatically inserts the regular expression `'(-?\d+\.\d+\.\d+)?'` before `.min.js` or `.js` for each file name included in the `com.fortify.sca.skip.libraries.jQuery` property value.

**Note:** You can also exclude local files or entire directories with the `-exclude` command-line option. For more information about this option, see "Translation options" on page 134.

To provide a thorough analysis, dependent files are included in the translation even if the dependency is in a language that is disabled with the `-disable-language` option. For more information about the option to disable languages, see "Translation options" on page 134).

# Managing issue detection in NPM dependencies

By default, Fortify Static Code Analyzer does not report issues in NPM dependencies (files in the `node_modules` directory). This is configured with the `com.fortify.sca.exclude.node.modules` property, which is set to `true` by default.

Setting the `com.fortify.sca.exclude.node.modules` property to `false` directs Fortify Static Code Analyzer to use the following options, which determines what results to report for NPM dependencies:

- The `com.fortify.sca.follow.imports` property is enabled by default and directs Fortify Static Code Analyzer to resolve all imported files (including NPM dependencies) used in the project

and include them in the translation and the subsequent analysis. For resolution to find imported files within the project, Fortify Static Code Analyzer uses an algorithm similar to Node.js (see the Node.js website for more information).

Setting this property to false prevents imported NPM dependencies that are not explicitly included on the command-line from being included in the translation and analysis.

- The `com.fortify.sca.exclude.unimported.node.modules` property is enabled by default and directs Fortify Static Code Analyzer to exclude node_modules directories that are not referenced by the project. This property is enabled by default to avoid translating dependencies that are not needed for the final project such as those only required for the build system.

  Setting this property to false causes Fortify Static Code Analyzer to include in the translation (and subsequent analysis) all modules discovered during resolution (with the `com.fortify.sca.follow.imports` property enabled) that are not referenced by the project.

You can use the `-exclude` option together with the two properties listed previously to specifically exclude modules. Use of this option takes precedence over the previously described property configurations.

**Note:** OpenText does not recommend using the -exclude option to exclude node modules if `com.fortify.sca.exclude.node.modules` is set to `true`, because it can change the quality of the results.

**See also**

"Examples of excluding NPM dependencies" below

## Examples of excluding NPM dependencies

The following examples illustrate three different scenarios for excluding NPM dependencies. All these examples use the following directory structure:

```
./
  RootProjectDir
    innerSrcDir
      node_modules
        innerProjectReferencedModule
          index.ts
        moduleNotReferencedByProject
          index.ts
      innerProject.ts (contains import from innerProjectReferencedModule)
    node_modules
      projectReferencedModule
        index.ts
      moduleNotReferencedByProject
        index.ts
    projectMain.ts (contains import from projectReferencedModule)
```

**Example 1**

This example shows the files are translated with `com.fortify.sca.exclude.unimported.node.modules` set to `false`. In this case, `com.fortify.sca.follow.imports` and `com.fortify.sca.exclude.unimported.node.modules` are both set to `true`.

```
sourceanalyzer RootProjectDir/ -Dcom.fortify.sca.exclude.node.modules=false
```

The following files are included in the translation for Example 1:

```
./RootProjectDir/innerSrcDir/innerProject.ts
./RootProjectDir/innerSrcDir/node_
modules/innerProjectReferencedModule/index.ts
./RootProjectDir/projectMain.ts
./RootProjectDir/node_modules/projectReferencedModule/index.ts
```

**Example 2**

This example shows that in addition to modules referenced by the project, modules found during resolution but not referenced by the project are also included in the translation.

```
sourceanalyzer RootProjectDir/ -
Dcom.fortify.sca.exclude.unimported.node.modules=false
```

The following files are included in the translation for Example 2:

```
./RootProjectDir/innerSrcDir/innerProject.ts
./RootProjectDir/innerSrcDir/node_
modules/innerProjectReferencedModule/index.ts
./RootProjectDir/innerSrcDir/node_
modules/moduleNotReferencedByProject/index.ts
./RootProjectDir/projectMain.ts
./RootProjectDir/node_modules/projectReferencedModule/index.ts
./RootProjectDir/node_modules/moduleNotReferencedByProject/index.ts
```

**Example 3**

This example shows use of the -exclude option to exclude all files under any `node_modules` directory. The `-exclude` option overrides resolution of modules based on the configuration of the `com.fortify.sca.follow.imports` and `com.fortify.sca.exclude.unimported.node.modules` properties.

```
sourceanalyzer RootProjectDir/ -exclude "**/node_modules/*.*"
```

The following files are included in the translation for Example 3:

```
./RootProjectDir/innerSrcDir/innerProject.ts
./RootProjectDir/projectMain.ts
```

# Translating JavaScript projects with HTML files

If the project contains HTML files in addition to JavaScript files, set the `com.fortify.sca.EnableDOMModeling` property to true in the `fortify-sca.properties` file or on the command line as shown in the following example:

```
sourceanalyzer –b MyProject <js_file_or_dir>
-Dcom.fortify.sca.EnableDOMModeling=true
```

When you set the `com.fortify.sca.EnableDOMModeling` property to true, this can decrease false negative reports of DOM-related attacks, such as DOM-related cross-site scripting issues.

**Note:** If you enable this option, Fortify Static Code Analyzer generates JavaScript code to model the DOM tree structure in the HTML files. The duration of the analysis phase might increase (because there is more translated code to analyze).

If you set the `com.fortify.sca.EnableDOMModeling` property to `true`, you can also specify additional HTML tags for Fortify Static Code Analyzer to include in the DOM modeling with the `com.fortify.sca.DOMModeling.tags` property. By default, Fortify Static Code Analyzer includes the following HTML tags: `body`, `button`, `div`, `form`, `iframe`, `input`, `head`, `html`, and `p`.

For example, to additionaly include the HTML tags `ul` and `li` in the DOM model, use the following command:

```
sourceanalyzer –b MyProject <js_file_or_dir>
-Dcom.fortify.sca.DOMModeling.tags=ul,li
```

# Including external JavaScript or HTML in the translation

To include external JavaScript or HTML files that are specified with the `src` attribute, you can specify which domains Fortify Static Code Analyzer can download and include in the translation phase. To do this, specify one or more domains with the `com.fortify.sca.JavaScript.src.domain.whitelist` property.

**Note:** You can also set this property globally in the `fortify-sca.properties` file.

For example, you might have the following statement in your HTML file:

```
<script src='http://xyzdomain.com/foo/bar.js' language='text/javascript'/>
</script>
```

If you are confident that the `xyzdomain.com` domain is a safe location from which to download files, then you can include it in the translation phase by adding the following property specification on the command line:

```
-Dcom.fortify.sca.JavaScript.src.domain.whitelist="xyzdomain.com/foo"
```

**Note:** You can omit the `www.` prefix from the domain in the property value. For example, if the src tag in the original HTML file specifies to download files from `www.google.com`, you can specify just the `google.com` domain.

To trust more than one domain, include each domain separated by the vertical bar character (`|`) as shown in the following example:

```
-Dcom.fortify.sca.JavaScript.src.domain.whitelist=
"xyzdomain.com/foo|abcdomain.com|123.456domain.com”
```

If you are using a proxy server, then you need to include the proxy server information on the command line as shown in the following example:

```
-Dhttp.proxyHost=example.proxy.com -Dhttp.proxyPort=8080
```

For a complete list of proxy server options, see the Networking Properties Java documentation.

# Chapter 9: Translating Python code

Fortify Static Code Analyzer translates Python applications, and processes files with the `.py` extension as Python source code. Fortify Static Code Analyzer supports translation of the Django and Flask frameworks.

This section contains the following topics:



# Python translation command-line syntax

The basic command-line syntax to translate Python code is:

```
sourceanalyzer -b <build_id> -python-version <python_version>
-python-path <dirs> <files>
```

**Note:** When you translate Python code, make sure that you specify all source files together in one invocation. Fortify Static Code Analyzer does not support adding new files to the file list associated with the build ID on subsequent invocations.

## Python command-line options

The following table describes the Python options.

| Python option | Description |
|---|---|
| `-python-version <version>` | Specifies the Python source code version to scan. The valid values for *<version>* are 2 and 3. The default value is 3.<br>**Equivalent property name:**<br>`com.fortify.sca.PythonVersion` |

| Python option | Description |
|---|---|
| `-python-no-auto-root-calculation` | Disables the automatic calculation of a common root directory of all project source files to use for importing modules and packages.<br>**Equivalent property name:**<br>`com.fortify.sca.PythonNoAutoRootCalculation` |
| `-python-path <dirs>` | Specifies a semicolon-separated (Windows) or colon-separated (non-Windows) list of additional import directories. You can use the `-python-path` option to specify all paths used to import packages or modules. Include all paths to namespace package directories with this option. Fortify Static Code Analyzer sequentially searches the specified paths for each imported file and uses the first file encountered.<br>**Equivalent property name:**<br>`com.fortify.sca.PythonPath` |
| `-django-template-dirs <dirs>` | Specifies a semicolon-separated (Windows) or colon-separated (non-Windows) list of directories that contain Django templates. Fortify Static Code Analyzer sequentially searches the specified paths for each Django template file and uses the first template file encountered.<br>**Equivalent property name:**<br>`com.fortify.sca.DjangoTemplateDirs` |
| `-django-disable-autodiscover` | Specifies that Fortify Static Code Analyzer does not automatically discover Django templates.<br>**Equivalent property name:**<br>`com.fortify.sca.DjangoDisableAutodiscover` |
| `-jinja-template-dirs <dirs>` | Specifies a semicolon-separated (Windows) or colon-separated (non-Windows) list of directories that contain Jinja2 templates. Fortify Static Code Analyzer sequentially searches the specified paths for each Jinja2 template file and uses the first template file encountered.<br>**Equivalent property name:**<br>`com.fortify.sca.JinjaTemplateDirs` |
| `-disable-template-autodiscover` | Specifies that Fortify Static Code Analyzer does not automatically discover Django or Jinja2 templates.<br>**Equivalent property name:**<br>`com.fortify.sca.DisableTemplateAutodiscover` |

**See also**

"Python properties" on page 202

## Python command-line examples

Translate Python 3 code on Windows:

```
sourceanalyzer -b Python3Proj -python-path
"C:\Python312\Lib;C:\Python312\Lib\site-packages" src/*.py
```

Translate Python 2 code on Windows:

```
sourceanalyzer -b MyPython2 -python-version 2 -python-path
"C:\Python27\Lib;C:\Python27\Lib\site-packages" src/*.py
```

Translate Python 3 code on non-Windows:

```
sourceanalyzer -b Python3Proj -python-path
/usr/lib/python3.12:/usr/local/lib/python3.12/site-packages src/*.py
```

Translate Python 2 code on non-Windows:

```
sourceanalyzer -b MyPython2 -python-version 2 -python-path
/usr/lib/python2.7:/usr/local/lib/python2.7/site-packages src/*.py
```

# Translating Python in a virtual environment

This section describes how to translate Python projects in virtual environments. Make sure that all project dependencies are installed in your virtual environment. To translate a Python project in a virtual environment, include the `-python-path` option to specify the project dependencies.

## Python virtual environment example

To translate a Python project where the virtual environment name is `myenv` and the dependencies for the project are installed in the `myenv/lib/python<version>/site-packages` directory, type:

```
sourceanalyzer –b mybuild -python-path "myenv/lib/python<version>/site-
packages/" myproject/
```

## Conda environment example

To translate a Python project where the conda environment name is `myenv` and the project dependencies are installed in the *<conda_install_*

*dir*>/envs/myenv/lib/python<*version*>/site-packages directory, type:

```
sourceanalyzer –b mybuild -python-path "<conda_install_
dir>/envs/myenv/lib/python<version>/site-packages/" myproject/
```

# Including imported modules and packages

To translate Python applications and prepare for a scan, Fortify Static Code Analyzer searches for any imported modules and packages used by the application. Fortify Static Code Analyzer does not respect the PYTHONPATH environment variable, which the Python runtime system uses to find imported modules and packages.

Fortify Static Code Analyzer searches for imported modules and packages using the list of directories in the following order:

1. The common root directory for all project source files. which Fortify Static Code Analyzer calculates automatically. For example, if there are two project directories PrimaryDir/project1/* and PrimaryDir/project2/*, the common root directory is PrimaryDir.
   To remove the common root directory as a search target for imported modules and packages, include the -python-no-auto-root-calculation option in the translation command.
2. The directories specified with the -python-path option.
   Fortify Static Code Analyzer includes a subset of modules from the standard Python library (module "builtins", all modules originally written in C, and others) in the translation. Fortify Static Code Analyzer first searches for a standard Python library module in the set included with Fortify Static Code Analyzer and then in the paths specified with the -python-path option. If your Python code imports any module that Fortify Static Code Analyzer cannot find, it produces a warning. To make sure that all modules of the standard Python library are found, add the path to your standard Python library in the -python-path list.
3. The current directory that contains the file being translated. For example, when Fortify Static Code Analyzer translates a PrimaryDir/project1/a.py, the directory PrimaryDir/project1 is added as the last directory to search for imported modules and packages.

# Including namespace packages

To translate namespace packages, include all the paths to the namespace package directories with the -python-path option. For example, if you have two subpackages for a namespace package package_name in multiple folders:

```
/path_1/package_name/subpackageA
/path_2/package_name/subpackageB
```

Include /path_1;/path_2 with the -python-path option in the sourceanalyzer command line.

# Translating Django and Flask

To translate code created using the Django or Flask framework, add the following properties to the *<sca_install_dir>*/Core/config/fortify-sca.properties configuration file:

```
com.fortify.sca.limiters.MaxPassthroughChainDepth=8
com.fortify.sca.limiters.MaxChainDepth=8
```

By default, Fortify Static Code Analyzer attempts to discover Django and Jinja2 templates in the project root directory. All detected Django and Jinja2 templates are automatically added to the translation. You can specify additional locations of Django or Jinja2 template files by adding the `-django-template-dirs` or the `-jinja-template-dirs` option to the sourceanalyzer command.

If you do not want Fortify Static Code Analyzer to automatically discover Django and Jinja2 templates, use the `-disable-template-autodiscover` option. If your project requires Django or Jinja2 templates, but the project is configured such that the templates are in an unexpected location, use the `-django-template-dirs` or `-jinja-template-dirs` option to specify the directories that contain the templates in addition to the `-disable-template-autodiscover` option as shown in the following non-Windows examples:

```
sourceanalyzer -b djangoProj -python-path
/usr/lib/python3.12:/usr/local/lib/python3.12/site-packages djangoProj -
django-template-dirs djangoProj/templatedir1:/djangoProj/dir2 -disable-
template-autodiscover
```

```
sourceanalyzer -b flaskProj -python-path
/usr/lib/python3.12:/usr/local/lib/python3.12/site-packages flaskProj -
jinja-template-dirs flaskProj/templatedir1:/flaskProj/dir2 -disable-
template-autodiscover
```

The following example translates a Python project that has a combination of Django and Jinja2 templates on Windows:

```
sourceanalyzer -b pythonProj -python-path
"C:\Python312\Lib;C:\Python312\Lib\site-packages" flaskProj -django-
template-dirs "C:\djangoProj\templatedir1;C:\djangoProj\dir2" -jinja-
template-dirs "C:\flaskProj\templatedir1;C:\flaskProj\dir2" -disable-
template-autodiscover
```

# Chapter 10: Translating code for mobile platforms

Fortify Static Code Analyzer supports analysis of the following mobile application source languages:

- Swift, Objective-C, and Objective-C++ for iOS applications developed using Xcode
- Java for Android applications

For information about translating Xamarin applications, see "Translating Visual Studio projects" on page 62.

This section contains the following topics:



# Translating Apple iOS projects

This section describes how to translate Swift, Objective-C, and Objective-C++ source code for iOS applications. Fortify Static Code Analyzer automatically integrates with the Xcode Command Line Tool, Xcodebuild, to identify the project source files.

## iOS project translation prerequisites

The following are the prerequisites for translating iOS projects:

- Objective-C++ projects must use the non-fragile Objective-C runtime (ABI version 2 or 3).
- Use Apple's `xcode-select` command-line tool to set your Xcode path. Fortify Static Code Analyzer uses the system global Xcode configuration to find the Xcode toolchain and headers.
- Make sure that all source files required for a successful Xcode build are provided.

  You can exclude files from the analysis using the `-exclude` option (see "iOS code analysis command-line syntax" on the next page).
- Make sure that you have any dependencies required to build the project available.
- To translate Swift code, make sure that you have available all third-party modules, including CocoaPods. Bridging headers must also be available. However, Xcode usually generates them automatically during the build.
- If your project includes property list files in binary format, you must first convert them to XML format. You can do this with the Xcode `putil` command.
- To translate Objective-C projects, ensure that the headers for third-party libraries are available.

- To translate WatchKit applications, make sure that you translate both the iPhone application target and the WatchKit extension target.

## iOS code analysis command-line syntax

The command-line syntax to translate iOS code using Xcodebuild is:

```
sourceanalyzer -b <build_id> xcodebuild [<compiler_options>]
```

where *<compiler_options>* are the supported options that are passed to the Xcode compiler. You must include the `build` option with any *<compiler_options>*. The Fortify Static Code Analyzer Xcodebuild integration does not support the output format of alternate build commands such as `xcodebuild archive`.

**Note:** Xcodebuild compiles the source code when you run this command.

To exclude files from the analysis, use the `-exclude` option (see "Translation options" on page 134). All source files that match the exclude specification are not translated, even if they are included in the Xcode build. The following is an example:

```
sourceanalyzer -b MyProject -exclude "**/TestFile.swift" xcodebuild clean
build
```

If your application uses any property list files (for example, *<file>*`.plist`), translate these files with a separate `sourceanalyzer` command. Use the same build ID that you used to translate the project files. The following is an example:

```
sourceanalyzer -b MyProject <path_to_plist_files>
```

If your project uses CocoaPods, include `-workspace` to build the project. For example:

```
sourceanalyzer -b DemoAppSwift xcodebuild clean build -workspace
DemoAppSwift.xcworkspace -scheme DemoAppSwift -sdk iphonesimulator
```

After the translation is complete, you can perform the analysis phase and save the results in an FPR file, as shown in the following example:

```
sourceanalyzer -b DemoAppSwift -scan -f MyResults.fpr
```

# Translating Android projects

This section describes how to translate Java source code for Android applications. You can use Fortify Static Code Analyzer to scan the code with Gradle from either:

- Your operating system's command line
- A terminal window running in Android Studio

The way you use Gradle is the same for either method.

**Note:** You can also scan Android code directly from Android Studio with the Fortify Analysis Plugin for IntelliJ IDEA and Android Studio. For more information, see the *OpenText™ Fortify Analysis Plugin for IntelliJ IDEA and Android Studio User Guide*.

## Android project translation prerequisites

The following are the prerequisites for translating Android projects:

- Android Studio and the relevant Android SDKs are installed on the system where you will run the scans
- Your Android project uses Gradle for builds.

  If you have an older project that does not use Gradle, you must add Gradle support to the associated Android Studio project

  Use the same version of Gradle that is provided with the version of Android Studio that you use to create your Android project
- Make sure you have available all dependencies that are required to build the Android code in the application's project
- To translate your Android code from a command window that is not displayed within Android Studio, make sure that Gradle Wrapper (`gradlew`) is defined on the system path

## Android code analysis command-line syntax

Use gradlew to scan Android projects, which is similar to using Gradle except that you use the Gradle Wrapper. For information about how to translate your Android project using the Gradle Wrapper, see "Using Gradle integration" on page 125.

## Filtering issues detected in Android layout files

If your Android project contains layout files (used to design the user interface), your project files might include `R.java` source files that are automatically generated by Android Studio. When you scan the project, Fortify Static Code Analyzer can detect issues associated with these layout files.

OpenText recommends that Issues reported in any layout file be included in your standard audit so you can carefully determine if any of them are false positives. After you identify issues in layout files that you are not interested in, you can filter them out as described in "Filtering the analysis" on page 178. You can filter out the issues based on the Instance ID.

# Chapter 11: Translating Go code

This section describes how to translate Go code. Fortify Static Code Analyzer supports analysis of Go code on Windows, Linux, and macOS.

This section contains the following topics:



## Go command-line syntax

For the best results, your project must be compilable and you must have all required dependencies available.

The following entities are excluded from the translation (and the scan):

- Vendor folder
- All projects defined by any `go.mod` files in subfolders, except the project defined by the `go.mod` file under the %PROJECT_ROOT%
- All files with the `_test.go` suffix (unit tests)

The basic command-line syntax to translate Go code is:

```
sourceanalyzer -b <build_id> [-gopath <dir>] [-goroot <dir>] <files>
```

## Go command-line options

The following table describes the command-line options that are specifically for translating Go code.

| Go option | Description |
|---|---|
| `-gotags <go_build_tags>` | Specifies a comma-separated list of custom build tags for a Go project.<br>This is equivalent to the `-tags` option for the `go` command. For more information, see "Including custom Go build tags" on page 87.<br>**Equivalent property name:**<br>`com.fortify.sca.gotags` |

| Go option | Description |
|---|---|
| `-gopath <dir>` | Specifies the value of the GOPATH environment variable to use for translating a Go project. If this option is not specified, then Fortify Static Code Analyzer uses the existing value of the GOPATH system environment variable.<br><br>You must specify the gopath directory as an absolute path. The following examples are valid values for *<dir>*:<br><br>`/home/projects/go_workspace/my_proj`<br>`C:\projects\go_workspace\my_proj`<br><br>The following example is an invalid value for *<dir>*:<br><br>`go_workspace/my_proj`<br><br>If this option and the GOPATH system environment variable is not set, then the gopath defaults to a subdirectory named `go` in the user's home directory (`$HOME/go` on Linux and `%USERPROFILE%\go` on Windows), unless that directory contains a Go distribution.<br><br>When using modules, the GOPATH environment variable is not required to resolve package imports as described in the go compiler command documentation. However, GOPATH still determines the output directory to use when downloading missing module dependencies.<br><br>**Note:** Fortify Static Code Analyzer does not fully support older Go projects that rely solely on the GOPATH environment variable to resolve package imports.<br><br>**Equivalent property name:**<br>`com.fortify.sca.GOPATH` |
| `-goroot <dir>` | Specifies the location of the Go installation. If this option is not specified, the GOROOT system environment variable is used.<br><br>If this option is not specified and the GOROOT system environment variable is not set, then Fortify Static Code Analyzer uses the Go compiler included in the Fortify Static Code Analyzer installation.<br><br>**Equivalent property name:**<br>`com.fortify.sca.GOROOT` |

| Go option | Description |
|---|---|
| `-goproxy <url>` | Specifies one or more comma-separated proxy URLs. You can also specify `direct` or `off` (to disable network usage).<br>If this option is not specified and the GOPROXY system environment variable is not set, then Fortify Static Code Analyzer uses `https://proxy.golang.org,direct`.<br>**Equivalent property name:**<br>`com.fortify.sca.GOPROXY` |

**See also**

"Go properties" on page 204

# Including custom Go build tags

If your Go project includes files that require custom build tags, then you can include these build tags in the Fortify Static Code Analyzer translation using the `-gotags` option. For example:

```
sourceanalyzer -b MyProject -gotags release "src/**/*.go"
```

The Fortify Static Code Analyzer `-gotags` option does not allow you to override automatic build tags for the operating system, architecture, or Go version (for example, `//go:build linux`, `//go:build arm`, `//go:build go1.21`). To translate your Go project for a different operating system or architecture, set the appropriate cross-compile targets in the GOOS and GOARCH environment variables. To set a specific Go version, specify the path for the Go SDK version in the GOROOT environment variable or the `-goroot` option.

# Resolving dependencies

Fortify Static Code Analyzer supports two dependency management systems built into Go:

- Modules

  Fortify Static Code Analyzer downloads all required dependencies using the native Go toolchain. If access to the internet is restricted on the machine where you run Fortify Static Code Analyzer, then do one of the following:

  - If you are using an artifact management system such as Artifactory, set the GOPROXY environment variable or use the `-goproxy` option described in "Go command-line options" on page 85.
  - Download all required dependencies using modules and vendoring.

- GOPATH dependency resolution

  If you are using a third-party dependency management system such as dep, you must download all dependencies before you start the translation.

# Chapter 12: Translating Dart and Flutter code

This section describes how to translate Dart and Flutter code. Fortify Static Code Analyzer supports analysis of Dart and Flutter code on Windows and Linux.

This section contains the following topics:



## Dart and Flutter translation prerequisites

The following are the prerequisites for translating Dart and Flutter projects:

- Make sure that you have a supported Dart SDK (for Dart-only projects) and the Flutter SDK (for Flutter projects) installed on your system. See the *Fortify Software System Requirements* document for the supported Dart and Flutter SDK versions.
- Download the project dependencies by running one of the following commands:
  - For Flutter projects, use `flutter pub get`.
  - For Dart-only projects, use `dart pub get` .

  For example, to download the dependencies for a Flutter project that has the project root `myproject`, run the following commands:

```
cd myproject

flutter pub get
```

**Important!** If the project includes nested packages with different `pubspec.yaml` files, you must run `dart pub get` or `flutter pub get` for each package root.

**Important!** Make sure that the following are included in the project directory:

- The `pubspec.yaml` file, which specifies the dependencies
- The `.dart_tool` directory, which includes the `package_config.json` file automatically generated by the `pub` tool

# Dart and Flutter command-line syntax

The basic command-line syntax to translate Dart and Flutter code is:

```
sourceanalyzer –b <build_id> <translation_options> <dirs>
sourceanalyzer –b <build_id> <translation_options> <files>
```

# Dart and Flutter command-line examples

To translate a Dart or Flutter project with the `my_app` project root directory:

```
sourceanalyzer -b MyProject my_app/
```

To translate the `a_widget.dart` file in the `my_app` project root directory:

```
sourceanalyzer -b MyProject my_app/a_widget.dart
```

To translate all dart source files in the `my_dart_proj` directory:

```
sourceanalyzer -b MyProject "my_dart_proj/**/*.dart"
```

# Chapter 13: Translating Ruby code

This section contains the following topics:



# Ruby command-line syntax

The basic command-line syntax to translate Ruby code is:

```
sourceanalyzer -b <build_id> <file>
```

where *<file>* is the name of the Ruby file you want to scan. To include multiple Ruby files, separate them with a space, as shown in the following example:

```
sourceanalyzer -b <build_id> file1.rb file2.rb file3.rb
```

In addition to listing individual Ruby files, you can use the asterisk (*) wildcard to select all Ruby files in a specified directory. For example, to find all the Ruby files in a directory called `src`, use the following `sourceanalyzer` command:

```
sourceanalyzer -b <build_id> src/*.rb
```

**Note:** When you translate Ruby code, make sure that you specify all source files together in one invocation. Fortify Static Code Analyzer does not support adding new files to the file list associated with the build ID on subsequent invocations.

## Ruby command-line options

The following table describes the Ruby translation options.

| Ruby option | Description |
|---|---|
| `-ruby-path <dirs>` | Specifies one or more paths to directories that contain Ruby libraries (see "Adding libraries " on the next page)<br>**Equivalent property name:**<br>`com.fortify.sca.RubyLibraryPaths` |

| Ruby option | Description |
| --- | --- |
| `-rubygem-path <dirs>` | Specifies the path(s) to a RubyGems location (see "Adding gem paths" below)<br>**Equivalent property name:**<br>`com.fortify.sca.RubyGemPaths` |

**See also**

"Ruby properties" on page 204

# Adding libraries

If your Ruby source code requires a specific library, add the Ruby library to the `sourceanalyzer` command. Include all ruby libraries that are installed with ruby gems. For example, if you have a `utils.rb` file that resides in the `/usr/share/ruby/myPersonalLibrary` directory, then add the following to the `sourceanalyzer` command:

```
-ruby-path /usr/share/ruby/myPersonalLibrary
```

Separate multiple libraries with semicolons (Windows) or colons (non-Windows). The following is an example of the option on non-Windows system:

```
-ruby-path /path/one:/path/two:/path/three
```

# Adding gem paths

To add all RubyGems and their dependency paths, import all RubyGems. To obtain the Ruby gem paths, run the `gem env` command. Under **GEM PATHS**, look for a directory similar to:

```
/home/myUser/gems/ruby-version
```

This directory contains another directory called `gems`, which contains directories for all the gem files installed on the system. For this example, use the following in your command line:

```
-rubygem-path /home/myUser/gems/ruby-version/gems
```

If you have multiple `gems` directories, separate them with semicolons (Windows) or colons (non-Windows ) such as:

```
-rubygem-path /path/to/gems:/another/path/to/more/gems
```

**Note:** On Windows systems, separate the `gems` directories with a semicolon.

# Chapter 14: Translating COBOL code

The COBOL translation runs on Windows systems only and supports modern COBOL dialects. Alternatively, you can use the legacy COBOL translation (see "Using Legacy COBOL translation" on page 96).

For a list of supported technologies for translating COBOL code, see the *Fortify Software System Requirements* document. Fortify Static Code Analyzer does not currently support custom rules for COBOL applications.

**Note:** To scan COBOL with Fortify Static Code Analyzer, you must have a Fortify Static Code Analyzer license file that specifically includes COBOL scanning capabilities. Contact Customer Support for more information about how to obtain the required license file.

This section contains the following topics:

# Preparing COBOL source and copybook files for translation

Before you can analyze a COBOL program, you must copy the following program components to the Windows system where you run Fortify Static Code Analyzer:

- COBOL source code

  OpenText strongly recommends that your COBOL source code files have extensions `.CBL`, `.cbl`, `.COB`, or `.cob`. If your source code files do not have extensions or have non-standard extensions, you must follow the instructions in "Translating COBOL source files without file extensions" on the next page and "Translating COBOL source files with arbitrary file extensions" on the next page.
- All copybook files that the COBOL source code uses

  This includes All SQL INCLUDE files that the COBOL source code references (a SQL INCLUDE file is technically a copybook file)

  **Important!** The copybook files must have the extension `.CPY` or `.cpy`.

  If your COBOL source code contains:

  ```
  COPY FOO
  ```

  or

  ```
  EXEC SQL INCLUDE FOO END-EXEC
  ```

  then `FOO` is the name of a COBOL copybook and the corresponding copybook file has the name `FOO.CPY` or `FOO.cpy`.

OpenText recommends that you place your COBOL source code files in a directory called `sources` and your copybook files in a directory called `copybooks`. Create these directories at the same level.

# COBOL command-line syntax

The basic syntax used to translate a single COBOL source code file is:

```
sourceanalyzer -b <build_id> <path>
```

The basic syntax used to scan a translated COBOL program and save the analysis results in an FPR file is:

```
sourceanalyzer -b <build_id> -scan -f <results>.fpr
```

**See also**

"Specifying files and directories" on page 146

## Translating COBOL source files without file extensions

If you have COBOL source files (not copybook files) retrieved from a mainframe without `.COB` or `.CBL` file extensions (which is typical for COBOL file names), then you must include the following in the translation command line:

```
-noextension-type COBOL
```

The following example command translates COBOL source code without file extensions:

```
sourceanalyzer -b MyProject -noextension-type COBOL -copydirs copybooks
sources
```

## Translating COBOL source files with arbitrary file extensions

If you have COBOL source files with an arbitrary extension `.xyz`, then you must include the following in the translation command line:

```
-Dcom.fortify.sca.fileextensions.xyz=COBOL
```

You must also include the expression `*.xyz` in the file or directory specifier, if any (see "Specifying files and directories" on page 146).

## COBOL command-line options

The following table describes the COBOL command-line options. To use legacy COBOL translation, see "Legacy COBOL translation command-line options" on the next page.

| COBOL option | Description |
|---|---|
| `-copydirs <dirs>` | Specifies one or more semicolon-separated directories where Fortify Static Code Analyzer looks for copybook files.<br>**Equivalent property name:**<br>`com.fortify.sca.CobolCopyDirs` |
| `-dialect <dialect>` | Specifies the COBOL dialect. The valid values for *<dialect>* are `COBOL390` and `MICROFOCUS`. The dialect value is case insensitive. The default value is `COBOL390`.<br>**Equivalent property name:**<br>com.fortify.sca.CobolDialect |
| `-checker-` | Specifies one or more semicolon-separated COBOL checker directives. |

| COBOL option | Description |
|---|---|
| `directives <directives>` | **Note:** This option is intended for advanced users of OpenText™ Server Express.<br>**Equivalent property name:**<br>`com.fortify.sca.CobolCheckerDirectives` |

# Using Legacy COBOL translation

Use the legacy COBOL translation if either of the following is true:

- You run Fortify Static Code Analyzer on a non-Windows operating system.

  For supported non-Windows platforms and architectures, see the *Fortify Software System Requirements* document.
- Your COBOL dialect is different than what is supported by the default COBOL translation (see the `-dialect` option in "COBOL command-line options" on the previous page).

Prepare the COBOL source code and copybook files as described in "Preparing COBOL source and copybook files for translation" on page 94 and use the command-line syntax described in "COBOL command-line syntax" on page 94. Note that the legacy COBOL translation accepts copybook files with or without file extensions. If the copybook files have file extensions, use the `-copy-extensions` command-line option (see "Legacy COBOL translation command-line options" below).

## Legacy COBOL translation command-line options

The following table describes the command-line options for the legacy COBOL translation.

| Legacy COBOL option | Description |
|---|---|
| `-cobol-legacy` | Specifies translation of COBOL code using legacy COBOL translation. This option is required to enable legacy COBOL translation.<br>**Equivalent Property Name:**<br>`com.fortify.sca.CobolLegacy` |
| `-copydirs <dirs>` | Specifies one or more semicolon- or colon-separated directories where Fortify Static Code Analyzer looks for copybook files.<br>**Equivalent Property Name:**<br>`com.fortify.sca.CobolCopyDirs` |
| `-copy-extensions <ext>` | Specifies one or more semicolon- or colon-separated copybook file |

| Legacy COBOL option | Description |
|---|---|
| | extensions.<br><br>**Equivalent Property Name:**<br>`com.fortify.sca.CobolCopyExtensions` |
| `-fixed-format` | Specifies fixed-format COBOL to direct Fortify Static Code Analyzer to only look for source code between columns 8–72 in all lines of code. The default is free-format.<br><br>IBM Enterprise COBOL code is typically fixed-format. The following are indications that you might need the `-fixed-format` option:<br>• The COBOL translation appears to hang indefinitely<br>• Fortify Static Code Analyzer reports numerous parsing errors in the COBOL translation<br><br>**Equivalent Property Name:**<br>`com.fortify.sca.CobolFixedFormat` |

# Chapter 15: Translating Salesforce Apex and Visualforce code

This section contains the following topics:



## Apex and Visualforce translation prerequisites

To translate Apex and Visualforce projects, make sure that all the source code to scan is available on the same machine where you have installed Fortify Static Code Analyzer.

To scan your custom Salesforce app, download it to your local computer from your Salesforce organization (org) where you develop and deploy it. The downloaded version of your app consists of:

- Apex classes in files with the `.cls` extension
- Visualforce web pages in files with the `.page` extension
- Apex code files called database “trigger” functions in files with the `.trigger` extension
- Visualforce component files in files with the `.component` extension
- Objects in files with the `.object` extension

Use the Ant Migration Tool available on the Salesforce website to download your app from your org in the Salesforce cloud to your local computer. Make sure that the project manifest files are set up correctly for the specified target in your `build.xml` file. For example, the following `package.xml` manifest file provides Fortify Static Code Analyzer with all classes, custom objects, pages, and components.

```
<?xml version="1.0" encoding="UTF-8"?>
<Package xmlns=http://soap.sforce.com/2006/04/metadata>
    <types>
        <members>*</members>
        <name>ApexClass</name>
    </types>
    <types>
        <members>*</members>
        <name>ApexTrigger</name>
    </types>
    <types>
        <members>*</members>
        <name>ApexPage</name>
    </types>
    <types>
        <members>*</members>
        <name>ApexComponent</name>
    </types>
    <types>
        <members>*</members>
        <name>CustomObject</name>
    </types>
    <version>55.0</version>
</Package>
```

Configure the retrieve targets using the Ant Migration Tool documentation. If your organization uses any apps from the app exchange, make sure that these are downloaded as packaged targets.

# Apex and Visualforce command-line syntax

The basic command-line syntax to translate Apex and Visualforce code is:

```
sourceanalyzer -b <build_id> <files>
```

where *<files>* is an Apex or Visualforce file or a path to the source files.

**Important!** Supported file extensions for the source files are: `.cls`, `.component`, `.trigger`, `.object`, and `.page`.

# Chapter 16: Translating other languages and configurations

This section contains the following topics:



# Analyzing Solidity code

The basic command-line syntax to translate and scan Solidity code is:

```
sourceanalyzer -b <build_id> <files>

sourceanalyzer -b <build_id> -scan -f <results>.fpr
```

## Importing dependencies

Fortify Static Code Analyzer translation only supports import statements for files with relative and absolute paths. Import statements for libraries is not supported.

## Managing compiler versions

Fortify Static Code Analyzer downloads compilers that are referenced in the code with the pragma statement from the Solidity compiler repository. By default, Fortify Static Code Analyzer downloads Solidity compilers to `${flight.workdir}/solidity`.

If a file does not contain a pragma statement, then the default of `^0.8.0` is used. You can specify different default compiler version to use in the analysis by including the `flight.solidity.defaultCompilerVersion` property on the command line. The version you specify must exist in the Solidity compiler repository. For example:

```
sourceanalyzer -b MyProject ./

sourceanalyzer -b MyProject -scan -
Dflight.solidity.defaultCompilerVersion=0.8.16 -f MyResults.fpr
```

If a proxy is required for the connection to download Solidity compilers, include the proxy information with `-Dhttps.proxyHost` and `-Dhttps.proxyPort`. For example:

```
sourceanalyzer -b MyProject ./

sourceanalyzer -b MyProject -scan -Dhttps.proxyHost=MyProxyHost -
Dhttps.proxyPort=1234 -f MyResults.fpr
```

You can add `flight.solidity.defaultCompilerVersion` to the `fortify-sca.properties` file.

**See also**

"Properties files" on page 184

# Translating PHP code

The syntax to translate a single PHP file named `MyPHP.php` is shown in the following example:

```
sourceanalyzer -b <build_id> MyPHP.php
```

To translate a file where the source or the `php.ini` file entry includes a relative path name (starts with `./` or `../`), consider setting the PHP source root as shown in the following example:

```
sourceanalyzer -php-source-root <path> -b <build_id> MyPHP.php
```

For more information about the `-php-source-root` option, see the description in "PHP command-line options" on the next page.

**Note:** When you translate PHP code, make sure that you specify all source files together in one invocation. Fortify Static Code Analyzer does not support adding new files to the file list

associated with the build ID on subsequent invocations.

## PHP command-line options

The following table describes the PHP-specific command-line options.

| PHP option | Description |
|---|---|
| `-php-source-root <path>` | Specifies an absolute path to the project root directory. The relative path name first expands from the current directory. If the file is not found, then the path expands from the specified PHP source root directory.<br>**Equivalent property name:**<br>`com.fortify.sca.PHPSourceRoot` |
| `-php-version <version>` | Specifies the PHP version. The default version is 8.2. For a list of valid versions, see the *Fortify Software System Requirements* document.<br>**Equivalent property name:**<br>`com.fortify.sca.PHPVersion` |

**See also**

"PHP properties" on page 206

# Translating ABAP code

ABAP code translation requires additional preparation steps to extract the code from the SAP database and prepare it for scanning. See "Importing the transport request" on the next page for more information. This section assumes you have a basic understanding of SAP and ABAP.

To translate ABAP code, the Fortify ABAP Extractor program downloads source files to the presentation server, and optionally, starts Fortify Static Code Analyzer. You need to use an account with permission to download files to the local system and execute operating system commands.

Because the extractor program is executed online, you might receive a `max dialog work process time reached` exception message if the volume of source files selected for extraction exceeds the allowable process run time. To work around this, download large projects as a series of smaller Extractor tasks. For example, if your project consists of four different packages, download each package separately into the same project directory. If the exception occurs frequently, work with your SAP Basis administrator to increase the maximum time limit (`rdisp/max_wprun_time`).

When a PACKAGE is extracted from ABAP, the Fortify ABAP Extractor extracts everything from `TDEVC` with a `parentcl` field that matches the package name. It then recursively extracts everything else from `TDEVC` with a `parentcl` field equal to those already extracted from `TDEVC`. The field extracted from `TDEVC` is `devclass`.

The devclass values are treated as a set of program names and handled the same way as a program name, which you can provide.

Programs are extracted from TRDIR by comparing the name field with either:

- The program name specified in the selection screen
- The list of values extracted from TDEVC if a package was provided

The rows from TRDIR are those for which the name field has the given program name and the expression LIKEprogramname is used to extract rows.

This final list of names is used with READ REPORT to get code out of the SAP system. This method reads classes and methods out as well as merely REPORTS, for the record.

Each READ REPORT call produces a file in the temporary folder on the local system. Fortify Static Code Analyzer translates and scans this set of files to produce an FPR file that you can open with Fortify Audit Workbench.

**See also**

"ABAP properties" on page 206

## INCLUDE processing

As source code is downloaded, the Fortify ABAP Extractor detects INCLUDE statements in the source. When found, it downloads the include targets to the local machine for analysis.

## Importing the transport request

To scan ABAP code, you need to import the Fortify ABAP Extractor transport request on your SAP Server. You can find the transport request in *<sca_install_dir>*/Tools/SAP_Extractor.zip.

The Fortify ABAP Extractor package, SAP_Extractor.zip, contains the following files:

- K900XXX.S9S (where the “XXX” is the release number)
- R900XXX.S9S (where the “XXX” is the release number)

These files make up the SAP transport request that you must import into your SAP system from outside your local Transport Domain. Have your SAP administrator or an individual authorized to install transport requests on the system import the transport request.

The S95 files contain a program, a transaction (YSCA), and the program user interface. After you import them into your system, you can extract your code from the SAP database and prepare it for Fortify Static Code Analyzer scanning.

**Installation note**

The Fortify ABAP Extractor transport request is supported on a system running SAP release 7.02, SP level 0006. If you run a different SAP version and you get the transport request import error: Install release does not match the current version, then the transport request installation has failed.

To try to resolve this issue, perform the following steps:

1. Re-run the transport request import.
   The Import Transport Request dialog box opens.
2. Select the **Options** tab.
3. Select the **Ignore Invalid Component Version** check box.
4. Complete the import procedure.

If this does not resolve the issue or if your system runs on an SAP version with a different table structure, OpenText recommends that you export your ABAP file structure using your own technology so that Fortify Static Code Analyzer can scan the ABAP code.

## Adding Fortify Static Code Analyzer to your Favorites list

Adding Fortify Static Code Analyzer to your Favorites list is optional, but doing so can make it quicker to access and start Fortify Static Code Analyzer scans. The following steps assume that you use the user menu in your day-to-day work. If your work is done from a different menu, add the Favorites link to the menu that you use. Before you create the Fortify Static Code Analyzer entry, make sure that the SAP server is running and you are in the SAP Easy Access area of your web-based client.

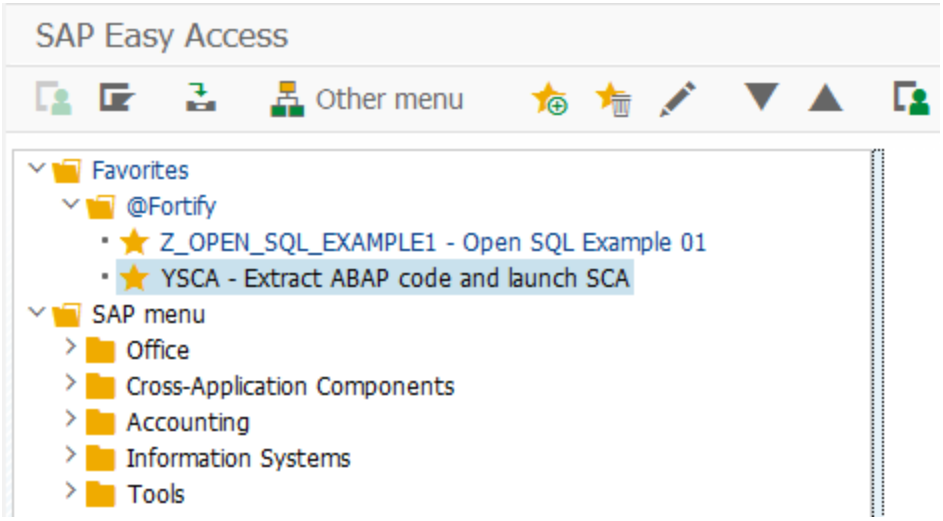
To add Fortify Static Code Analyzer to your Favorites list:

1. From the **SAP Easy Access** menu, type `S000` in the transaction box.
   The **SAP Menu** opens.
2. Right-click the **Favorites** folder and select **Insert transaction**.
   The **Manual entry of a transaction** dialog box opens.
3. Type `YSCA` in the **Transaction Code** box.
4. Click the green check mark icon.
   The **Extract ABAP code and launch SCA** item appears in the **Favorites** list.
5. Click the **Extract ABAP code and launch SCA** link to start the Fortify ABAP Extractor.

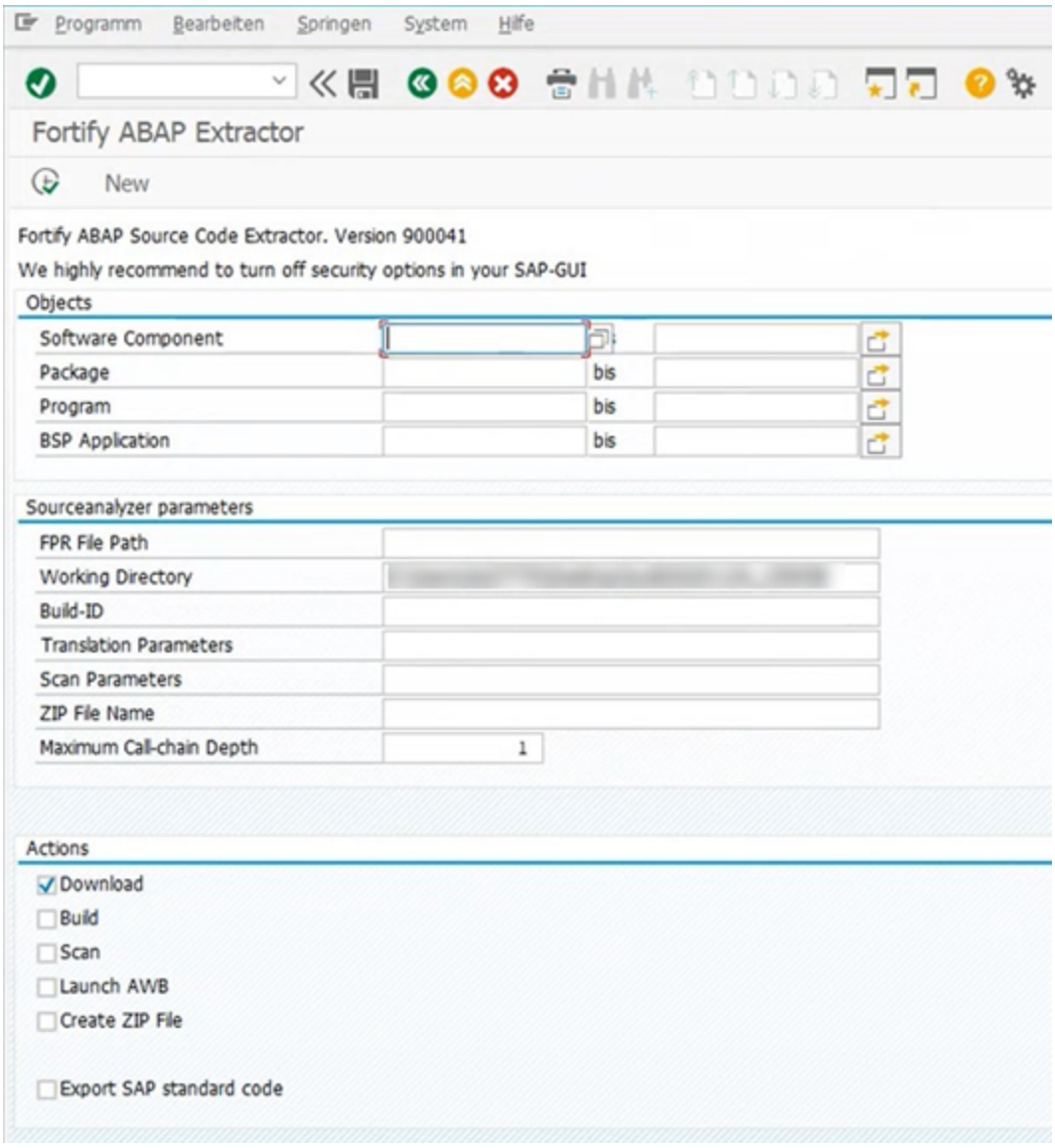
## Running the Fortify ABAP Extractor

To run the Fortify ABAP Extractor:

1. Start the Fortify ABAP Extractor from the **Favorites** link, the transaction code, or manually start the Extractor object.

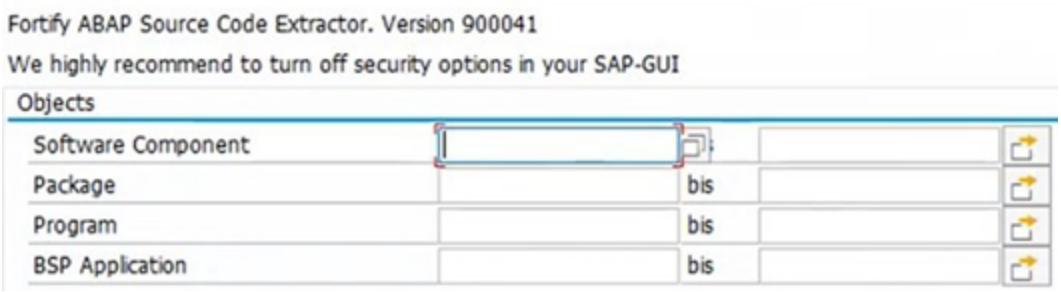



This opens the Fortify ABAP Extractor.

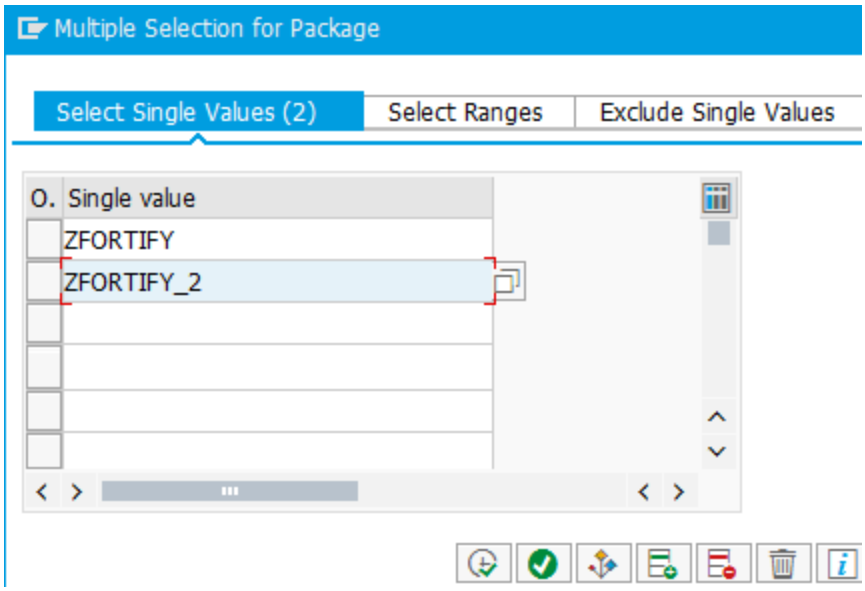



2. Select the code to download.

   Provide the start and end name for the range of software components, packages, programs, or BSP applications that you want to scan.




**Note:** You can specify multiple objects or ranges.




3. Provide the Fortify Static Code Analyzer-specific parameters described in the following table.

| Field | Description |
|---|---|
| FPR File Path | (Optional) Type or select the directory where you want to store the scan results file (FPR). Include the name for the FPR file in the path name. You must provide the FPR file path to automatically scan the downloaded code on the same machine where you are running the extraction process. |
| Working Directory | Type or select the directory where you want to store the extracted source code. |
| Build-ID | (Optional) Type the build ID for the scan. Fortify Static Code Analyzer uses the build ID to identify the translated source code, which is necessary to scan the code. You must specify the build ID to automatically translate the downloaded code on the same machine where you are running the extraction process. |
| Translation Parameters | (Optional) Type any additional Fortify Static Code Analyzer command-line translation options. You must specify translation options to automatically translate the downloaded code on the same machine where you are running the extraction process or to customize the translation options. |
| Scan Parameters | (Optional) Type any Fortify Static Code Analyzer command-line scan options. You must specify scan options to scan the downloaded code automatically on the same machine where you are running the extraction process or to customize the scan options. |

| Field | Description |
|---|---|
| ZIP File Name | (Optional) Type a ZIP file name if you want your output in a compressed package. |
| Maximum Call-chain Depth | A global SAP-function F is not downloaded unless F was explicitly selected or unless F can be reached through a chain of function calls that start in explicitly-selected code and whose length is this number or less. OpenText recommends that you do not specify a value greater than 2 unless directed to do so by Customer Support. |

4. Provide action information described in the following table.

| Field | Description |
|---|---|
| Download | Select the **Download** check box to have Fortify Static Code Analyzer download the source code extracted from your SAP database. |
| Build | Select the **Build** check box to have Fortify Static Code Analyzer translate all downloaded ABAP code and store it using the specified build ID. This action requires that you have an installed version of Fortify Static Code Analyzer on the machine where you are running the Fortify ABAP Extractor. It is often easier to move the downloaded source code to a system where Fortify Static Code Analyzer is installed. |
| Scan | Select the **Scan** check box to have Fortify Static Code Analyzer run a scan of the specified build ID. This action requires that the translate (build) action was previously performed. This action requires that you have an installed version of Fortify Static Code Analyzer on the machine where you are running the Fortify ABAP Extractor. It is often easier to move the downloaded source code to a predefined Fortify Static Code Analyzer machine. |
| Launch AWB | Select the **Launch AWB** check box to start Fortify Audit Workbench and open the specified FPR file. |
| Create ZIP File | Select the **Create ZIP File** check box to compress the output. You can also manually compress the output after the source code is extracted from your SAP database. |
| Export SAP standard code | Select the **Export SAP standard code** check box to export SAP standard code as well as custom code. |

5. Click **Execute**.

## Uninstalling the Fortify ABAP Extractor

To uninstall the ABAP extractor:

1. In ABAP Workbench, open the Object Navigator.
2. Select package Y_FORTIFY_ABAP.
3. Expand the **Programs** tab.
4. Right-click the following element, and then select **Delete**.
   - Program: Y_FORTIFY_SCA

# Translating Flex and ActionScript

The basic command-line syntax to translate ActionScript is:

```
sourceanalyzer -b <build_id> -flex-libraries <libs> <files>
```

where:

*<libs>* is a semicolon-separated (Windows) or a colon-separated (non-Windows) list of library names to which you want to "link" and *<files>* are the files to translate.

## Flex and ActionScript command-line options

Use the following command-line options to translate Flex files. You can also specify this information in the properties configuration file (`fortify-sca.properties`) as noted in each description.

| Flex and ActionScript option | Description |
|---|---|
| `-flex-sdk-root` *<dir>* | Specifies the location of the root of a valid Flex SDK. This directory must contain a frameworks folder that contains a `flex-config.xml` file. It must also contain a `bin` folder that contains an MXMLC executable.<br>**Equivalent property name:**<br>`com.fortify.sca.FlexSdkRoot` |
| `-flex-libraries` *<libs>* | Specifies a semicolon-separated (Windows) or a colon-separated (non-Windows) list of library names to which you want to link. In most cases, this list includes `flex.swc`, `framework.swc`, and `playerglobal.swc` (usually found in `frameworks/libs/` in your Flex SDK root). |

| Flex and ActionScript option | Description |
|---|---|
| | **Note:** You can specify SWC or SWF files as Flex libraries (SWZ is not currently supported).<br>**Equivalent property name:**<br>`com.fortify.sca.FlexLibraries` |
| `-flex-source-roots <dirs>` | Specifies a semicolon-separated (Windows) or a colon-separated (non-Windows) list of root directories where MXML sources are located.<br>Normally, these contain a subfolder named `com`.<br>For example, if the Flex source root specified is `foo/bar/src`, then `foo/bar/src/com/fortify/manager/util/Foo.mxml` is transformed into an object named `com.fortify.manager.util.Foo` (an object named `Foo` in the package `com.fortify.manager.util`).<br>**Equivalent property name:**<br>`com.fortify.sca.FlexSourceRoots` |

**Note:** The `-flex-sdk-root` and `–flex-source-roots` options are primarily for MXML translation, and are optional if you are scanning pure ActionScript. Use `–flex-libraries` for to resolve all ActionScript linked libraries.

Fortify Static Code Analyzer translates MXML files into ActionScript, and then runs them through an ActionScript parser. The generated ActionScript is simple to analyze; not rigorously correct like the Flex runtime model. Consequently, you might get parse errors with MXML files. For instance, the XML parsing might fail, translation to ActionScript might fail, and the parsing of the resulting ActionScript might also fail. If you see any errors that do not have a clear connection to the original source code, notify Customer Support.

**See also**

"Flex and ActionScript properties" on page 207

## ActionScript command-line examples

The following examples provide command-line syntax to translation ActionScript.

**Example 1**

The following example is for a simple application that contains only one MXML file and a single SWF library (`MyLib.swf`):

```
sourceanalyzer -b MyFlexApp -flex-libraries lib/MyLib.swf -flex-sdk-root
/home/myself/flex-sdk/ -flex-source-roots . my/app/FlexApp.mxml
```

This identifies the location of the libraries to include and the Flex SDK and the Flex source root locations. The single MXML file, located in `/my/app/FlexApp.mxml`, results in the translation of the MXML application as a single ActionScript class called `FlexApp` and located in the `my.app` package.

**Example 2**

The following example is for an application in which the source files are relative to the `src` directory. It uses a single SWF library, `MyLib.swf`, and the Flex and framework libraries from the Flex SDK:

```
sourceanalyzer -b MyFlexProject -flex-sdk-root /home/myself/flex-sdk/
-flex-source-roots src/ -flex-libraries lib/MyLib.swf "src/**/*.mxml"
"src/**/*.as"
```

This example locates the Flex SDK and uses file specifiers to include the `.as` and `.mxml` files in the `src` folder. It is not necessary to explicitly specify the `.SWC` files located in the `–flex-sdk-root`, although this example does so for the purposes of illustration. Fortify Static Code Analyzer automatically locates all `.SWC` files in the specified Flex SDK root, and it assumes that these are libraries intended for use in translating ActionScript or MXML files.

**Example 3**

In this example, the Flex SDK root and Flex libraries are specified in the properties file because typing the information for each sourceanalyzer run is time consuming and the data does not change often. Divide the application into two sections and store them in folders: a main section folder and a modules folder. Each folder contains a `src` folder where the paths start. File specifiers contain wild cards to pick up all the `.mxml` and `.as` files in both `src` folders. An `MXML` file in `main/src/com/foo/util/Foo.mxml` is translated as an ActionScript class named `Foo` in the package `com.foo.util`, for example, with the source roots specified here:

```
sourceanalyzer -b MyFlexProject -flex-source-roots main/src:modules/src
"./main/src/**/*.mxml" "./main/src/**/*.as" "./modules/src/**/*.mxml"
"./modules/src/**/*.as"
```

# Handling resolution warnings

To see all warnings that were generated during translation, type the following command before you start the scan phase:

```
sourceanalyzer -b <build_id> -show-build-warnings
```

## ActionScript warnings

You might receive a message similar to the following:

```
The ActionScript front end was unable to resolve the following imports:
a.b at y.as:2. foo.bar at somewhere.as:5. a.b at foo.mxml:8.
```

This error occurs when Fortify Static Code Analyzer cannot find all the required libraries. You might need to specify additional SWC or SWF Flex libraries (using the `-flex-libraries` option or the `com.fortify.sca.FlexLibraries` property) so that Fortify Static Code Analyzer can complete the analysis.

# Translating ColdFusion code

To treat undefined variables in a CFML page as tainted, uncomment the following line in *<sca_install_dir>*`/Core/config/fortify-sca.properties`:

```
#com.fortify.sca.CfmlUndefinedVariablesAreTainted=true
```

This instructs the Dataflow Analyzer to watch out for register-globals-style vulnerabilities. However, enabling this property interferes with Dataflow Analyzer findings in which a variable in an included page is initialized to a tainted value in an earlier-occurring included page.

## ColdFusion command-line syntax

The basic command-line syntax to translate ColdFusion source code is:

```
sourceanalyzer -b <build_id> -source-base-dir <dir> <files> | <file_specifiers>
```

where:

- *<build_id>* specifies a build ID for the project
- *<dir>* specifies the root directory of the web application
- *<files>* | *<file_specifiers>* specifies the CFML source code files

  For a description of how to use *<file_specifiers>*, see "Specifying files and directories" on page 146.

**Note:** Fortify Static Code Analyzer calculates the relative path to each CFML source file with the `-source-base-dir` directory as the starting point. Fortify Static Code Analyzer uses these relative paths when it generates instance IDs. If you move the entire application source tree to a different directory, the Fortify Static Code Analyzer- generated instance IDs remain the same if you specify an appropriate parameter for the `-source-base-dir` option.

## ColdFusion (CFML) command-line options

The following table describes the CFML options.

| ColdFusion option | Description |
|---|---|
| `-source-base-dir <web_app_root_dir> <files> \| <file_specifiers>` | The web application root directory.<br>**Equivalent property name:**<br>`com.fortify.sca.SourceBaseDir` |

**See also**

"ColdFusion (CFML) properties" on page 207

# Analyzing SQL

On Windows (and Linux for .NET projects only), Fortify Static Code Analyzer assumes that files with the `.sql` extension are T-SQL rather than PL/SQL. If you have PL/SQL files with the `.sql` extension on Windows, you must configure Fortify Static Code Analyzer to treat them as PL/SQL.

The basic syntax to translate and scan PL/SQL is:

```
sourceanalyzer -b <build_id> -sql-language PL/SQL <files>

sourceanalyzer -b <build_id> -sql-language PL/SQL -scan -f <results>.fpr
```

Alternatively, you can change the default behavior for files with the `.sql` extension. In the `fortify-sca.properties` file, set the `com.fortify.sca.fileextensions.sql` property to `PLSQL`.

The basic syntax to translate and scan T-SQL is:

```
sourceanalyzer -b <build_id> -sql-language TSQL <files>

sourceanalyzer -b <build_id> -scan -f <results>.fpr
```

**See also**

"SQL properties" on page 208

## PL/SQL command-line example

The following example commands translate and scan two PL/SQL files:

```
sourceanalyzer -b MyProject -sql-language PL/SQL x.pks y.pks
```

```
sourceanalyzer -b MyProject -sql-language PL/SQL -scan -f MyResults.fpr
```

The following example commands translate and scan all PL/SQL files in the `sources` directory:

```
sourceanalyzer -b MyProject -sql-language PL/SQL "sources/**/*.pks"
sourceanalyzer -b MyProject -sql-language PL/SQL -scan -f MyResults.fpr
```

### T-SQL command-line example

The following example translates two T-SQL files:

```
sourceanalyzer -b MyProject x.sql y.sql
```

The following example translates all T-SQL files in the `sources` directory:

```
sourceanalyzer -b MyProject "sources\**\*.sql"
```

**Note:** This example assumes the `com.fortify.sca.fileextensions.sql` property in `fortify-sca.properties` is set to `TSQL`, which is the property's default value.

## Translating Scala code

Translating Scala code requires the following:

- The Scala compiler plugin

  You can download this plugin from the Maven Central Repository.
- A Lightbend license file

  This license file is included with the Fortify Static Code Analyzer installation in the *<sca_install_dir>*`/plugins/lightbend` directory

For instructions on how set up the license and translate Scala code, see Lightbend documentation Fortify SCA for Scala.

**Important!** If your project contains source code other than Scala, you must translate the Scala code using the Scala Fortify compiler plugin, and then translate other source code with sourceanalyzer using the same build ID before you run the analysis phase.

# Translating Infrastructure as Code (IaC)

Fortify Static Code Analyzer translates Azure Resource Manager (ARM), Bicep, AWS CloudFormation, and HCL templates.

**Note:** HCL analysis support is specific to Terraform and supported cloud provider Infrastructure as Code (IaC) configurations.

For best results, make sure that the template files are deployment valid. The templates must not contain:

- Validation errors that are static and locally detectable (for example, type errors or references to undefined variables or functions).
- Predeployment errors that occur during template interpretation, but before any resources are deployed or modified (for example, invalid array indexing operations).
- Deployment errors that occur in the cloud (for example, dynamically referencing a non-existent resource).

OpenText recommends that AWS CloudFormation file name extensions are `.json`, `.yaml`, `.template`, or `.txt`. Fortify Static Code Analyzer supports other extensions only if they are not commonly used by other languages or file types (such as `.java` or `.html`).

By default, Fortify Static Code Analyzer translates files with the HCL extensions `.hcl` and `.tf`.

## ARM translation command-line examples

Translate an ARM template:

```
sourceanalyzer -b MyProject ArmTemplate.json
```

Translate all ARM templates in a directory:

```
sourceanalyzer -b MyProject "src/**/*.json"
```

## Bicep translation command-line examples

Translate a single Bicep template:

```
sourceanalyzer -b MyProject BicepTemplate.bicep
```

Translate all Bicep templates in a directory:

```
sourceanalyzer -b MyProject "src/**/*.bicep"
```

## AWS CloudFormation translation command-line examples

Translate AWS CloudFormation templates that have different extensions:

```
sourceanalyzer -b MyProject CFTemplateA.template CFTemplateB.yaml
CFTemplateC.json CFTemplateD.customext
```

Translate all AWS CloudFormation templates in a directory that have the `.template` extension:

```
sourceanalyzer -b MyProject "src/**/*.template"
```

Translate all AWS CloudFormation templates in a directory that have either the `.json` or `.yaml` extension:

```
sourceanalyzer -b MyProject "src/**/*.json" "src/**/*.yaml"
```

## HCL translation command-line examples

Translate two HCL templates with different extensions:

```
sourceanalyzer -b MyProject HCLTemplateA.hcl HCLTemplateB.tf
```

Translate all HCL templates in a directory:

```
sourceanalyzer -b MyProject "src/**/*.tf" "src/**/*.hcl"
```

**See also**

"Translating JSON" below

"Translating YAML" on the next page

# Translating JSON

By default, Fortify Static Code Analyzer translates files with the JSON extension `.json` as JSON. The following example translates a JSON file:

```
sourceanalyzer -b MyProject x.json
```

The following example translates all JSON files in the `sources` directory:

```
sourceanalyzer -b MyProject "sources/**/*.json"
```

# Translating YAML

By default, Fortify Static Code Analyzer translates files with the YAML extensions `.yaml` and `.yml`. The following example translates two YAML files with different file extensions:

```
sourceanalyzer -b MyProject x.yaml y.yml
```

The following example translates all YAML files in the `sources` directory:

```
sourceanalyzer -b MyProject "sources/**/*.yaml" "sources/**/*.yml"
```

# Translating Dockerfiles

By default, Fortify Static Code Analyzer translates the following files as Dockerfiles: `Dockerfile*`, `dockerfile*`, `*.Dockerfile`, and `*.dockerfile`.

**Note:** You can modify the file name extension used to detect Dockerfiles using the `com.fortify.sca.fileextensions` property. See "Translation and analysis phase properties" on page 186.

Fortify Static Code Analyzer accepts the following escape characters in Dockerfiles: backslash (`\`) and backquote (`` ` ``). If the escape character is not set in the Dockerfile, then Fortify Static Code Analyzer assumes that the backslash is the escape character.

The syntax to translate a directory that contains Dockerfiles is shown in the following example:

```
sourceanalyzer -b <build_id> <dir>
```

If the Dockerfile is malformed, Fortify Static Code Analyzer writes an error to the log file to indicate that the file cannot be parsed and skips the analysis of the Dockerfile. The following is an example of the error written to the log:

```
Unable to parse dockerfile ProjA.Dockerfile, error on Line 1:20: mismatched
input '\n' expecting {LINE_EXTEND, WHITESPACE}

Unable to parse config file
C:/Users/jsmith/MyProj/docker/dockerfile/ProjA.Dockerfile
```

# Translating ASP/VBScript virtual roots

Fortify Static Code Analyzer allows you to handle ASP virtual roots. For web servers that use virtual directories as aliases that map to physical directories, Fortify Static Code Analyzer enables you to use

an alias.

For example, you can have virtual directories named `Include` and `Library` that refer to the physical directories `C:\WebServer\CustomerOne\inc` and `C:\WebServer\CustomerTwo\Stuff`, respectively.

The following example shows the ASP/VBScript code for an application that uses *virtual* includes:

```
<!--#include virtual="Include/Task1/foo.inc"-->
```

For this example, the previous ASP code refers to the file in the following physical location:

```
C:\Webserver\CustomerOne\inc\Task1\foo.inc
```

The real directory replaces the virtual directory name `Include` in this example.

## Accommodating virtual roots

To provide the mapping of each virtual directory to Fortify Static Code Analyzer, you must set the `com.fortify.sca.ASPVirtualRoots.name_of_virtual_directory` property in your Fortify Static Code Analyzer command-line invocation as shown in the following example:

```
sourceanalyzer -Dcom.fortify.sca.ASPVirtualRoots.<virtual_directory>=<full_
path_to_corresponding_physical_directory>
```

**Note:** On Windows, if the physical path includes spaces, you must enclose the property setting in quotes:

```
sourceanalyzer "-Dcom.fortify.sca.ASPVirtualRoots.<virtual_
directory>=<full_path_to_corresponding_physical_directory>"
```

To expand on the example in the previous section, pass the following property value to Fortify Static Code Analyzer:

```
-Dcom.fortify.sca.ASPVirtualRoots.Include="C:\WebServer\CustomerOne\inc"

-Dcom.fortify.sca.ASPVirtualRoots.Library="C:\WebServer\CustomerTwo\Stuff"
```

This maps `Include` to `C:\WebServer\CustomerOne\inc` and `Library` to `C:\WebServer\CustomerTwo\Stuff`.

When Fortify Static Code Analyzer encounters the `#include` directive:

```
<!-- #include virtual="Include/Task1/foo.inc" -->
```

Fortify Static Code Analyzer determines if the project contains a physical directory named `Include`. If there is no such physical directory, Fortify Static Code Analyzer looks through its runtime properties and finds the `-Dcom.fortify.sca.ASPVirtualRoots.Include="C:\WebServer\CustomerOne\inc"` setting. Fortify Static Code Analyzer then looks for this file: `C:\WebServer\CustomerOne\inc\Task1\foo.inc`.

Alternatively, you can set this property in the `fortify-sca.properties` file located in *<sca_install_dir>*`\Core\config`. You must escape the backslash character (\) in the path of the physical directory as shown in the following example:

```
com.fortify.sca.ASPVirtualRoots.Library=C:\\WebServer\\CustomerTwo\\Stuff

com.fortify.sca.ASPVirtualRoots.Include=C:\\WebServer\\CustomerOne\\inc
```

**Note:** The previous version of the ASPVirtualRoot property is still valid. You can use it on the Fortify Static Code Analyzer command line as follows:
`-Dcom.fortify.sca.ASPVirtualRoots=C:\WebServer\CustomerTwo\Stuff;C:\WebServer\CustomerOne\inc`

This prompts Fortify Static Code Analyzer to search through the listed directories in the order specified when it resolves a virtual include directive.

## Using virtual roots example

You have a file as follows:

```
C:\files\foo\bar.asp
```

To specify this file, use the following include:

```
<!-- #include virtual="/foo/bar.asp">
```

Then set the virtual root in the `sourceanalyzer` command as follows:

```
-Dcom.fortify.sca.ASPVirtualRoots=C:\files\foo
```

This strips the `/foo` from the front of the virtual root. If you do not specify `foo` in the `com.fortify.sca.ASPVirtualRoots` property, then Fortify Static Code Analyzer looks for `C:\files\bar.asp` and fails.

The sequence to specify virtual roots is as follows:

1. Remove the first part of the path in the source.
2. Replace the first part of the path with the virtual root as specified on the command line.

# Classic ASP command-line example

To translate a single file classic ASP written in VBScript named `MyASP.asp`, type:

```
sourceanalyzer -b mybuild "MyASP.asp"
```

# VBScript command-line example

To translate a VBScript file named `myApp.vb`, type:

```
sourceanalyzer -b mybuild "myApp.vb"
```

# Chapter 17: Integrating the analysis into a build

You can integrate the analysis into supported build tools.

This section contains the following topics:



## Build integration

You can translate entire projects with a single operation. Prefix your original build operation with the `sourceanalyzer` command followed by the Fortify Static Code Analyzer options.

The basic command-line syntax to translate a complete project is:

```
sourceanalyzer -b <build_id> [<sca_options>] <build_tool> [<build_tool_
options>]
```

where *<build_tool>* is the name of your build tool, such as make, gmake, msbuild, devenv, or xcodebuild. See the *Fortify Software System Requirements* document for a list of supported build tools. Fortify Static Code Analyzer executes your build tool and intercepts all compiler operations to collect the specific command line used for each input.

**Note:** Fortify Static Code Analyzer only processes the compiler commands that the build tool executes. If you do not clean your project before you execute the build, then Fortify Static Code Analyzer only processes those files that the build tool re-compiles.

For information about how to integrate with Xcodebuild, see "iOS code analysis command-line syntax" on page 83. For information about integration with MSBuild, see "Translating Visual Studio projects" on page 62.

Successful build integration requires that the build tool:

- Executes a Fortify Static Code Analyzer-supported compiler
- Executes the compiler on the operating system path search, not with a hardcoded path (This requirement does not apply to xcodebuild integration.)
- Executes the compiler, rather than executing a sub-process that then executes the compiler

If you cannot meet these requirements in your environment, see "Modifying a build script to start the analysis" below.

**Make example**

If you build your project with the following build commands:

```
make clean
make
make install
```

then you can simultaneously translate and compile the entire project with the following example commands:

```
make clean
sourceanalyzer -b MyProject make
make install
```

# Modifying a build script to start the analysis

As an alternative to build integration, you can modify your build script to prefix each compiler, linker, and archiver operation with the `sourceanalyzer` command. For example, a makefile often defines variables for the names of these tools:

```
CC=gcc
CXX=g++
LD=ld
AR=ar
```

You can prepend the tool references in the makefile with the `sourceanalyzer` command and the appropriate options.

```
CC=sourceanalyzer -b MyProject gcc
CXX=sourceanalyzer -b MyProject g++
LD=sourceanalyzer -b MyProject ld
AR=sourceanalyzer -b MyProject ar
```

When you use the same build ID for each operation, Fortify Static Code Analyzer automatically combines each of the separately-translated files into a single translated project.

# Integrating with Ant

You can translate Java source files for projects that use an Ant build file. You can apply this integration on the command line without modifying the Ant `build.xml` file. When the build runs, Fortify Static Code Analyzer intercepts all `javac` task invocations and translates the Java source files as they are compiled. Make sure that you pass any properties to Ant by adding them to the ANT_OPTS environment variable. Do not include them in the sourceanalyzer command.

**Note:** You must translate any JSP files, configuration files, or any other non-Java source files that are part of the application in a separate step.

To use the Ant integration, make sure that the `sourceanalyzer` executable is in the PATH environment variable.

Prepend your Ant command-line with the `sourceanalyzer` command as follows:

```
sourceanalyzer -b <build_id> [<sca_options>] ant [<ant_options>]
```

For example, to translate a Java project and exclude a file from the translation:

```
sourceanalyzer -b MyProjectA -logfile MyProjectA.log -exclude src/module-
info.java ant
```

# Integrating with Bazel

You can translate projects written in Java or Python that are built with Bazel. When the build runs, Fortify Static Code Analyzer translates the source files as they are compiled. See the *Fortify Software System Requirements* document for supported Bazel versions.

Make sure the following requirements are met before you run the Fortify Static Code Analyzer Bazel integration:

- Your Bazel build runs without errors.
- The `sourceanalyzer` executable is included in the PATH environment variable.
  After the build is complete, always run the Fortify Static Code Analyzer analysis phase with the same version of Fortify Static Code Analyzer that is included in the PATH environment variable.

To run the translation phase for the configured Java or Python, go to the Bazel workspace directory, and then run the Fortify Static Code Analyzer command with the target you want to build. Prepend the Bazel build command line with the `sourceanalyzer` command as follows:

```
sourceanalyzer -b <build_id> <sca_options> bazel build <target>
```

**Note:** If you have multiple Fortify Static Code Analyzer installations, make sure that the version you want to use for your Bazel projects is defined before all other Fortify Static Code Analyzer versions included in the PATH environment variable.

## Bazel build integration examples

Translate a project for a specific target:

```
sourceanalyzer -b MyProjectA bazel build //proja:my-prj
```

Translate target `abc` in package `proja/abc`:

```
sourceanalyzer -b MyProjectA bazel build //proja/abc
```

or

```
sourceanalyzer -b MyProjectA bazel build //proja/abc:abc
```

Translate all targets in the package `proja/abc`:

```
sourceanalyzer -b MyProjectA bazel build //proja/abc:all
```

Translate all targets within the `projb/` directory:

```
sourceanalyzer -b MyProjectB bazel build //projb/...
```

Specify a specific JDK version for the translation:

```
sourceanalyzer -b MyProjectC -jdk 17 bazel build //projc:my-java-prj
```

Translate a project and exclude a file from the translation:

```
sourceanalyzer -b MyProjectC -exclude C:\test\MY-JAVA-
APP\src\main\java\com\example\HelpContent.java bazel build //projc:my-java-
prj
```

Specify Python project dependencies for the translation:

```
sourceanalyzer -b MyProjectD -python-path /usr/local/lib/python3.6/ bazel
build //projd:my-python-app
```

Fortify Static Code Analyzer Bazel integration does not support multiple targets and related actions such as excluding targets.

**See also**

"Java command-line options" on page 51

"Python command-line options" on page 77

# Integrating with CMake

On non-Windows systems, you can translate projects that are built with CMake by incorporating a JSON compilation database in the Fortify Static Code Analyzer command. This is only supported for Makefile and Ninja generators (see the CMake Reference Documentation for more information).

To integrate Fortify Static Code Analyzer with a CMake build:

1. Generate a `compile_commands.json` file for your CMake project.

   Add `-DCMAKE_EXPORT_COMPILE_COMMANDS=yes` to the `cmake` configure command. For example:

   ```
   cmake -G Ninja -DCMAKE_EXPORT_COMPILE_COMMANDS=yes
   ```

2. Include the JSON compilation database in your `sourceanalyzer` command as follows:

   ```
   sourceanalyzer -b <build_id> compile_commands.json
   ```

# Integrating with Gradle

Fortify Static Code Analyzer provides translation integration with projects that are built with Gradle. You can either integrate without modifying your build script or you can use the Fortify Static Code Analyzer Gradle plugin.

## Using Gradle integration

You can translate projects that are built with Gradle without any modification of the `build.gradle` file. When the build runs, Fortify Static Code Analyzer translates the source files as they are compiled. Alternatively, you can use the Fortify Static Code Analyzer Gradle Plugin to perform the analysis from within your Gradle build script (see "Using the Gradle plugin" on page 127).

See the *Fortify Software System Requirements* document for platforms and languages supported specifically for Gradle integration. Any files in the project in unsupported languages for Gradle integration are not translated (with no error reporting). These files are therefore not analyzed, and any existing potential vulnerabilities can go undetected.

To integrate Fortify Static Code Analyzer into your Gradle build, make sure that the `sourceanalyzer` executable is included in the PATH environment variable. Always use the `sourceanalyzer` executable from the system PATH for all Gradle commands to build the project.

**Note:** If you have multiple Fortify Static Code Analyzer installations, make sure that the version you want to use for your Gradle projects is defined before all other Fortify Static Code Analyzer versions included in the PATH environment variable.

Prepend the Gradle command line with the `sourceanalyzer` command as follows:

```
sourceanalyzer -b <build_id> <sca_options> gradle [<gradle_options>]
<gradle_tasks>
```

### Gradle integration examples

```
sourceanalyzer -b MyProject gradle clean build
sourceanalyzer -b MyProject gradle --info assemble
```

If your build file name is different than `build.gradle`, then include the build file name with the `--build-file` option as shown in the following example:

```
sourceanalyzer -b MyProject gradle --build-file sample.gradle clean
assemble
```

You can also use the Gradle Wrapper (`gradlew`) as shown in the following example:

```
sourceanalyzer -b MyProject gradlew [<gradle_options>]
```

Translate a project and exclude a file from the translation:

```
sourceanalyzer -b MyProject -exclude src\test\**\* gradlew build
```

If your application uses XML or property configuration files, translate these files with a separate `sourceanalyzer` command. Use the same build ID that you used for the project files. The following are examples:

```
sourceanalyzer -b MyProject <path_to_xml_files>
sourceanalyzer -b MyProject <path_to_properties_files>
```

After Fortify Static Code Analyzer translates the project with gradle or gradlew, you can then perform the analysis phase and save the results in an FPR file as shown in the following example:

```
sourceanalyzer -b MyProject -scan -f MyResults.fpr
```

**See also**

"Using the Gradle plugin" on the next page

## Troubleshooting Gradle integration

If you use configuration caching (`--configuration-cache` option) in your Gradle build with Fortify Static Code Analyzer Gradle integration, the build reports the following messages:

```
Configuration cache problems found in this build.
```

You also might see a failure message such as the following:

```
FAILURE: Build failed with an exception...
```

You can safely ignore this failure message with respect to the Fortify Static Code Analyzer translation because the project is translated. You can verify that the project is translated using the `-show-files` option. For example:

```
sourceanalyzer -b mybuild -show-files
```

## Using the Gradle plugin

The Fortify Static Code Analyzer installation includes a Gradle plugin located in *<sca_install_dir>*`/plugins/gradle`. To use the Fortify Static Code Analyzer Gradle Plugin, you need to first configure the plugin for your Java or Kotlin project and then use the plugin to analyze your project. The Gradle plugin provides three Fortify Static Code Analyzer tasks for the analysis: sca.clean, sca.translate, and sca.scan. See the *Fortify Software System Requirements* document for platforms and languages supported specifically for Fortify Static Code Analyzer Gradle plugin.

**Note:** If you have multiple Fortify Static Code Analyzer installations, make sure that the version you want to use for your Gradle projects is defined before all other Fortify Static Code Analyzer versions included in the PATH environment variable.

To configure the Fortify Static Code Analyzer Gradle Plugin:

1. Edit the Gradle settings file to specify the path to the plugin:
   - Groovy DSL (`settings.gradle`):

     ```
     pluginManagement {
       repositories {
       gradlePluginPortal()
         maven {
           url = uri("file://<sca_plugin_path>")
         }
       }
     }
     ```

   - Kotlin DSL (`settings.gradle.kts`):

     ```
     pluginManagement {
       repositories {
         maven(url = uri("file://<sca_plugin_path>"))
         gradlePluginPortal()
       }
     }
     ```

2. Add entries to the build script as shown in the following examples:
   - Groovy DSL (`build.gradle`):

     ```
     id 'com.fortify.sca.plugins.gradlebuild' version '24.4'
     ```

     and

     ```
     SCAPluginExtension {
       buildId = "MyProject"
       options = ["-encoding", "utf-8", "-logfile", "MyProject.log",
         "-debug-verbose"]
     }
     ```

     or the following example entry excludes files from the translation:

     ```
     SCAPluginExtension {
       buildId = "MyProject"
       options = ["-encoding", "utf-8", "-logfile", "MyProject.log",
         "-debug-verbose", "-exclude", "src/test/**/*"]
     }
     ```

   - Kotlin DSL (`build.gradle.kts`):

     ```
     plugins {
       id ("com.fortify.sca.plugins.gradlebuild") version "24.4"
       ...
     }
     ```

     and

     ```
     SCAPluginExtension {
       buildId = "MyProject"
       options = listOf("-encoding", "utf-8", "-logfile", "MyProject.log",
         "-debug-verbose")
     }
     ```

     or the following example entry excludes files from the translation:

     ```
     SCAPluginExtension {
       buildId = "MyProject"
       options = listOf("-encoding", "utf-8", "-logfile", "MyProject.log",
         "-debug-verbose", "-exclude", "src/test/**/*")
     }
     ```

3. Save and close the Gradle settings and Gradle build files.

Analyze a Java or Kotlin project with following command sequence:

- To remove all existing Fortify Static Code Analyzer temporary files for an existing Java or Kotlin project build, run the following:

  ```
  gradlew sca.clean
  ```

- To run the translation phase for the configured Java or Kotlin project, run the following:

  ```
  gradlew sca.translate
  ```

- To analyze the configured Java or Kotlin project, run the following:

  ```
  gradlew sca.scan
  ```

  This task runs successfully if Fortify Static Code Analyzer has already translated the project using the Fortify Static Code Analyzer Gradle Plugin.

### Working with Java or Kotlin projects that have subprojects

If you have a Java or Kotlin multi-project build (with subprojects), then you must configure the Fortify Static Code Analyzer Gradle plugin using an `allprojects` block. This is shown in the following examples.

Groovy DSL (`build.gradle`)

```
allprojects {
  apply plugin: "com.fortify.sca.plugins.gradlebuild"
  SCAPluginExtension {
    buildId = "MyProject"
    options = ["-encoding", "utf-8", "-logfile", "MyProject.log",
      "-debug-verbose"]
    ...
  }
}
```

Kotlin DSL (`build.gradle.kts`):

```
allprojects {
  apply(plugin = "com.fortify.sca.plugins.gradlebuild")
  SCAPluginExtension {
    buildId = "MyProject"
    options = listOf("-encoding", "utf-8", "-logfile", "MyProject.log",
      "-debug-verbose")
    ...
  }
}
```

**See also**

"Using Gradle integration" on page 125

# Integrating with Maven

Fortify Static Code Analyzer includes a Maven plugin that provides a way to add the following capabilities to your Maven project builds:

- Fortify Static Code Analyzer clean, translate, scan
- Fortify Static Code Analyzer export mobile build session (MBS) for a Fortify Static Code Analyzer translated project
- Send translated code to Fortify ScanCentral SAST
- Upload results to Fortify Software Security Center

You can use the plugin directly or integrate its functionality into your build process.

## Installing and updating the Fortify Maven Plugin

The Fortify Maven Plugin is located in *<sca_install_dir>*`/plugins/maven`. This directory contains a binary and a source version of the plugin in both zip and tarball archives. To install the plugin, extract the version (binary or source) that you want to use, and then follow the instructions in the included `README.TXT` file. Perform the installation in the directory where you extracted the archive.

For information about supported versions of Maven, see the *Fortify Software System Requirements* document.

If you have a previous version of the Fortify Maven Plugin installed, then install the latest version.

### Uninstalling the Fortify Maven Plugin

To uninstall the Fortify Maven Plugin, manually delete all files from the *<maven_local_repo>*`/repository/com/fortify/ps/maven/plugin` directory.

## Testing the Fortify Maven Plugin installation

After you install the Fortify Maven Plugin, use one of the included sample files to be sure your installation works properly.

To test the Fortify Maven Plugin using the Eightball sample file:

1. Add the directory that contains the `sourceanalyzer` executable to the path environment variable.

   For example:

   ```
   export set PATH=$PATH:/<sca_install_dir>/bin
   ```

   or

   ```
   set PATH=%PATH%;<sca_install_dir>/bin
   ```

2. Type `sourceanalyzer -version` to test the path setting.
   Fortify Static Code Analyzer displays the version information if the path setting is correct.
3. Navigate to the sample Eightball directory: *<root_dir>*`/samples/EightBall`.
4. Type the following command:

   ```
   mvn com.fortify.sca.plugins.maven:sca-maven-plugin:<ver>:clean
   ```

   where *<ver>* is the version of the Fortify Maven Plugin you are using. If the version is not specified, Maven uses the latest version of the Fortify Maven Plugin installed in the local repository.

   **Note:** To see the version of the Fortify Maven Plugin, open the `pom.xml` file that you extracted in *<root_dir>* in a text editor. The Fortify Maven Plugin version is specified in the `<version>` element.

5. If the command in step 4 completed successfully, then the Fortify Maven Plugin is installed correctly. The Fortify Maven Plugin is not installed correctly if you get the following error message:

   ```
   [ERROR] Error resolving version for plugin
   'com.fortify.sca.plugins.maven:sca-maven-plugin' from the repositories
   ```

   Check the Maven local repository and try to install the Fortify Maven Plugin again.

## Using the Fortify Maven Plugin

There are two ways to perform an analysis on a maven project:

- As a Maven Plugin

  In this method, you perform the analysis tasks as goals with the `mvn` command. For example, use the following command to translate source code:

  ```
  mvn com.fortify.sca.plugins.maven:sca-maven-plugin:24.4.0:translate
  ```

  For example, use the following command to translate source code and exclude test files:

  ```
  mvn -Dfortify.sca.exclude="**/Test/*.java"
  com.fortify.sca.plugins.maven:sca-maven-plugin:24.4.0:translate
  ```

  To analyze your code this way, see the documentation included with the Fortify Maven Plugin. The following table describes where to find the documentation after you install the Fortify Maven Plugin.

  | Package type | Documentation location |
  |---|---|
  | Binary | *<root_dir>*/docs/index.html |
  | Source | *<root_dir>*/sca-maven-plugin/target/site/index.html |

- In a Fortify Static Code Analyzer build integration

  In this method, prepend the maven command used to build your project with the `sourceanalyzer` command and any Fortify Static Code Analyzer options. To analyze your files as part of a Fortify Static Code Analyzer build integration:

  a. Clean out the previous build:

  ```
  sourceanalyzer -b MyProject -clean
  ```

  b. Translate the code:

  ```
  sourceanalyzer -b MyProject [<sca_options>] [<mvn_command_with_
  options>]
  ```

  Examples:

  ```
  sourceanalyzer -b MyProject mvn package
  ```

  ```
  sourceanalyzer -b MyProject -exclude "**/Test/*.java" mvn clean
  install
  ```

See "Command-line interface" on page 134 for descriptions of available Fortify Static Code Analyzer options.

c. Run the scan and save the results in an FPR file as shown in the following example:

```
sourceanalyzer -b MyProject [<sca_scan_options>] -scan -f
MyResults.fpr
```

# Chapter 18: Command-line interface

This chapter describes general Fortify Static Code Analyzer command-line options and how to specify source files for analysis. Command-line options that are specific to a language are described in the chapter for that language.

This section contains the following topics:



## Translation options

The following table describes the translation options.

| Translation option | Description |
|---|---|
| `-b <build_id>` | Specifies a build ID. Fortify Static Code Analyzer uses a build ID to track the files that are compiled and combined as part of a build, and then later, to scan those files.<br>**Equivalent property name:**<br>`com.fortify.sca.BuildID` |
| `-disable-language <Languages>` | Specifies a colon-separated list of languages to exclude from the translation phase. The valid language values are `abap`, `actionscript`, `apex`, `cfml`, `cobol`, `configuration`, `cpp`, `dart`, `dotnet`, `golang`, `java`, `javascript`, `jsp`, `kotlin`, `objc`, `php`, `plsql`, `python`, `ruby`, `scala`, `sql`, `swift`, `tsql`, `typescript`, and `vb`.<br>**Equivalent property name:**<br>`com.fortify.sca.DISabledLanguages` |

| Translation option | Description |
|---|---|
| `-enable-language <Languages>` | Specifies a colon-separated list of languages to translate. The valid language values are `abap`, `actionscript`, `apex`, `cfml`, `cobol`, `configuration`, `cpp`, `dart`, `dotnet`, `golang`, `java`, `javascript`, `jsp`, `kotlin`, `objc`, `php`, `plsql`, `python`, `ruby`, `scala`, `sql`, `swift`, `tsql`, `typescript`, and `vb`.<br>**Equivalent property name:**<br>`com.fortify.sca.EnabledLanguages` |
| `-exclude <file_specifiers>` | Specifies the files to exclude from the translation. Files excluded from translation are also not scanned. Separate multiple file paths with semicolons (Windows) or colons (non-Windows). The following example excludes all Java files in any `Test` subdirectory.<br>`sourceanalyzer -b MyProject -cp "**/*.jar" "**/*" -exclude "**/Test/*.java"`<br>See "Specifying files and directories" on page 146 for more information on how to use file specifiers.<br>**Equivalent property name:**<br>`com.fortify.sca.exclude` |
| `-encoding <encoding_name>` | Specifies the source file encoding type. Fortify Static Code Analyzer enables you to scan a project that contains differently encoded source files. To work with a multi-encoded project, you must specify the `-encoding` option in the translation phase, when Fortify Static Code Analyzer first reads the source code file. Fortify Static Code Analyzer remembers this encoding in the build session and propagates it into the FVDL file.<br>Valid encoding names are from the `java.nio.charset.Charset`.<br>Typically, if you do not specify the encoding type, Fortify Static Code Analyzer uses `file.encoding` from the `java.io.InputStreamReader` constructor with no encoding parameter. In a few cases (for example with the ActionScript parser), Fortify Static Code Analyzer defaults to `UTF-8` encoding.<br>**Equivalent property name:**<br>`com.fortify.sca.InputFileEncoding` |
| `-nc` | When specified before a compiler command line, Fortify Static Code |

| Translation option | Description |
|---|---|
| | Analyzer translates the source file but does not run the compiler. |
| `-noextension-type <file_type>` | Specifies the file type for source files that have no extension. The valid file type values are ABAP, ACTIONSCRIPT, APEX, APEX_OBJECT, APEX_TRIGGER, ARCHIVE, ASPNET, ASP, ASPX, BITCODE, BSP, BYTECODE, CFML, COBOL, CSHARP, DART, DOCKERFILE, FLIGHT, GENERIC, GO, HOCON, HTML, INI, JAVA, JAVA_PROPERTIES, JAVASCRIPT, JSP, JSPX, KOTLIN, MSIL, MXML, OBJECT, PHP, PLSQL, PYTHON, RUBY, RUBY_ERB, SCALA, SWIFT, SWC, SWF, TLD, SQL, TSQL, TYPESCRIPT, VB, VB6, VBSCRIPT, VISUAL_FORCE, VUE, and XML. |
| `-disable-compiler-resolution` | Specifies to include build script files that have the same name as a build tool (such as gradlew) during translation as source files.<br>**Equivalent property name:**<br>`com.fortify.sca.DisableCompilerName` |
| `-project-root` | Specifies the directory to store intermediate files generated in the translation and analysis phases. Fortify Static Code Analyzer makes extensive use of intermediate files located in this project root directory. In some cases, you can achieve better performance for analysis by making sure this directory is on local storage rather than on a network drive.<br>**Equivalent property name:**<br>`com.fortify.sca.ProjectRoot` |

# Analysis options

The following table describes the analysis options.

| Analysis option | Description |
|---|---|
| `-b <build_id>` | Specifies the build ID used in a prior translation command.<br>**Equivalent property name:**<br>`com.fortify.sca.BuildID` |
| `-scan` | Causes Fortify Static Code Analyzer to perform a security analysis for the specified build ID. |

| Analysis option | Description |
|---|---|
| `-scan-policy <policy_name>` \| `-sc <policy_name>` | Specifies a scan policy for the analysis. The valid policy names are `classic`, `security`, and `devops`. For more information, see "Applying a scan policy to the analysis" on page 46.<br>**Equivalent property name:**<br>`com.fortify.sca.ScanPolicy` |
| `-analyzers <analyzer_list>` | Specifies the analyzers you want to enable with a colon- or comma-separated list of analyzers. The valid analyzer names are `buffer`, `content`, `configuration`, `controlflow`, `dataflow`, `nullptr`, `semantic`, and `structural`. You can use this option to disable analyzers that are not required for your security requirements.<br>**Equivalent property name:**<br>`com.fortify.sca.DefaultAnalyzers` |
| `-p <level>` \| `-scan-precision <level>` | Uses speed dial to scan the project with a scan precision level. The lower the scan precision level, the faster the scan performance. The valid values are `1`, `2`, `3`, and `4`. For more information, see "Configuring scan speed with speed dial" on page 160.<br>**Equivalent property name:**<br>`com.fortify.sca.PrecisionLevel` |
| `-project-root` | Specifies the directory to store intermediate files generated in the translation and analysis phases. Fortify Static Code Analyzer makes extensive use of intermediate files located in this project root directory. In some cases, you can achieve better performance for analysis by making sure this directory is on local storage rather than on a network drive.<br>**Equivalent property name:**<br>`com.fortify.sca.ProjectRoot` |
| `-project-template <file>` | Specifies the issue template file to use for the scan. This only affects scans on the local machine. If you upload the FPR to Fortify Software Security Center, it uses the issue template assigned to the application version.<br>**Equivalent property name:**<br>`com.fortify.sca.ProjectTemplate` |

| Analysis option | Description |
|---|---|
| `-quick` | Quickly scan the project for critical- and high-priority issues using the `fortify-sca-quickscan.properties` file, which provides a less in-depth analysis. By default, quick scan disables the Buffer Analyzer and the Control Flow Analyzer. In addition, it applies the Quick View filter set. For more information, see "Quick scan" on page 159.<br><br>**Equivalent property name:**<br>`com.fortify.sca.QuickScanMode` |
| `-filter <file>` | Specifies a results filter file. For more information, see "Filtering the analysis" on page 178.<br><br>**Equivalent property name:**<br>`com.fortify.sca.FilterFile` |
| `-bin <binary>` \| `-binary-name <binary>` | Specifies a subset of source files to scan. Only the source files that were linked in the named binary at build time are included in the scan. You can use this option multiple times to specify the inclusion of multiple binaries in the scan.<br><br>**Equivalent property name:**<br>`com.fortify.sca.BinaryName` |
| `-disable-default-rule-type <type>` | Disables all rules of the specified type in the default Rulepacks. You can use this option multiple times to specify multiple rule types.<br><br>The *<type>* parameter is the XML tag minus the suffix `Rule`. For example, use `DataflowSource` for DataflowSourceRule elements. You can also specify specific sections of characterization rules, such as `Characterization:Control flow`, `Characterization:Issue`, and `Characterization:Generic`.<br><br>The *<type>* parameter is case-insensitive. |
| `-no-default-issue-rules` | Disables rules in default Rulepacks that lead directly to issues. Fortify Static Code Analyzer still loads rules that characterize the behavior of functions.<br><br>**Note:** This is equivalent to disabling the following rule types: DataflowSink, Semantic, Controlflow, Structural, Configuration, Content, Statistical, Internal, and Characterization:Issue. |

| Analysis option | Description |
|---|---|
| | **Equivalent property name:**<br>`com.fortify.sca.NoDefaultIssueRules` |
| `-no-default-rules` | Disables loading of rules from the default Rulepacks. Fortify Static Code Analyzer processes the Rulepacks for description elements and language libraries, but processes no rules.<br>**Equivalent property name:**<br>`com.fortify.sca.NoDefaultRules` |
| `-no-default-source-rules` | Disables source rules in the default Rulepacks.<br>**Note:** Characterization source rules are not disabled.<br>**Equivalent property name:**<br>`com.fortify.sca.NoDefaultSourceRules` |
| `-no-default-sink-rules` | Disables sink rules in the default Rulepacks.<br>**Note:** Characterization sink rules are not disabled.<br>**Equivalent property name:**<br>`com.fortify.sca.NoDefaultSinkRules` |
| `-rules <file>\|<dir>` | Specifies a custom Rulepack or directory. You can use this option multiple times to specify multiple Rulepack files. If you specify a directory, Fortify Static Code Analyzer includes all the files in the directory with the `.bin` and `.xml` extensions.<br>**Equivalent property name:**<br>`com.fortify.sca.RulesFile` |

# Output options

The following table describes the output options. Apply all these options during the analysis phase (with the `-scan` option). You can specify the `build-label`, `build-project`, and `build-version` options during the translation phase and they are overridden if specified again for the analysis phase.

| Output option | Description |
|---|---|
| `-f <file>\|-output-file` | Specifies the file to which analysis results are written. If you do not specify an output file, Fortify Static Code Analyzer writes the output to |

| Output option | Description |
|---|---|
| *<file>* | the terminal.<br><br>**Equivalent property name:**<br>`com.fortify.sca.ResultsFile` |
| `-format` *<format>* | Controls the output format. Valid options are `fpr`, `fvdl`, `fvdl.zip`, `text`, and `auto`. The default is `auto`, which selects the output format based on the file name extension of the file provided with the `-f` option.<br><br>The FVDL is an XML file that contains the detailed Fortify Static Code Analyzer analysis results. This includes vulnerability details, rule descriptions, code snippets, command-line options used in the scan, and any scan errors or warnings.<br><br>The FPR is a package of the analysis results that includes the FVDL file as well as extra information such as a copy of the source code used in the scan, the external metadata, and custom rules (if applicable). Fortify Audit Workbench is automatically associated with the `.fpr` extension.<br><br>**Note:** If you use result certification, you must specify the `fpr` format. See the *OpenText™ Fortify Audit Workbench User Guide* for information about result certification.<br><br>You can prevent some information from being included in the FPR or FVDL file to improve scan time or output file size. See other options in this table and see "Optimizing FPR files" on page 163.<br><br>**Equivalent property name:**<br>`com.fortify.sca.Renderer` |
| `-append` | Appends results to the file specified with the `-f` option. The resulting FPR file contains the issues from the earlier scan as well as issues from the current scan. The build information and program data (lists of sources and sinks) sections are also merged. To use this option, the output file format must be `fpr` or `fvdl`. For information on the `-format` output option, see the description in this table.<br><br>The engine data, which includes Fortify Software Security Content information, command-line options, system properties, warnings, errors, and other information about the execution of Fortify Static Code Analyzer (as opposed to information about the program being analyzed), is not merged. Because engine data is not merged with |

| Output option | Description |
|---|---|
| | the `-append` option, OpenText does not certify results generated with `-append`.<br>If this option is not specified, Fortify Static Code Analyzer adds any new findings to the FPR file, and labels the older result as `previous` findings.<br>In general, only use the `-append` option when it is impossible to analyze an entire application at once.<br>**Equivalent property name:**<br>`com.fortify.sca.OutputAppend` |
| `-build-label <label>` | Specifies a label for the project to include in the analysis results. You can include this option during the translation or the analysis phase. Fortify Static Code Analyzer does not use this label for code analysis.<br>**Equivalent property name:**<br>`com.fortify.sca.BuildLabel` |
| `-build-project <project_name>` | Specifies a name for the project to include in the analysis results. You can include this option during the translation or the analysis phase. Fortify Static Code Analyzer does not use this name for code analysis.<br>**Equivalent property name:**<br>`com.fortify.sca.BuildProject` |
| `-build-version <version>` | Specifies a version for the project to include in the analysis results. You can include this option during the translation or the analysis phase. Fortify Static Code Analyzer does not use this version for code analysis.<br>**Equivalent property name:**<br>`com.fortify.sca.BuildVersion` |
| `-disable-source-bundling` | Excludes source files from the analysis results file.<br>**Equivalent property name:**<br>`com.fortify.sca.FPRDisableSourceBundling` |
| `-fvdl-no-descriptions` | Excludes the Fortify Software Security Content descriptions from the analysis results file.<br>**Equivalent property name:**<br>`com.fortify.sca.FVDLDisableDescriptions` |
| `-fvdl-no-enginedata` | Excludes engine data from the analysis results file. The engine data |

| Output option | Description |
|---|---|
| | includes Fortify Software Security Content information, command-line options, system properties, warnings, errors, and other information about the Fortify Static Code Analyzer execution.<br>**Equivalent property name:**<br>`com.fortify.sca.FVDLDisableEngineData` |
| `-fvdl-no-progdata` | Excludes program data from the analysis results file. This removes the taint source information from the Functions view in Fortify Audit Workbench.<br>**Equivalent property name:**<br>`com.fortify.sca.FVDLDisableProgramData` |
| `-fvdl-no-snippets` | Excludes the code snippets from the analysis results file.<br>**Equivalent property name:**<br>`com.fortify.sca.FVDLDisableSnippets` |

# Other options

The following table describes other options.

| Other option | Description |
|---|---|
| `@<file>` | Reads command-line options from the specified file.<br>**Note:** By default, this file uses the JVM system encoding. You can change the encoding by using the `com.fortify.sca.CmdlineOptionsFileEncoding` property specified in the `fortify-sca.properties` file. For more information about this property, see "Translation and analysis phase properties" on page 186. |
| `-h` \|<br>`-?` \|<br>`-help` | Prints a summary of the command-line options. |
| `-debug` | Includes debug information in the Static Code Analyzer Support log file, which is only useful for Customer Support to help troubleshoot. |

| Other option | Description |
|---|---|
| | **Equivalent property name:**<br>`com.fortify.sca.Debug` |
| `-debug-verbose` | This is the same as the `-debug` option, but it includes more details, specifically for parse errors.<br>**Equivalent property name:**<br>`com.fortify.sca.DebugVerbose` |
| `-debug-mem` | Includes performance information in the Static Code Analyzer Support log.<br>**Equivalent property name:**<br>`com.fortify.sca.DebugTrackMem` |
| `-verbose` | Sends verbose status messages to the console and to the Static Code Analyzer Support log file.<br>**Equivalent property name:**<br>`com.fortify.sca.Verbose` |
| `-logfile <file>` | Specifies the log file that Fortify Static Code Analyzer creates.<br>**Equivalent property name:**<br>`com.fortify.sca.LogFile` |
| `-clobber-log` | Directs Fortify Static Code Analyzer to overwrite the log file for each run of sourceanalyzer. Without this option, Fortify Static Code Analyzer appends information to the log file.<br>**Equivalent property name:**<br>`com.fortify.sca.ClobberLogFile` |
| `-quiet` | Disables the command-line progress information.<br>**Equivalent property name:**<br>`com.fortify.sca.Quiet` |
| `-version` \|<br>`-v` | Displays the Fortify Static Code Analyzer version and versions of various independent modules included with Fortify Static Code Analyzer (all other functionality is contained in Fortify Static Code Analyzer). |
| `-autoheap` | Enables automatic allocation of memory based on the physical memory available on the system. This is the default memory allocation setting. |

| Other option | Description |
|---|---|
| `-Xmx<size>M\|G` | Specifies the maximum amount of memory Fortify Static Code Analyzer uses.<br><br>Heap sizes between 32 GB and 48 GB are not advised due to internal JVM implementations. Heap sizes in this range perform worse than at 32 GB. The JVM optimizes heap sizes smaller than 32 GB. If your scan requires more than 32 GB, then you need 64 GB or more. As a guideline, assuming no other memory intensive processes are running, do not allocate more than 2/3 of the available memory.<br><br>When you specify this option, make sure that you do not allocate more memory than is physically available, because this degrades performance. As a guideline, and the assumption that no other memory intensive processes are running, do not allocate more than 2/3 of the available memory.<br><br>**Note:** Specifying this option overrides the default memory allocation obtained with the `-autoheap` option. |

# Directives

Use only one directive at a time and do not use any directive in conjunction with translation or analysis commands. Use the directives described in the following table to list information about previous translation commands.

| Directive | Description |
|---|---|
| `-clean` | Deletes all Fortify Static Code Analyzer intermediate files and build records. If you specify a build ID, only files and build records that relate to that build ID are deleted. |
| `-show-binaries` | Displays all objects created but not used in the production of any other binaries. If fully integrated into the build, it lists all the binaries produced. |
| `-show-build-ids` | Displays a list of all known build IDs. |
| `-show-build-tree` | When you scan with the `-bin` option, displays all files used to create the binary and all files used to create those files in a tree layout. If the `-bin` option is not present, the tree is displayed for each binary. |

| Directive | Description |
|---|---|
| | **Note:** This option can generate an extensive amount of information. |
| `-show-build-warnings` | Use with the `-b` option to display any errors and warnings that occurred in the translation phase on the console.<br><br>**Note:** Fortify Audit Workbench also displays these errors and warnings in the results Certification tab. |
| `-show-files` | Displays the files included in the specified build ID. When the `-bin` option is present, displays only the source files that went into the binary. |
| `-show-loc` | Use with the `-b` option to display the number of lines in the translated code. |

## LIM license directives

Fortify Static Code Analyzer provides directives to manage the usage of your LIM license. You can store or clear the LIM license pool credentials. You can also request (and release) a detached lease for offline analysis if the specified license pool permits detached leases.

**Note:** By default, Fortify Static Code Analyzer requires an HTTPS connection to the LIM server and you must have a trusted certificate. For more information, see "Adding trusted certificates" on page 40.

Use the directives described in the following table for a license managed by the LIM.

| LIM directive | Description |
|---|---|
| `-store-license-pool-credentials "<lim_url>\|<lim_pool_name>\|<lim_pool_pwd>\|<proxy_url>\|<proxy_user>\|<proxy_pwd>"` | Stores your LIM license pool credentials so that Fortify Static Code Analyzer uses the LIM for licensing. The proxy information is optional. Fortify Static Code Analyzer stores the pool password and the proxy credentials provided with this directive in the `fortify-sca.properties` file as encrypted data. If your license pool credentials change after you have installed Fortify Static Code Analyzer, you can run this directive again to save the new credentials.<br><br>Example:<br><br>`sourceanalyzer -store-license-pool-credentials` |

| LIM directive | Description |
|---|---|
| | `"https://<ip_address>:<port>\|TeamA\|mypassword"`<br>**Associated property names:**<br>`com.fortify.sca.lim.Url`<br>`com.fortify.sca.lim.PoolName`<br>`com.fortify.sca.lim.PoolPassword`<br>`com.fortify.sca.lim.ProxyUrl`<br>`com.fortify.sca.lim.ProxyUsername`<br>`com.fortify.sca.lim.ProxyPassword` |
| `-clear-license-pool-credentials` | Removes the LIM license pool credentials from the `fortify-sca.properties` file. If your license pool credentials change, you can remove them with this directive, and then use the `-store-license-pool-credentials` directive to save the new credentials. |
| `-request-detached-lease <duration>` | Requests a detached lease from the LIM license pool for exclusive use on this system for the specified duration (in minutes). This enables you to run Fortify Static Code Analyzer even when disconnected from your corporate intranet.<br>**Note:** To use this directive, the license pool must be configured to allow detached leases. |
| `-release-detached-lease` | Releases a detached lease back to the license pool. |

# Specifying files and directories

File specifiers are expressions that allow you to pass a long list of files or a directory to Fortify Static Code Analyzer using wildcard characters. Fortify Static Code Analyzer recognizes two types of wildcard characters: a single asterisk character (*) matches part of a file name, and double asterisk characters (**) recursively matches directories. You can specify one or more files, one or more file specifiers, or a combination of files and file specifiers.

*<files> | <file_dir_specifiers>*

Windows and many Linux shells automatically expand parameters that contain the asterisk character (*), so you must enclose file-specifier expressions in quotes. Also, on Windows, you can use the backslash character (\) as the directory separator instead of the forward slash (/).

**Note:** File specifiers do not apply to C, C++, or Objective-C++.

The following table describes examples of file and directory specifiers.

| File / Directory Specifier | Description |
|---|---|
| *<dir>*<br>"*<dir>*/**/*" | Matches all files in the named directory and any subdirectories or the named directory when used for a directory parameter. |
| "*<dir>*/**/Example.java" | Matches any file named `Example.java` found in the named directory or any subdirectories. |
| "*<dir>*/*.java"<br>"*<dir>*/*.jar" | Matches any file with the specified extension found in the named directory. |
| "*<dir>*/**/*.kt"<br>"*<dir>*/**/*.jar" | Matches any file with the specified extension found in the named directory or any subdirectories. |
| "*<dir>*/**/beta/**" | Matches all directories and files found in the named directory that have `beta` in the path, including `beta` as a file name. |
| "*<dir>*/**/classes/" | Matches all directories and files with the name `classes` found in the named directory and any subdirectories. |
| "**/test/**" | Matches all files in the current directory tree that have a `test` element in the path, including `test` as a file name. |
| "**/webgoat/*" | Matches all files in any `webgoat` directory in the current directory tree.<br>Matches:<br>• `/src/main/java/org/owasp/webgoat`<br>• `/test/java/org/owasp/webgoat`<br>Does not match (`assignments` directory does not match)<br>• `/test/java/org/owasp/webgoat/assignments` |

# Chapter 19: Command-line tools

Fortify Static Code Analyzer command-line tools enable you to manage Fortify Software Security Content, perform post-installation configurations, and monitor scans. These tools are located in `<sca_install_dir>/bin`. The tools for Windows are provided as `.bat` or `.cmd` files. The following table describes the command-line tools installed with Fortify Static Code Analyzer.

**Note:** By default, log files for Fortify Static Code Analyzer tools are written to the following directory:

- Windows: `C:\Users\<username>\AppData\Local\Fortify\<tool_name>-<version>\log`
- Non-Windows: `<userhome>/.fortify/<tool_name>-<version>/log`

| Tool | Description | More information |
|---|---|---|
| fortifyupdate | Compares installed security content to the current version and makes any required updates | "About updating Fortify Software Security Content" on the next page |
| FPRUtility | With this tool you can:<br>• Merge audited projects<br>• Verify FPR signatures<br>• Display information from an FPR file<br>• Combine or split source code files and audit projects into FPR files<br>• Alter an FPR | *OpenText™ Fortify Static Code Analyzer Applications and Tools Guide* |
| scapostinstall | This tool enables you to migrate properties files from a previous version of Fortify Static Code Analyzer, specify a locale, and specify a proxy server for security content updates and for Fortify Software Security Center. | "Running the post-install tool" on page 37 |
| SCAState | Provides state analysis information on the JVM during the analysis phase | "Checking the scan status with SCAState" on page 152 |

This section contains the following topics:



# About updating Fortify Software Security Content

You can use the fortifyupdate command-line tool to download the latest Fortify Secure Coding Rulepacks and metadata from OpenText.

The fortifyupdate tool gathers information about the existing security content in your Fortify Static Code Analyzer installation and contacts the Fortify Rulepack update server with this information. The server returns new or updated security content, and removes any obsolete security content from your Fortify Static Code Analyzer installation. If your installation is current, a message is displayed to that effect.

## Updating Fortify Software Security Content

Use the fortifyupdate command-line tool to either download security content or import a local copy of the security content. This tool is located in the `<sca_install_dir>/bin` directory.

The default read timeout for this tool is 180 seconds. To change the timeout setting, add the `rulepackupdate.SocketReadTimeoutSeconds` property in the `server.properties` configuration file. For more information, see the *OpenText™ Fortify Static Code Analyzer Applications and Tools Guide*.

The basic command-line syntax for fortifyupdate is shown in the following example:

```
fortifyupdate [<options>]
```

To update your Fortify Static Code Analyzer installation with the latest Fortify Secure Coding Rulepacks and external metadata from the Fortify Rulepack update server, type the following command:

```
fortifyupdate
```

To update security content from the local system:

```
fortifyupdate -import <my_local_rules>.zip
```

To update security content from a Fortify Software Security Center server using credentials:

```
fortifyupdate -url <ssc_url> -sscUser <username> -sscPassword <password>
```

## fortifyupdate command-line options

The following table describes the fortifyupdate options.

| fortifyupdate option | Description |
|---|---|
| `-acceptKey` | Specifies to accept the public key. When this is specified, you are not prompted to provide a public key. Use this option to accept the public key if you update Fortify Software Security Content from a non-standard location with the `-url` option. |
| `-acceptSSLCertificate` | Specifies to use the SSL certificate provided by the server. |
| `-import <file>.zip` | Imports the ZIP file that contains security content. By default, Rulepacks are imported into the *<sca_install_dir>*`/Core/config/rules` directory. |
| `-coreDir <dir>` | Specifies a core directory where fortifyupdate stores the update. If this is not specified, the fortifyupdate performs the update in the *<sca_install_dir>*.<br><br>**Important!** Make sure that you copy the contents of the *<sca_install_dir>*`/config/keys` folder and paste it to a `config/keys` folder in this directory before you run `fortifyupdate`. |
| `-includeMetadata` | Specifies to only update external metadata. |
| `-includeRules` | Specifies to only update Rulepacks. |
| `-locale <locale>` | Specifies a locale. English is the default if no security content exists for the specified locale. The valid values are:<br><br>• `en` (English)<br>• `es` (Spanish)<br>• `ja` (Japanese)<br>• `ko` (Korean)<br>• `pt_BR` (Brazilian Portuguese) |

| fortifyupdate option | Description |
|---|---|
| | • `zh_CN` (Simplified Chinese)<br>• `zh_TW` (Traditional Chinese)<br>**Note:** The values are ***not*** case-sensitive.<br>Alternatively, you can specify a default locale for security content updates in the `fortify.properties` configuration file. For more information, see the *OpenText™ Fortify Static Code Analyzer Applications and Tools Guide*. |
| `-proxyhost <host>` | Specifies a proxy server network name or IP address. |
| `-proxyport <port>` | Specifies a proxy server port number. |
| `-proxyUsername <username>` | Specifies a user name if the proxy server requires authentication. |
| `-proxyPassword <password>` | Specifies the password if the proxy server requires authentication. |
| `-showInstalledRules` | Displays the currently installed Rulepacks including any custom rules and custom metadata. |
| `-showInstalledExternalMetadata` | Displays the currently installed external metadata. |
| `-url <url>` | Specifies a URL from which to download the security content. The default URL is `https://update.fortify.com` or the value set for the `rulepackupdate.server` property in the `server.properties` configuration file.<br>For more information about the `server.properties` configuration file, see the *OpenText™ Fortify Static Code Analyzer Applications and Tools Guide*.<br>You can download the security content from a Fortify Software Security Center server by providing a Fortify Software Security Center URL. |
| Specify one of the following types of credentials if you update security content from Fortify Software Security Center with the `-url` option: | |

| fortifyupdate option | Description |
|---|---|
| `-sscUsername`<br>`-sscPassword` | Specifies a Fortify Software Security Center user account by user name and password. |
| `-sscAuthToken` | Specifies a Fortify Software Security Center authentication token of type UnifiedLoginToken, CIToken, or ToolsConnectToken. |

# Checking the scan status with SCAState

Use the SCAState tool to see up-to-date state analysis information during the analysis phase.

To check the state:

1. Start a scan.
2. Open another command window.
3. Type the following at the command prompt:

```
SCAState [<options>]
```

**See also**

"SCAState command-line options" below

## SCAState command-line options

The following table describes the SCAState options.

| SCAState option | Description |
|---|---|
| `-a` \|<br>`--all` | Displays all available information. |
| `-debug` | Displays information that is useful to debug SCAState behavior. |
| `-ftd` \|<br>`--full-thread-dump` | Prints a thread dump for every thread. |
| `-h` \|<br>`--help` | Displays the help information for the SCAState tool. |
| `-hd` *<filename>* \|<br>`--heap-dump` *<filename>* | Specifies the file to which the heap dump is written. The file is interpreted relative to the remote scan's working directory; this is |

| SCAState option | Description |
|---|---|
| | not necessarily the same directory where you are running SCAState. |
| `-liveprogress` | Displays the ongoing status of a running scan. This is the default. If possible, this information is displayed in a separate terminal window. |
| `-nogui` | Causes the Fortify Static Code Analyzer state information to display in the current terminal window instead of in a separate window. |
| `-pi` \| `--program-info` | Displays information about the source code being scanned, including how many source files and functions it contains. |
| `-pid` *<process_id>* | Specifies the currently running Fortify Static Code Analyzer process ID. Use this option if there are multiple Fortify Static Code Analyzer processes running simultaneously.<br>To obtain the process ID on Windows systems:<br>1. Open a command window.<br>2. At the command prompt, type `tasklist`. A list of processes is displayed.<br>3. Find the `java.exe` process in the list and note its PID.<br>To find the process ID on Linux systems:<br>• At the command prompt, type `ps aux \| grep sourceanalyzer`. |
| `-progress` | Displays scan information up to the point at which the command is issued. This includes the elapsed time, the current phase of the analysis, and the number of results already obtained. |
| `-properties` | Displays configuration settings (this does not include sensitive information such as passwords). |
| `-scaversion` | Displays the Fortify Static Code Analyzer version number for the sourceanalyzer that is currently running. |
| `-td` \| `--thread-dump` | Prints a thread dump for the main scanning thread. |

| SCAState option | Description |
|---|---|
| `-timers` | Displays information from the timers and counters that are instrumented in Fortify Static Code Analyzer. |
| `-version` | Displays the SCAState version. |
| `-vminfo` | Displays the following statistics that JVM standard MXBeans provides: ClassLoadingMXBean, CompilationMXBean, GarbageCollectorMXBeans, MemoryMXBean, OperatingSystemMXBean, RuntimeMXBean, and ThreadMXBean. |
| <none> | Displays scan progress information (this is the same as `-progress`). |

**Note:** Fortify Static Code Analyzer writes Java process information to the location of the TMP system environment variable. On Windows systems, the TMP system environment variable location is `C:\Users\<username>\AppData\Local\Temp`. If you change this TMP system environment variable to point to a different location, SCAState cannot locate the `sourceanalyzer` Java process and does not return the expected results. To resolve this issue, change the TMP system environment variable to match the new TMP location. OpenText recommends that you run SCAState as an administrator on Windows.

# Chapter 20: Improving performance

This chapter provides guidelines and tips to optimize memory usage and performance when analyzing different types of codebases with Fortify Static Code Analyzer.

This section contains the following topics:



## Antivirus software

The use of antivirus software can negatively impact Fortify Static Code Analyzer performance. If you notice long scan times, OpenText recommends that you temporarily exclude the internal Fortify Static Code Analyzer files from your antivirus software scan. You can also do the same for the directories where the source code resides, however the performance impact on the analysis is less than with the internal directories.

By default, Fortify Static Code Analyzer creates internal files in the following location:

- Windows: `c:\Users\<username>\AppData\Local\Fortify\sca<version>`
- Non-Windows: `<userhome>/.fortify/sca<version>`

where *<version>* is the version of Fortify Static Code Analyzer you are using.

# Hardware considerations

The variety of source code makes accurate predictions of memory usage and scan times impossible. The factors that affect memory usage and performance consists of many different factors including:

- Code type
- Codebase size and complexity
- Ancillary languages used (such as JSP, JavaScript, and HTML)
- Number of vulnerabilities
- Type of vulnerabilities (analyzer used)

OpenText developed the following set of "best guess" hardware recommendations based on real-world application scan results. The following table lists these recommendations based on the complexity of the application. In general, increasing the number of available cores might improve scan times.

| Application complexity | CPU cores | RAM (GB) | Description |
|---|---|---|---|
| Simple | 4 | 16 | A standalone system that runs on a server or desktop such as a batch job or a command-line tool. |
| Medium | 8 | 32 | A standalone system that works with complex computer models such as a tax calculation system or a scheduling system. |
| Complex | 16 | 128 | A three-tiered business system with transactional data processing such as a financial system or a commercial website. |
| Very Complex | 32 | 256 | A system that delivers content such as an application server, database server, or content management system. |

**Note:** TypeScript and JavaScript scans increase the analysis time significantly. If the total lines of code in an application consist of more than 20% TypeScript or JavaScript, use the next highest recommendation.

The *Fortify Software System Requirements* document describes the system requirements. However, for large and complex applications, Fortify Static Code Analyzer requires more capable hardware. This includes:

- **Disk I/O**—Fortify Static Code Analyzer is I/O intensive and therefore the faster the hard drive, the more savings on the I/O transactions. OpenText recommends a 7,200 RPM drive, although a 10,000 RPM drive (such as the WD Raptor) or an SSD drive is better.
- **Memory**—See "Memory tuning" on page 170 for more information about how to determine the amount of memory required for optimal performance.
- **CPU**—OpenText recommends a 2.1 GHz or faster processor.

# Sample scans

These sample scans were performed using Fortify Static Code Analyzer version 24.4.0 on dedicated virtual machines. These scans were run with Fortify Software Security Content 24.4. The following table shows the scan times you can expect for several common open-source projects.

| Language | Project name | Translation time (mm:ss) | Analysis (scan) time (mm:ss) | Total issues | LOC | System configuration |
|---|---|---|---|---|---|---|
| .NET (C#) | SharpZipLib | 01:43 | 02:08 | 789 | 41,773 | Windows VM with 8 CPUs and 32 GB of RAM |
| C/C++ | nasm 0.98.38 | 00:36 | 04:15 | 761 | 35,960 | Linux VM with 8 CPUs and 32 GB of RAM |
| Java | WebGoat 8 | 00:36 | 01:00 | 253 | 23,412 | |
| Java | WordPress for Android | 00:14 | 01:33 | 534 | 35,167 | |
| JavaScript | three.js | 06:00 | 16:00 | 280 | 678,332 | |
| PHP | CakePHP | 00:19 | 02:53 | 4,572 | 136,463 | |
| PHP | phpBB 3 | 00:37 | 01:58 | 1,325 | 206,733 | |
| Python 3 | numpy-1.13.3 | 02:16 | 08:00 | 216 | 562,731 | |
| Swift | MediaBrowser | 00:20 | 01:45 | 10 | 17,611 | macOS VM with 4 CPUs and 16 GB of RAM |
| TypeScript | rxjs-7.8.1 | 29:00 | 06:00 | 86 | 248,558 | Linux VM with 8 CPUs and 32 GB of RAM |

# Tuning options

Fortify Static Code Analyzer can take a long time to process complex projects. The time is spent in different phases:

- Translation
- Analysis

Fortify Static Code Analyzer can produce large analysis result files (FPRs), which can take a long time to audit and upload to Fortify Software Security Center. This is referred to as the following phase:

- Audit/Upload

The following table lists tips on how to improve performance in the different time-consuming phases.

| Phase | Option | Description | More information |
|---|---|---|---|
| Translation | `-export-build-session`<br>`-import-build-session` | Translate and scan on different machines | "Mobile build sessions" on page 44 |
| Analysis | `-quick` | Run a quick scan | "Quick scan" on the next page |
| Analysis | `-scan-precision` | Set the scan precision | "Configuring scan speed with speed dial" on page 160 |
| Analysis | `-bin` | Scan the files related to a binary | "Breaking down codebases" on page 161 |
| Analysis | `-Xmx<size>M\|G` | Set maximum heap size | "Memory tuning" on page 170 |
| Analysis | `-Xss<size>M\|G` | Set stack size for each thread | "Memory tuning" on page 170 |
| Analysis<br>Audit/Upload | `-filter <file>` | Apply a filter using a filter file | "Using filter files" on page 163 |
| Analysis<br>Audit/Upload | `-disable-source-bundling` | Exclude source files from the FPR file | "Excluding source code from the FPR" on page 164 |

# Quick scan

Quick scan mode provides a way to quickly scan your projects for critical- and high-priority issues. Fortify Static Code Analyzer performs the scan faster by reducing the depth of the analysis. It also applies the Quick View filter set. Quick scan settings are configurable. For more details about the configuration of quick scan mode, see "fortify-sca-quickscan.properties" on page 213.

Quick scans are a great way to get many applications through an assessment so that you can quickly find issues and begin remediation. The performance improvement you get depends on the complexity and size of the application. Although the scan is faster than a full scan, it does not provide as robust a result set. OpenText recommends that you run full scans whenever possible.

## Limiters

The depth of the Fortify Static Code Analyzer analysis sometimes depends on the available resources. Fortify Static Code Analyzer uses a complexity metric to trade off these resources with the number of vulnerabilities that it can find. Sometimes, this means giving up on a particular function when it does not look like Fortify Static Code Analyzer has enough resources available.

Fortify Static Code Analyzer enables the user to control the “cutoff” point by using Fortify Static Code Analyzer limiter properties. The different analyzers have different limiters. You can run a predefined set of these limiters using a quick scan. See the "fortify-sca-quickscan.properties" on page 213 for descriptions of the limiters.

To enable quick scan mode, use the `-quick` option with `-scan` option. With quick scan mode enabled, Fortify Static Code Analyzer applies the properties from the *<sca_install_dir>*`/Core/config/fortify-sca-quickscan.properties` file, in addition to the standard *<sca_install_dir>*`/Core/config/fortify-sca.properties` file. You can adjust the limiters that Fortify Static Code Analyzer uses by editing the `fortify-sca-quickscan.properties` file. If you modify `fortify-sca.properties`, it also affects quick scan behavior. OpenText recommends that you do performance tuning in quick scan mode, and leave the full scan in the default settings to produce a highly accurate scan. For description of the quick scan mode properties, see "Properties files" on page 184.

## Using quick scan and full scan

- **Run full scans periodically**—A periodic full scan is important as it might find issues that quick scan mode does not detect. Run a full scan at least once per software iteration. If possible, run a full scan periodically when it will not interrupt the development workflow, such as on a weekend.
- **Compare quick scan with a full scan**—To evaluate the accuracy impact of a quick scan, perform a quick scan and a full scan on the same codebase. Open the quick scan results in Fortify Audit Workbench and merge it into the full scan. Group the issues by **New Issue** to produce a list of issues detected in the full scan but not in the quick scan.
- **Quick scans and Fortify Software Security Center**—To avoid overwriting the results of a full scan, by default Fortify Software Security Center ignores uploaded FPR files scanned in quick scan

mode. However, you can configure a Fortify Software Security Center application version so that FPR files scanned in quick scan are processed. For more information, see analysis results processing rules in the *OpenText™ Fortify Software Security Center User Guide*.

# Configuring scan speed with speed dial

You can configure the speed and depth of the scan by specifying a precision level for the analysis phase. You can use these precision levels to adjust the scan time to fit for example, into a pipeline and quickly find a set of vulnerabilities while the developer is still working on the code. Although scans with the speed dial settings are faster than a full scan, it does not provide as robust a result set. OpenText recommends that you run full scans whenever possible.

The precision level controls the depth and precision of the scan by associating configuration properties with each level. The configuration properties files for each level are in the `<sca_install_dir>/Core/config/scales` directory. There is one file for each level: (`level-<precision_level>.properties`). You can modify the settings in these files to create your own specific precision levels.

**Notes:**

- By default, Fortify Software Security Center blocks uploaded analysis results that were created with a precision level less than four. However, you can configure your Fortify Software Security Center application version so that uploaded audit projects scanned with these precision levels are processed.
- If you merge a speed dial scan with a full scan, this might remove issues from previous scans that still exist in your application (and would be detected again with a full scan).

To specify the speed dial setting for a scan, include the `-scan-precision` (or `-p`) option in the scan phase as shown in the following example:

```
sourceanalyzer -b MyProject -scan -scan-precision <level> -f MyResults.fpr
```

**Note:** You cannot use the speed dial setting and the `-quick` option in the same scan command.

The following table describes the four precision levels.

| Precision level | Description |
|---|---|
| 1 | This is the quickest scan and is recommended to scan a few files. By default, a scan with this precision level disables the Buffer Analyzer, Control Flow Analyzer, Dataflow Analyzer, and Null Pointer Analyzer. |
| 2 | By default, a scan with this precision level enables all analyzers. The scan runs quicker by performing with reduced limiters. This results in fewer issues detected. |

| Precision level | Description |
|---|---|
| 3 | This precision level improves intermediate development scan speeds by up to 50% (with a reduction in reported issues). Specifically, this level improves the scan time for typed languages such as Java and C/C++. |
| 4 | This is equivalent to a full scan. |

You can also specify the scan precision level with the `com.fortify.sca.PrecisionLevel` property in the `fortify-sca.properties` file. For example:

```
com.fortify.sca.PrecisionLevel=1
```

# Breaking down codebases

It is more efficient to break down large projects into independent modules. For example, if you have a portal application that consists of several modules that are independent of each other or have few interactions, you can translate and scan the modules separately. The caveat to this is that you might lose dataflow issue detection if some interactions exist.

For C/C++, you might reduce the scan time by using the `–bin` option with the `–scan` option. You need to pass the binary file as the parameter (such as `-bin <filename>.exe -scan` or `-bin <filename>.dll -scan`). Fortify Static Code Analyzer finds the related files associated with the binary and scans them. This is useful if you have several binaries in a makefile.

The following table lists some useful Fortify Static Code Analyzer command-line options to break down codebases.

| Option | Description |
|---|---|
| `-bin <binary>` | Specifies a subset of source files to scan. Only the source files that were linked in the named binary at build time are included in the scan. You can use this option multiple times to specify the inclusion of multiple binaries in the scan. |
| `-show-binaries` | Displays all objects that were created but not used in the production of any other binaries. If fully integrated into the build, it lists all the binaries produced. |
| `-show-build-tree` | When used with the `-bin` option, displays all files used to create the binary and all files used to create those files in a tree layout. If the `-bin` option is not present, Fortify Static Code Analyzer displays the tree for each binary. |

# Limiting analyzers and languages

Occasionally, you might find that a significant amount of the scan time is spent either running one analyzer or analyzing a particular language. It is possible that this analyzer or language is not important to your security requirements. You can limit the specific analyzers that run and the specific languages that Fortify Static Code Analyzer translates.

## Disabling analyzers

To disable specific analyzers, include the `-analyzers` option to Fortify Static Code Analyzer at scan time with a comma- or colon-separated list of analyzers to enable. The valid parameter values for the `-analyzers` option are `buffer`, `content`, `configuration`, `controlflow`, `dataflow`, `nullptr`, `semantic`, and `structural`.

For example, to run a scan that only includes the Dataflow, Control Flow, and Buffer analyzers, use the following scan command:

```
sourceanalyzer -b MyProject -analyzers dataflow:controlflow:buffer -scan -f
MyResults.fpr
```

You can also do the same thing by setting `com.fortify.sca.DefaultAnalyzers` in the Fortify Static Code Analyzer property file *<sca_install_dir>*`/Core/config/fortify-sca.properties`. For example, to achieve the equivalent of the previous scan command, set the following in the properties file:

```
com.fortify.sca.DefaultAnalyzers=dataflow:controlflow:buffer
```

## Disabling languages

To disable specific languages, include the `-disable-language` option in the translation phase, which specifies a list of languages that you want to exclude. The valid language values are `abap`, `actionscript`, `apex`, `cfml`, `cobol`, `configuration`, `cpp`, `dart`, `dotnet`, `golang`, `java`, `javascript`, `jsp`, `kotlin`, `objc`, `php`, `plsql`, `python`, `ruby`, `scala`, `sql`, `swift`, `tsql`, `typescript`, and `vb`.

For example, to perform a translation that excludes SQL and PHP files, use the following command:

```
sourceanalyzer -b MyProject <src_files> -disable-language sql:php
```

You can also disable languages by setting the `com.fortify.sca.DISabledLanguages` property in the Fortify Static Code Analyzer properties file *<sca_install_dir>*`/Core/config/fortify-sca.properties`. For example, to achieve the equivalent of the previous translation command, set the following in the properties file:

```
com.fortify.sca.DISabledLanguages=sql:php
```

# Optimizing FPR files

This chapter describes how to handle performance issues related to the audit results (FPR) file. These topics describe how to reduce the scan time, reduce FPR file size, and tips for opening large FPR files.

## Using filter files

You can use a file to filter out specific vulnerability instances, rules, and vulnerability categories from the analysis results. If you determine that a certain issue category or rule is not relevant for a particular scan, you can stop Fortify Static Code Analyzer from adding them to the FPR. Using a filter file can reduce both the scan time and analysis results file size.

For example, if you scan a simple program that just reads a specified file, you might not want to see path manipulation issues, because these are not likely planned as part of the functionality. To filter out path manipulation issues, create a file that contains a single line:

```
Path Manipulation
```

Save this file as `filter.txt`. Use the `-filter` option in the analysis phase as shown in the following example:

```
sourceanalyzer -b MyProject -scan -filter filter.txt -f MyResults.fpr
```

The analysis output in `MyResults.fpr` does not include any issues with the category Path Manipulation. For more information and an example of a filter file, see "Excluding issues with filter files" on page 178.

## Using filter sets

Filters in an issue template determine how the results from Fortify Static Code Analyzer are shown. In addition to filters, filter sets enable you to have a selection of filters used at any one time. Each FPR has an issue template associated with it. You can use filter sets to reduce the number of issues based on conditions you specify with filters in an issue template. This can dramatically reduce the size of an FPR.

To do this, use Fortify Audit Workbench to create a filter in a filter set, and then run the Fortify Static Code Analyzer scan with the filter set and the containing issue template. For more information and a basic example of how to create a filter set, see "Using filter sets to exclude issues" on page 182.

**Note:** Although filtering issues with a filter set can reduce the size of the FPR, they do not usually reduce the scan time. Fortify Static Code Analyzer examines the filter set after it calculates the issues to determine whether to write them to the FPR file. The filters in a filter set determine the rule types that Fortify Static Code Analyzer loads.

## Excluding source code from the FPR

You can reduce the size of the FPR file by excluding the source code information from the FPR. This is especially valuable for large source files or codebases. Typically, you do not get a scan time reduction for small source files using this method.

There are properties you can use to prevent Fortify Static Code Analyzer from including source code in the FPR. You can set either property in the `<sca_install_dir>/Core/config/fortify-sca.properties` file or specify an option on the command line. The following table describes these settings.

| Property name | Description |
|---|---|
| `com.fortify.sca.FPRDisableSourceBundling=true`<br>**Command-Line Option:**<br>`-disable-source-bundling` | Excludes source code from the FPR. |
| `com.fortify.sca.FVDLDisableSnippets=true`<br>**Command-Line Option:**<br>`–fvdl-no-snippets` | Excludes code snippets from the FPR. |

The following command-line example uses both options to exclude both the source code and code snippets from the FPR:

```
sourceanalyzer -b MyProject -disable-source-bundling
-fvdl-no-snippets -scan -f MySourcelessResults.fpr
```

## Reducing the FPR file size

There are a few ways to reduce the size of FPR files. The quickest way to do this without affecting results is to exclude the source code from the FPR as described in "Excluding source code from the FPR" above. You can also reduce the size of a merged FPR with the FPRUtility (see the *OpenText™ Fortify Static Code Analyzer Applications and Tools Guide*).

There are a few other properties that you can use to select what is excluded from the FPR. You can set these properties in the `<sca_install_dir>/Core/config/fortify-sca.properties` file or specify an option on the command line for the analysis (scan) phase.

| Property name | Description |
|---|---|
| `com.fortify.sca.FPRDisableMetatable` | Excludes the metatable from the FPR. Fortify Audit Workbench uses the metatable to map information |

| Property name | Description |
|---|---|
| `=true`<br>**Command-Line Option:**<br>`-disable-metatable` | in Functions view. |
| `com.fortify.sca.FVDLDisableDescriptions=true`<br>**Command-Line Option:**<br>`-fvdl-no-descriptions` | Excludes rule descriptions from the FPR. If you do not use custom descriptions, the descriptions in the Fortify Taxonomy (https://vulncat.fortify.com) are used. |
| `com.fortify.sca.FVDLDisableEngineData=true`<br>**Command-Line Option:**<br>`-fvdl-no-enginedata` | Excludes engine data from the FPR. This is useful if your FPR contains many warnings when you open the file in Fortify Audit Workbench.<br>**Note:** If you exclude engine data from the FPR, you must merge the FPR with the current audit project locally before you upload it to Fortify Software Security Center. Fortify Software Security Center cannot merge it on the server because the FPR does not contain the Fortify Static Code Analyzer version. |
| `com.fortify.sca.FVDLDisableProgramData=true`<br>**Command-Line Option:**<br>`-fvdl-no-progdata` | Excludes the program data from the FPR. This removes the Taint Sources information from the Functions view in Fortify Audit Workbench. This property typically only has a minimal effect on the overall size of the FPR file. |

## Opening large FPR files

To reduce the time required to open a large FPR file in Fortify Audit Workbench, you can set some properties in the *<sca_install_dir>*`/Core/config/fortify.properties` file. For more information about these properties, see the *OpenText™ Fortify Static Code Analyzer Applications and Tools Guide*. The following table describes the properties you can use to reduce the time to open large FPR files.

| Property name | Description |
|---|---|
| `com.fortify.model.DisableProgramInfo=true` | Disables use of the code navigation features in Fortify Audit Workbench. |

| Property name | Description |
|---|---|
| com.fortify.<br>model.IssueCutOffStartIndex<br>=*<num>* (inclusive)<br><br>com.fortify.<br>model.IssueCutOffEndIndex<br>=*<num>* (exclusive) | Sets the start and end index for issue cutoff.<br>The IssueCutOffStartIndex property is inclusive and IssueCutOffEndIndex is exclusive so that you can specify a subset of issues you want to see. For example, to see the first 100 issues, specify the following:<br><br>com.fortify.model.<br>IssueCutOffStartIndex=0<br><br>com.fortify.model.<br>IssueCutOffEndIndex=101<br><br>Because the IssueCutOffStartIndex is 0 by default, you do not need to specify this property. |
| com.fortify.<br>model.IssueCutOffByCategoryStartIndex=<br>*<num>* (inclusive)<br><br>com.fortify.<br>model.IssueCutOffByCategoryEndIndex=<br>*<num>* (exclusive) | Sets the start index for issue cutoff by category. These two properties are similar to the previous cutoff properties except these are specified for each category. For example, to see the first five issues for every category, specify the following:<br><br>com.fortify.model.<br>IssueCutOffByCategoryEndIndex=6 |
| com.fortify.<br>model.MinimalLoad=true | Minimizes the data loaded from the FPR. This also restricts usage of the Functions view and might prevent Fortify Audit Workbench from loading the source from the FPR. |
| com.fortify.<br>model.MaxEngineErrorCount=<br>*<num>* | Specifies the number of Fortify Static Code Analyzer reported warnings to load from the FPR. For projects with many scan warnings, reducing this number from a default of 3000 can speed up the load time of large FPR files. |
| com.fortify.<br>model.ExecMemorySetting | Specifies the JVM heap memory size for Fortify Audit Workbench to start external command-line tools such as iidmigrator and fortifyupdate. |

# Monitoring long running scans

When you run Fortify Static Code Analyzer, large and complex scans can often take a long time to complete. During the scan it is not always clear what is happening. While OpenText recommends that you provide your debug logs to the Customer Support team, there are a couple of ways to see what Fortify Static Code Analyzer is doing and how it is performing in real-time.

## Using the SCAState tool

The SCAState command-line tool enables you to see up-to-date state analysis information during the analysis phase. The SCAState tool is located in the *<sca_install_dir>*/bin directory. In addition to a live view of the analysis, it also provides a set of timers and counters that show where Fortify Static Code Analyzer spends its time during the analysis phase. For more information about how to use SCAState, see the *"Checking the scan status with SCAState" on page 152*.

## Using JMX tools

You can use tools to monitor Fortify Static Code Analyzer with JMX technology. These tools can provide a way to track Fortify Static Code Analyzer performance over time. For more information about these tools, see the full Oracle documentation available at: http://docs.oracle.com.

**Note:** These are third-party tools and OpenText does not provide or support them.

### Using JConsole

JConsole is an interactive monitoring tool that complies with the JMX specification. The disadvantage of JConsole is that you cannot save the output.

To use JConsole, you must first set some additional JVM parameters. Set the following environment variable:

```
export SCA_VM_OPTS="-Dcom.sun.management.jmxremote
-Dcom.sun.management.jmxremote.port=9090
-Dcom.sun.management.jmxremote.ssl=false
-Dcom.sun.management.jmxremote.authenticate=false"
```

After the JMX parameters are set, start a Fortify Static Code Analyzer scan. During the scan, start JConsole to monitor Fortify Static Code Analyzer locally or remotely with the following command:

```
jconsole <host_name>:9090
```

### Using Java VisualVM

Java VisualVM offers the same capabilities as JConsole. It also provides more detailed information on the JVM and enables you to save the monitor information to an application snapshot file. You can store these files and open them later with Java VisualVM.

Similar to JConsole, before you can use Java VisualVM, you must set the same JVM parameters described in "Using JConsole" on the previous page.

After the JVM parameters are set, start the scan. You can then start Java VisualVM to monitor the scan either locally or remotely with the following command:

```
jvisualvm <host_name>:9090
```

# Chapter 21: Troubleshooting

This section contains the following topics:



## Exit codes

The following table describes the possible Fortify Static Code Analyzer exit codes.

| Exit code | Description |
|---|---|
| 0 | Success |
| 1 | Generic failure |
| 2 | Invalid input files<br>(this might indicate that an attempt was made to translate a file that has an extension that Fortify Static Code Analyzer does not support) |
| 3 | Process timed out |
| 4 | Analysis completed with numbered warning messages written to the console and/or to the log file |
| 5 | Analysis completed with numbered error messages written to the console and/or to the log file |
| 6 | Scan phase was unable to generate issue results |
| 7 | Unable to detect a valid license or the LIM license expired at run time |

By default, Fortify Static Code Analyzer only returns exit codes 0, 1, 2, 3, or 7.

You can extend the default exit code options by setting the `com.fortify.sca.ExitCodeLevel` property in the *<sca_install_dir>*`/Core/Config/fortify-sca.properties` file.

The valid values are:

- `nothing`—Returns any of the default exit codes (0, 1, 2, 3, or 7).
- `warnings`—Returns exit codes 4 and 5 in addition to the default exit codes.
- `errors`—Returns exit code 5 in addition to the default exit codes.
- `no_output_file`—Returns exit code 6 in addition to the default exit codes.

# Memory tuning

The amount of physical RAM required for a scan depends on the complexity of the code. By default, Fortify Static Code Analyzer automatically allocates the memory it uses based on the physical memory available on the system. This is generally sufficient. As described in "Output options" on page 139, you can adjust the Java heap size with the -Xmx command-line option.

This section describes suggestions for what you can do if you encounter OutOfMemory errors during the analysis.

**Note:** You can set the memory allocation options discussed in this section to run for all scans by setting the `SCA_VM_OPTS` environment variable.

## Java heap exhaustion

Java heap exhaustion is the most common memory problem that might occur during Fortify Static Code Analyzer scans. It is caused by allocating too little heap space to the Java virtual machine that Fortify Static Code Analyzer uses to scan the code. You can identify Java heap exhaustion from the following symptom.

**Symptom**

One or more of these messages appears in the Fortify Static Code Analyzer log file and in the command-line output:

```
There is not enough memory available to complete analysis. For details on
making more memory available, please consult the user manual.
java.lang.OutOfMemoryError: Java heap space
java.lang.OutOfMemoryError: GC overhead limit exceeded
```

**Resolution**

To resolve a Java heap exhaustion problem, allocate more heap space to the Fortify Static Code Analyzer Java virtual machine when you start the scan. To increase the heap size, use the `-Xmx` command-line option when you run the Fortify Static Code Analyzer scan. For example, `-Xmx1G` makes 1 GB available. Before you use this parameter, determine the maximum allowable value for Java heap space. The maximum value depends on the available physical memory.

Heap sizes between 32 GB and 48 GB are not advised due to internal JVM implementations. Heap sizes in this range perform worse than at 32 GB. Heap sizes smaller than 32 GB are optimized by the JVM. If your scan requires more than 32 GB, then you need 64 GB or more. As a guideline, assuming no other memory intensive processes are running, do not allocate more than 2/3 of the available memory.

If the system is dedicated to running Fortify Static Code Analyzer, you do not need to change it. However, if the system resources are shared with other memory-intensive processes, subtract an allowance for those other processes.

**Note:** You do not need to account for other resident but not active processes (while Fortify Static Code Analyzer is running) that the operating system might swap to disk. Allocating more physical memory to Fortify Static Code Analyzer than is available in the environment might cause "thrashing," which typically slows down the scan along with everything else on the system.

## Native heap exhaustion

Native heap exhaustion is a rare scenario where the Java virtual machine can allocate the Java memory regions on startup, but is left with so few resources for its native operations (such as garbage collection) that it eventually encounters a fatal memory allocation failure that immediately terminates the process.

**Symptom**

You can identify native heap exhaustion by abnormal termination of the Fortify Static Code Analyzer process and the following output on the command line:

```
# A fatal error has been detected by the Java Runtime Environment:
#
# java.lang.OutOfMemoryError: requested ... bytes for GrET ...
```

Because this is a fatal Java virtual machine error, it is usually accompanied by an error log created in the working directory with the file name `hs_err_pidNNN.log`.

**Resolution**

Because the problem is a result of overcrowding within the process, the resolution is to reduce the amount of memory used for the Java memory regions (Java heap). Reducing this value should reduce the crowding problem and allow the scan to complete successfully.

## Stack overflow

Each thread in a Java application has its own stack. The stack holds return addresses, function/method call arguments, and so on. If a thread tends to process large structures with recursive algorithms, it might need a large stack for all those return addresses. With the JVM, you can set that size with the `-Xss` option.

**Symptoms**

This message typically appears in the Fortify Static Code Analyzer log file, but might also appear in the command-line output:

```
java.lang.StackOverflowError
```

**Resolution**

The default stack size is 16 MB. To increase the stack size, pass the `-Xss` option to the `sourceanalyzer` command. For example, `-Xss32M` increases the stack to 32 MB.

# Scanning complex functions

During a Fortify Static Code Analyzer scan, the Dataflow Analyzer might encounter a function for which it cannot complete the analysis and reports the following message:

```
Function <name> is too complex for <analyzer> analysis and will be skipped
(<identifier>)
```

where:

- *<name>* is the name of the source code function
- *<analyzer>* is the name of the analyzer
- *<identifier>* is the type of complexity, which is one of the following:
  - `l`: Too many distinct locations
  - `m`: Out of memory
  - `s`: Stack size too small
  - `t`: Analysis taking too much time
  - `v`: Function visits exceed the limit

The depth of analysis Fortify Static Code Analyzer performs sometimes depends on the available resources. Fortify Static Code Analyzer uses a complexity metric to trade off these resources against the number of vulnerabilities that it can find. Sometimes, this means giving up on a particular function when Fortify Static Code Analyzer does not have enough resources available. This is normally when you see the "Function too complex" messages.

When you see this message, it does not necessarily mean that Fortify Static Code Analyzer completely ignored the function in the program. For example, the Dataflow Analyzer typically visits a function many times before completing the analysis, and might not have run into this complexity limit in the previous visits. In this case, the results include everything learned from the previous visits.

You can control the "give up" point using Fortify Static Code Analyzer properties called limiters. Different analyzers have different limiters.

The following sections provide a discussion of a resolution for this issue.

## Dataflow Analyzer limiters

There are three types of complexity identifiers for the Dataflow Analyzer:

- `l`: Too many distinct locations
- `m`: Out of memory
- `s`: Stack size too small
- `v`: Function visits exceed the limit

To resolve the issue identified by `s`, increase the stack size for by setting `-Xss` to a value greater than 16 MB.

To resolve the complexity identifier of `m`, increase the physical memory for Fortify Static Code Analyzer.

To resolve the complexity identifier of `l`, you can adjust the following limiters in the Fortify Static Code Analyzer property file *<sca_install_dir>*`/Core/config/fortify-sca.properties` or on the command line.

| Property name | Default value |
|---|---|
| `com.fortify.sca.limiters.MaxTaintDefForVar` | 1000 |
| `com.fortify.sca.limiters.MaxTaintDefForVarAbort` | 4000 |
| `com.fortify.sca.limiters.MaxFieldDepth` | 4 |

The `MaxTaintDefForVar` limiter is a dimensionless value expressing the complexity of a function, while `MaxTaintDefForVarAbort` is the upper bound for it. Use the `MaxFieldDepth` limiter to measure the precision when the Dataflow Analyzer analyzes any given object. Fortify Static Code Analyzer always tries to analyze objects at the highest precision possible.

If a given function exceeds the `MaxTaintDefForVar` limit at a given precision, the Dataflow Analyzer analyzes that function with lower precision (by reducing the `MaxFieldDepth` limiter). When you reduce the precision, it reduces the complexity of the analysis. When the precision cannot be reduced any further, Fortify Static Code Analyzer then proceeds with analysis at the lowest precision until either it finishes, or the complexity exceeds the `MaxTaintDefForVarAbort` limiter. In other words, Fortify Static Code Analyzer tries harder at the lowest precision to get at least some results from the function. If Fortify Static Code Analyzer reaches the `MaxTaintDefForVarAbort` limiter, it gives up on the function entirely and you get the "Function too complex" warning.

To resolve the complexity identifier of `v`, you can adjust the property `com.fortify.sca.limiters.MaxFunctionVisits`. This property sets the maximum number of times the taint propagation analyzer visits functions. The default is `50`.

## Control Flow and Null Pointer analyzer limiters

There are two types of complexity identifiers for both Control Flow and Null Pointer analyzers:

- `m`: Out of memory
- `t`: Analysis taking too much time

Due to the way that the Dataflow Analyzer handles function complexity, it does not take an indefinite amount of time. Control Flow and Null Pointer analyzers, however, can take an exceptionally long time when analyzing complex functions. Therefore, Fortify Static Code Analyzer provides a way to abort the analysis when this happens, and then you get the "Function too complex" message with a complexity identifier of `t`.

To change the maximum amount of time these analyzers spend to analyze functions, you can adjust the following property values in the Fortify Static Code Analyzer property file *<sca_install_dir>*`/Core/config/fortify-sca.properties` or on the command line.

| Property name | Description | Default value |
|---|---|---|
| `com.fortify.sca.CtrlflowMaxFunctionTime` | Sets the time limit (in milliseconds) for Control Flow analysis on a single function. | 600000 (10 minutes) |
| `com.fortify.sca.NullPtrMaxFunctionTime` | Sets the time limit (in milliseconds) for Null Pointer analysis on a single function. | 300000 (5 minutes) |

To resolve the complexity identifier of `m`, increase the physical memory for Fortify Static Code Analyzer.

**Note:** If you increase these limiters or time settings, it makes the analysis of complex functions take longer. It is difficult to characterize the exact performance implications of a particular value for the limiters/time, because it depends on the specific function in question. If you never want to see the "Function too complex" warning, you can set the limiters/time to an extremely high value, however it can cause unacceptable scan time.

# Issue non-determinism

Running in parallel analysis mode might introduce issue non-determinism. If you experience any problems, contact Customer Support, and disable parallel analysis mode. Disabling parallel analysis mode results in sequential analysis, which can be substantially slower but provides deterministic results across multiple scans.

To disable parallel analysis mode:

1. Open the `fortify-sca.properties` file located in the *<sca_install_dir>*`/Core/config` directory in a text editor.

2. Change the value for the `com.fortify.sca.MultithreadedAnalysis` property to `false`.

```
com.fortify.sca.MultithreadedAnalysis=false
```

# Locating the log files

By default, Fortify Static Code Analyzer creates log files in the following location:

- Windows: `C:\Users\<username>\AppData\Local\Fortify\sca<version>\log`
- Non-Windows: `<userhome>/.fortify/sca<version>/log`

where `<version>` is the version of Fortify Static Code Analyzer that you are using.

The following table describes the Fortify Static Code Analyzer default log files.

| File names | Description |
|---|---|
| `sca.log`<br>`scaX.log` | The standard log provides a log of informational messages, warnings, and errors that occurred in the run of sourceanalyzer. |
| `sca_FortifySupport.log`<br>`scaX_FortifySupport.log` | The Static Code Analyzer Support log provides:<br>• The same log messages as the standard log file, but with additional details<br>• Additional detailed messages that are not included in the standard log file<br>This log file is helpful to Customer Support or the development team to troubleshoot any issues. |

If you encounter warnings or errors that you cannot resolve, provide the Static Code Analyzer Support log file to Customer Support.

# Configuring log files

You can configure the information that Fortify Static Code Analyzer writes to the log files by setting logging properties (see "Logging properties" on page 212). You can configure the following log file settings:

- The location and name of the log file

  Property: `com.fortify.sca.LogFile`

- Log level (see "Understanding log levels" below)

  Property: `com.fortify.sca.LogLevel`
- Whether to overwrite the log files for each run of sourceanalyzer

  Property: `com.fortify.sca.ClobberLogFile`

  Command-line option: `-clobber-log`

## Understanding log levels

The log level you select gives you all log messages equal to and greater than it. The following table lists the log levels in order from least to greatest. For example, the default log level of INFO includes log messages with the following levels: INFO, WARN, ERROR, and FATAL. You can set the log level with the `com.fortify.sca.LogLevel` property in the *<sca_install_dir>*`/Core/config/fortify-sca.properties` file or on the command-line using the `-D` option.

| Log level | Description |
|---|---|
| DEBUG | Includes information that Customer Support or the development team can use to troubleshoot an issue |
| INFO | Basic information about the translation or scan process |
| WARN | Information about issues where the translation or scan did not stop, but might require your attention for accurate results |
| ERROR | Information about an issue that might require attention |
| FATAL | Information about an error that caused the translation or scan to abort |

# Reporting issues and requesting enhancements

Feedback is critical to the success of this product. To request enhancements or patches, or to report issues, visit Customer Support at https://www.microfocus.com/support.

Include the following information when you contact customer support:

- Product: Fortify Static Code Analyzer
- Version number of Fortify Static Code Analyzer and any independent Fortify Static Code Analyzer modules: To determine the version numbers, run the following:

  ```
  sourceanalyzer -version
  ```

- Platform: (for example, Red Hat Enterprise Linux <*version*>)
- Operating system: (such as Linux)

To request an enhancement, include a description of the feature enhancement.

To report an issue, provide enough detail so that support can duplicate the issue. The more descriptive you are, the faster support can analyze and resolve the issue. Also include the log files, or the relevant portions of them, from when the issue occurred.

# Appendix A: Filtering the analysis

This section describes two methods of filtering out vulnerabilities from the analysis results (FPR) during the scan phase. You can use a filter file to remove issues based on specific vulnerability instances, rules, and vulnerability categories. You can also use a filter set (created in Fortify Audit Workbench) to remove issues that are hidden from view in an issue template.

**Caution!** OpenText recommends that you only use filter files if you are an advanced user. Do not use filter files for standard audits, because auditors typically want to see and evaluate all issues that Fortify Static Code Analyzer finds.

This section contains the following topics:



## Excluding issues with filter files

You can create a file to filter out particular vulnerability instances, rules, and vulnerability categories when you run the `sourceanalyzer` command. You specify the file with the `-filter` analysis option.

A filter file is a text file that you can create with any text editor. You specify only the filter items that you *do not* want in this file.

**Note:** The filter types described in this section apply to both filter files and scan policy files (see "Applying a scan policy to the analysis" on page 46).

The following table lists the available filter types and provides examples for each.

| Filter type | Notes | Examples |
|---|---|---|
| Category | A category only covers all subcategories<br><br>**Note:** Fortify Static Code Analyzer applies category filters in the initialization phase before any analysis has taken place. | `Poor Error Handling`<br><br>`J2EE Bad Practices: Leftover Debug Code` |
| Instance ID | An instance ID of a specific issue<br><br>**Note:** Fortify Static Code | `6291C6A33303ED270C269917AA8A1005` |

| Filter type | Notes | Examples |
|---|---|---|
| | Analyzer applies instance ID filters after the analysis phase. | |
| Rule ID | A rule ID that leads to the reporting of a specific issue<br><br>**Note:** Fortify Static Code Analyzer applies rule ID filters in the initialization phase before any analysis has taken place. | `823FE039-A7FE-4AAD-B976-9EC53FFE4A59` |
| Priority[1] | The priority values in ascending order are `low`, `medium`, `high`, and `critical`. | `priority <= low`<br>`priority < medium` |
| Taint flags | Enclose taint flag expressions in parentheses. Use the logical `&&`, `\|\|`, and `!` operators to specify an expression. For a list of taint flags, see *OpenText™ Fortify Static Code Analyzer Custom Rules Guide*. | `(SYSTEMINFO \|\| EXCEPTIONINFO)`<br>`(WEB \|\| (DATABASE && PRIVATE))`<br>`(NETWORK && !XSS)` |
| Impact[1] | | `impact < 0.5` |
| Likelihood[1] | | `likelihood <= 1.5` |
| Confidence[1] | | `confidence < 1.8` |
| Probability[1] | | `probability <= 1.2` |
| Accuracy[1] | | `accuracy <= 1.0` |

[1]For the priority and metadata filters, use less than (<) or less than or equal to (<=).

**See also**

"Filter file example" on the next page

## Filter file example

As an example, the following output is from a scan of the `EightBall.java` sample. This sample project is included in the `Fortify_SCA_Samples_<version>.zip` archive in the `basic/eightball` directory.

The following commands are executed to produce the analysis results:

```
sourceanalyzer -b eightball EightBall.java
sourceanalyzer -b eightball -scan
```

The following results show five detected issues:

```
[F7A138CDE5235351F6A4405BA4AD7C53 : low : Unchecked Return Value : semantic
]
EightBall.java(12) : Reader.read()

[6291C6A33303ED270C269917AA8A1005 : high : Path Manipulation : dataflow ]
EightBall.java(12) :  ->new FileReader(0)
    EightBall.java(8) : <=> (filename)
    EightBall.java(8) : <->Integer.parseInt(0->return)
    EightBall.java(6) : <=> (filename)
    EightBall.java(4) :  ->EightBall.main(0)

[176CC0B182267DD538992E87EF41815F : critical : Path Manipulation : dataflow
]
EightBall.java(12) :  ->new FileReader(0)
    EightBall.java(6) : <=> (filename)
    EightBall.java(4) :  ->EightBall.main(0)

[E4B3ACF92911ED6D98AAC15876739EC7 : high : Unreleased Resource : Streams :
controlflow ]

    EightBall.java(12) : start -> loaded : new FileReader(...)
    EightBall.java(14) : loaded -> end_of_scope : end scope : Resource
leaked

    EightBall.java(12) : start -> loaded : new FileReader(...)
    EightBall.java(12) : java.io.IOException thrown
    EightBall.java(12) : loaded -> loaded : throw
    EightBall.java(12) : loaded -> end_of_scope : end scope : Resource
leaked : java.io.IOException thrown

[BB9F74FFA0FF75C9921D0093A0665BEB : low : J2EE Bad Practices : Leftover
Debug Code : structural ]
    EightBall.java(4)
```

The following is an example filter file that performs the following:

- Remove all results related to the J2EE Bad Practice category
- Remove the Path Manipulation based on its instance ID
- Remove any dataflow issues that were generated from a specific rule ID

```
#This is a category to filter from scan output
J2EE Bad Practices
```

```
#This is an instance ID of a specific issue to be filtered
#from scan output
6291C6A33303ED270C269917AA8A1005

#This is a specific Rule ID that leads to the reporting of a
#specific issue in the scan output: in this case the
#dataflow sink for a Path Manipulation issue.
823FE039-A7FE-4AAD-B976-9EC53FFE4A59
```

To test the filtered output, copy the above text and paste it into a file with the name `test_filter.txt`.

To apply the filtering in the `test_filter.txt` file, execute the following command:

```
sourceanalyzer -b eightball -scan -filter test_filter.txt
```

The filtered analysis produces the following results:

```
[176CC0B182267DD538992E87EF41815F : critical : Path Manipulation : dataflow
]
EightBall.java(12) :  ->new FileReader(0)
    EightBall.java(6) : <=> (filename)
    EightBall.java(4) :  ->EightBall.main(0)

[E4B3ACF92911ED6D98AAC15876739EC7 : high : Unreleased Resource : Streams :
controlflow ]

    EightBall.java(12) : start -> loaded : new FileReader(...)
    EightBall.java(14) : loaded -> end_of_scope : end scope : Resource
leaked

    EightBall.java(12) : start -> loaded : new FileReader(...)
    EightBall.java(12) : java.io.IOException thrown
    EightBall.java(12) : loaded -> loaded : throw
    EightBall.java(12) : loaded -> end_of_scope : end scope : Resource
leaked : java.io.IOException thrown
```

# Using filter sets to exclude issues

You can use filter sets in an issue template created in Fortify Audit Workbench to filter issues from the analysis results. When you apply a filter set that hides issues from view during the analysis phase, Fortify Static Code Analyzer does not write the hidden issues to the FPR. To do this, use Fortify Audit Workbench to create a filter set, and then run the Fortify Static Code Analyzer scan with the filter set and the issue template, which contains the filter set. For more detailed instructions about how to

create filters and filter sets in Fortify Audit Workbench, see the *OpenText™ Fortify Audit Workbench User Guide*.

The following example describes the basic steps for how to create and use a filter in an issue template to remove issues from an FPR:

1. Suppose you use OWASP Top 10 2021, and you only want to see issues categorized within this standard. In Fortify Audit Workbench, create a new filter set called `OWASP_Filter`
2. In Fortify Audit Workbench, create a visibility filter in the `OWASP_Filter` filter set:

   ```
   If [OWASP Top 10 2021] does not contain A Then hide issue
   ```

   This filter looks through the issues and if an issue does not map to an OWASP Top 10 2021 category with 'A' in the name, then it hides it. Because all OWASP Top 10 2021 categories start with 'A' (A01, A02, ..., A10), then any category without the letter 'A' is not in the OWASP Top 10 2021. The filter hides the issues from view in Fortify Audit Workbench, but they are still in the FPR.
3. In Fortify Audit Workbench, export the issue template to a file called `IssueTemplate.xml`.
4. Using Fortify Static Code Analyzer, specify the filter set in the analysis phase with the following command:

   ```
   sourceanalyzer -b MyProject -scan -project-template IssueTemplate.xml
   -Dcom.fortify.sca.FilterSet=OWASP_Filter -f MyFilteredResults.fpr
   ```

Although filtering issues with a filter set can reduce the size of the FPR, it does not usually reduce the scan time. Fortify Static Code Analyzer examines the filter set after it calculates the issues to determine whether to write them to the FPR file. The filters in a filter set determine the rule types that Fortify Static Code Analyzer loads.

# Appendix B: Configuration options

The Fortify Static Code Analyzer installer places a set of properties files on your system. Properties files contain configurable settings for Fortify Static Code Analyzer runtime analysis, output, and performance.

This section contains the following topics:



## Properties files

The properties files are located in the `<sca_install_dir>/Core/config` directory. The installed properties files contain default values. OpenText recommends that you consult with your project leads before you make changes to the properties in the properties files. You can modify any of the properties in the configuration file with any text editor. You can also specify the property on the command line with the `-D` option.

The following table lists the Fortify Static Code Analyzer properties files. Property files for the Fortify Static Code Analyzer applications and tools are described in the *OpenText™ Fortify Static Code Analyzer Applications and Tools Guide*.

| Properties file name | Description | More information |
|---|---|---|
| `fortify-sca.properties` | Defines the Fortify Static Code Analyzer configuration properties. | "fortify-sca.properties" on page 186 |
| `fortify-sca-quickscan.properties` | Defines the configuration properties applicable for a Fortify Static Code Analyzer quick scan. | "fortify-sca-quickscan.properties" on page 213 |
| `fortify-rules.properties` | Defines the configuration properties that determine rule behavior. | "fortify-rules.properties" on page 216 |

## Properties file format

In the properties file, each property consists of a pair of strings: the first string is the property name and the second string is the property value.

```
com.fortify.sca.fileextensions.htm=HTML
```

As shown above, the property sets the translation to use for `.htm` files. The property name is `com.fortify.sca.fileextensions.htm` and the value is set to `HTML`.

**Note:** When you specify a path for Windows systems as the property value, you must escape any backslash character (\) with a backslash (for example: `com.fortify.sca.ASPVirtualRoots.Library=C:\\WebServer\\CustomerA\\inc`).

Disabled properties are commented out of the properties file. To enable these properties, remove the comment symbol (#) and save the properties file. In the following example, the `com.fortify.sca.LogFile` property is disabled in the properties file and is not part of the configuration:

```
# default location for the log file
#com.fortify.sca.LogFile=${com.fortify.sca.ProjectRoot}/sca/log/sca.log
```

## Precedence of setting properties

Fortify Static Code Analyzer uses properties settings in a specific order. You can override any previously set properties with the values that you specify. Keep this order in mind when making changes to the properties files.

The following table lists the order of precedence for Fortify Static Code Analyzer properties.

| Order | Property specification | Description |
|---|---|---|
| 1 | Command line with the `-D` option | Properties specified on the command line have the highest priority and you can specify them in any scan. |
| 2 | Fortify Static Code Analyzer quick scan configuration file | **Note:** You can specify either quick scan or a scan precision level. Therefore, these property settings both have second priority.<br>Properties specified in the quick scan configuration file (`fortify-sca-quickscan.properties`) have the second priority, but only if you include the `-quick` option to enable quick scan mode. |

| Order | Property specification | Description |
|---|---|---|
| | Fortify Static Code Analyzer scan precision property files | Properties specified in the scan precision property files have the second priority, but only if you include the `-scan-precision` option to enable scan precision. |
| 3 | Fortify Static Code Analyzer configuration file | Properties specified in the Fortify Static Code Analyzer configuration file (`fortify-sca.properties`) have the lowest priority. Edit this file to change the property values on a more permanent basis for all scans. |

Fortify Static Code Analyzer also relies on some properties that have internally defined default values.

# fortify-sca.properties

The following sections describe the properties available for use in the `fortify-sca.properties` file. See "fortify-sca-quickscan.properties" on page 213 for additional properties that you can use in this properties file. Each property description includes the value type, the default value, the equivalent command-line option (if applicable), and an example.

## Translation and analysis phase properties

The properties for the `fortify-sca.properties` file in the following table are general properties that apply to the translation and/or analysis (scan) phase.

| Property name | Description |
|---|---|
| **Translation and scan** | |
| `com.fortify.sca.BuildID` | Specifies the build ID of the build.<br>**Value type:** String<br>**Default:** (none)<br>**Command-line option:** `-b` |
| `com.fortify.sca.CmdlineOptionsFileEncoding` | Specifies the encoding of the command-line options file provided with `@<filename>` (see "Other options" on page 142). You can use this property, for example, to specify Unicode file paths in the options file. Valid encoding names are from the `java.nio.charset.Charset`<br>**Note:** This property is only valid in the `fortify-sca.properties` file and does not work in the `fortify-sca-quickscan.properites` file or with the `-D` option.<br>**Value type:** String<br>**Default:** JVM system default encoding |

| Property name | Description |
|---|---|
| | **Example:** `com.fortify.sca.CmdlineOptionsFileEncoding=UTF-8` |
| `com.fortify.sca.DISabledLanguages` | Specifies a colon-separated list of languages to exclude from the translation phase. The valid language values are `abap`, `actionscript`, `apex`, `cfml`, `cobol`, `configuration`, `cpp`, `dart`, `dotnet`, `golang`, `java`, `javascript`, `jsp`, `kotlin`, `objc`, `php`, `plsql`, `python`, `ruby`, `scala`, `sql`, `swift`, `tsql`, `typescript`, and `vb`.<br>**Value type:** String<br>**Default:** (none)<br>**Command-line option:** `-disable-language` |
| `com.fortify.sca.EnabledLanguages` | Specifies a colon-separated list of languages to translate. The valid language values are `abap`, `actionscript`, `apex`, `cfml`, `cobol`, `configuration`, `cpp`, `dart`, `dotnet`, `golang`, `java`, `javascript`, `jsp`, `kotlin`, `objc`, `php`, `plsql`, `python`, `ruby`, `scala`, `sql`, `swift`, `tsql`, `typescript`, and `vb`.<br>**Value type:** String<br>**Default:** All languages in the specified source are translated unless explicitly excluded with the `com.fortify.sca.DISabledLanguages` property.<br>**Command-line option:** `-enable-language` |
| `com.fortify.sca.DisableCompilerName` | If set to true, Fortify Static Code Analyzer includes build script files that have the same name as a build tool (such as gradlew) during translation as source files.<br>**Value type:** Boolean<br>**Default:** `false`<br>**Command-line option:** `-disable-compiler-resolution` |
| `com.fortify.sca.ProjectRoot` | Specifies the directory to store intermediate files generated in the translation and analysis phases. Fortify Static Code Analyzer makes extensive use of intermediate files located in this project root directory. In some cases, you achieve better performance for analysis by making sure this directory is on local storage rather than on a network drive.<br>**Value type:** String (path)<br>**Default (Windows):** `${win32.LocalAppdata}/Fortify`<br>**Note:** `${win32.LocalAppdata}` is a variable that points to the Windows Local Application Data shell folder.<br>**Default (non-Windows):** `$home/.fortify`<br>**Command-line option:** `-project-root`<br>**Example:** `com.fortify.sca.ProjectRoot=C:\Users\<username>\AppData\Local\` |
| **Translation** | |
| `com.fortify.sca.fileextensions.java` | Specifies how to translate specific file name extensions of languages that do not require build integration. The valid extension types are `ABAP`, `ACTIONSCRIPT`, `APEX`, `APEX_OBJECT`, `APEX_TRIGGER`, `ARCHIVE`, `ASPNET`, `ASP`, `ASPX`, `BITCODE`, `BSP`, `BYTECODE`, `CFML`, |

| Property name | Description |
|---|---|
| com.fortify.sca.fileextensions.cs<br>com.fortify.sca.fileextensions.js<br>com.fortify.sca.fileextensions.py<br>com.fortify.sca.fileextensions.rb<br>com.fortify.sca.fileextensions.aspx<br>com.fortify.sca.fileextensions.php<br>**Note:** This is a partial list. For the complete list, see the properties file. | COBOL, CSHARP, DART, DOCKERFILE, FLIGHT, GENERIC, GO, HOCON, HTML, INI, JAVA, JAVA_PROPERTIES, JAVASCRIPT, JSP, JSPX, KOTLIN, MSIL, MXML, OBJECT, PHP, PLSQL, PYTHON, RUBY, RUBY_ERB, SCALA, SWIFT, SWC, SWF, TLD, SQL, TSQL, TYPESCRIPT, VB, VB6, VBSCRIPT, VISUAL_FORCE, VUE, and XML.<br>**Value type:** String (valid language type)<br>**Default:** See the fortify-sca.properties file for the complete list.<br>**Examples:**<br>com.fortify.sca.fileextensions.java=JAVA<br>com.fortify.sca.fileextensions.cs=CSHARP<br>com.fortify.sca.fileextensions.js=TYPESCRIPT<br>com.fortify.sca.fileextensions.py=PYTHON<br>com.fortify.sca.fileextensions.swift=SWIFT<br>com.fortify.sca.fileextensions.razor=ASPNET<br>com.fortify.sca.fileextensions.php=PHP<br>com.fortify.sca.fileextensions.tf=HCL<br>You can also specify a value of oracle:*<path_to_script>* to programmatically supply a language type. Provide a script that accepts one command-line parameter of a file name that matches the specified extension. The script must write the valid Fortify Static Code Analyzer file type (see previous list) to stdout and exit with a return value of zero. If the script returns a non-zero return code or the script does not exist, the file is not translated and Fortify Static Code Analyzer writes a warning to the log file.<br>**Example:**<br>com.fortify.sca.fileextensions.jsp=<br>oracle:*<path_to_script>* |
| com.fortify.sca.compilers.javac=com.fortify.sca.util.compilers.JavacCompiler<br>com.fortify.sca.compilers.c++=com.fortify.sca.util.compilers.GppCompiler<br>com.fortify.sca.compilers.make=com.fortify.sca.util.compilers.TouchlessCompiler<br>com.fortify.sca.compilers.mvn=com.fortify.sca.util.compilers.MavenAdapter | Specifies custom-named compilers.<br>**Value type:** String (compiler)<br>**Default:** See the Compilers section in the fortify-sca.properties file for the complete list.<br>**Example:**<br>To tell Fortify Static Code Analyzer that “my-gcc” is a gcc compiler:<br>com.fortify.sca.<br>compilers.my-gcc=<br>com.fortify.sca.util.compilers.<br>GccCompiler<br>**Notes:**<br>• Compiler names can begin or end with an asterisk (*), which matches zero or more characters.<br>• Execution of clang/clang++ is not supported with the gcc/g++ command names. You can specify the following: com.fortify.sca.compilers.g++=com.fortify.sca.util.compilers.GppCompiler |

| Property name | Description |
|---|---|
| **Note:** This is a partial list. For the complete list, see the properties file. | |
| `com.fortify.sca.UseAntListener` | If set to true, Fortify Static Code Analyzer includes `com.fortify.dev.ant.SCAListener` in the compiler options.<br>**Value type:** Boolean<br>**Default:** `false` |
| `com.fortify.sca.exclude` | Specifies one or more files to exclude from translation. Separate multiple files with semicolons (Windows) or colons (non-Windows). See "Specifying files and directories" on page 146 for more information on how to use file specifiers.<br>**Note:** Fortify Static Code Analyzer only uses this property during translation without build integration. When you integrate with most compilers or build tools, Fortify Static Code Analyzer translates all source files that the compiler or build tool processes even if they are specified with this property. However, the Fortify Static Code Analyzer xcodebuild and MSBuild integrations do support the `-exclude` option.<br>**Value type:** String<br>**Default:** Not enabled<br>**Command-line option:** `-exclude`<br>**Example:** `com.fortify.sca.exclude=file1.x;file2.x` |
| `com.fortify.sca.InputFileEncoding` | Specifies the source file encoding type. Fortify Static Code Analyzer allows you to scan a project that contains differently encoded source files. To work with a multi-encoded project, you must specify the `-encoding` option in the translation phase, when Fortify Static Code Analyzer first reads the source code file. Fortify Static Code Analyzer remembers this encoding in the build session and propagates it into the FVDL file.<br>Typically, if you do not specify the encoding type, Fortify Static Code Analyzer uses `file.encoding` from the `java.io.InputStreamReader` constructor with no encoding parameter. In a few cases (for example with the ActionScript parser), Fortify Static Code Analyzer defaults to `UTF-8`.<br>**Value type:** String<br>**Default:** (none)<br>**Command-line option:** `-encoding`<br>**Example:**<br>`com.fortify.sca.InputFileEncoding=UTF-16` |
| `com.fortify.sca.RegExecutable` | On Windows platforms, specifies the path to the `reg.exe` system utility. Specify the paths in Windows syntax, not Cygwin syntax, even when you run Fortify Static Code Analyzer from within Cygwin. Escape backslashes with an additional backslash.<br>**Value type:** String (path) |

| Property name | Description |
|---|---|
| | **Default:** `reg`<br>**Example:**<br>`com.fortify.sca.RegExecutable=C:\\Windows\\System32\\reg.exe` |
| `com.fortify.sca.xcode.TranslateAfterError` | Specifies whether the xcodebuild touchless adapter continues translation if the xcodebuild subprocess exited with a non-zero exit code. If set to false, translation stops after encountering a non-zero xcodebuild exit code and the Fortify Static Code Analyzer touchless build halts with the same exit code. If set to true, the Fortify Static Code Analyzer touchless build executes translation of the build file identified prior to the xcodebuild exit, and Fortify Static Code Analyzer exits with an exit code of zero (unless some other error also occurs).<br>Regardless of this setting, if xcodebuild exits with a non-zero code, then the xcodebuild exit code, stdout, and stderr are written to the log file.<br>**Value type:** Boolean<br>**Default:** `false` |
| **Scan** | |
| `com.fortify.sca.AddImpliedMethods` | If set to true, Fortify Static Code Analyzer generates implied methods when it encounters implementation by inheritance.<br>**Value type:** Boolean<br>**Default:** `true` |
| `com.fortify.sca.alias.Enable` | If set to true, enables alias analysis.<br>**Value type:** Boolean<br>**Default:** `true` |
| `com.fortify.sca.analyzer.controlflow.EnableTimeOut` | Specifies whether to enable Control Flow Analyzer timeouts.<br>**Value type:** Boolean<br>**Default:** `true` |
| `com.fortify.sca.BinaryName` | Specifies a subset of source files to scan. Only the source files that were linked in the named binary at build time are included in the scan.<br>**Value type:** String (path)<br>**Default:** (none)<br>**Command-line option:** `-bin` or `-binary-name` |
| `com.fortify.sca.DefaultAnalyzers` | Specifies a comma- or colon-separated list of the types of analysis to perform. The valid values for this property are `buffer`, `content`, `configuration`, `controlflow`, `dataflow`, , `nullptr`, `semantic`, and `structural`.<br>**Value type:** String<br>**Default:** This property is commented out and all analysis types are used in scans.<br>**Command-line option:** `-analyzers` |

| Property name | Description |
|---|---|
| com.fortify.sca.DisableFunctionPointers | If set to true, disables function pointers during the scan.<br>**Value type:** Boolean<br>**Default:** false |
| com.fortify.sca.EnableAnalyzer | Specifies a comma- or colon-separated list of analyzers to use for a scan in addition to the default analyzers. The valid values for this property are buffer, content, configuration, controlflow, dataflow, nullptr, semantic, and structural.<br>**Value type:** String<br>**Default:** (none) |
| com.fortify.sca.ExitCodeLevel | Extends the default exit code options. See "Exit codes" on page 169 for a description of the exit codes and the valid values for this property. |
| com.fortify.sca.FilterFile | Specifies the path to a filter file for the scan. See "Excluding issues with filter files" on page 178 for more information.<br>**Value type:** String (path)<br>**Default:** (none)<br>**Command-line option:** -filter |
| com.fortify.sca.FilteredInstanceIDs | Specifies a comma-separated list of IIDs to be filtered out using a filter file.<br>**Value type:** String<br>**Default:** (none)<br>**Example:**<br>com.fortify.sca.FilteredInstanceIDs=CA4E1623A2424919B98EC19FCA279FFA,4418B3DC072647158B3758E6183C14CD |
| com.fortify.sca.hoa.Enable | If set to true, higher-order analysis is enabled.<br>**Value type:** Boolean<br>**Default:** true |
| com.fortify.sca.LowSeverityCutoff | Specifies the cutoff level for severity suppression. Fortify Static Code Analyzer ignores any issues found with a lower severity value than the one specified for this property.<br>**Value type:** Number<br>**Default:** 1.0 |
| com.fortify.sca.MaxPassthroughChainDepth | Specifies the length of a taint path between input and output parameters in a function call.<br>**Value type:** Integer<br>**Default:** 4 |
| com.fortify.sca.MultithreadedAnalysis | Specifies whether Fortify Static Code Analyzer runs in parallel analysis mode.<br>**Value type:** Boolean<br>**Default:** true |
| com.fortify.sca. | Specifies a comma-separated list of languages for which to run higher-order analysis. |

| Property name | Description |
|---|---|
| `Phase0HigherOrder.Languages` | Valid values are `python`, `swift`, `ruby`, `javascript`, and `typescript`.<br>**Value type:** String<br>**Default:** `python,ruby,swift,javascript,typescript` |
| `com.fortify.sca.Phase0HigherOrder.Timeout.Hard` | Specifies the total time (in seconds) for higher-order analysis. When the analyzer reaches the hard timeout limit, it exits immediately.<br>OpenText recommends this timeout limit in case some issue causes the analysis to run too long. OpenText recommends that you set the hard timeout to about 50% longer than the soft timeout, so that either the fixpoint pass limiter or the soft timeout occurs first.<br>**Value type:** Number<br>**Default:** `2700` |
| `com.fortify.sca.PrecisionLevel` | Specifies the scan precision. Scans with a lower precision level are performed faster. The valid values are 1, 2, 3, and 4.<br>**Value type:** Number<br>**Default:** (none)<br>**Command-line option:** `-scan-precision` \| `-p` |
| `com.fortify.sca.ProjectTemplate` | Specifies the issue template file to use for the scan. This only affects scans on the local machine. If you upload the FPR to Fortify Software Security Center, it uses the issue template assigned to the application version.<br>**Value type:** String<br>**Default:** (none)<br>**Command-line option:** `-project-template`<br>**Example:**<br>`com.fortify.sca.ProjectTemplate=test_issuetemplate.xml` |
| `com.fortify.sca.QuickScanMode` | If set to true, Fortify Static Code Analyzer performs a quick scan. Fortify Static Code Analyzer uses the settings from `fortify-sca-quickscan.properties`, instead of the `fortify-sca.properties` configuration file.<br>**Value type:** Boolean<br>**Default:** (not enabled)<br>**Command-line option:** `-quick` |
| `com.fortify.sca.ScanPolicy` | Specifies the scan policy for prioritizing reported vulnerabilities (see "Applying a scan policy to the analysis" on page 46). The valid scan policy values are `classic`, `security`, and `devops`.<br>**Value type:** String<br>**Default:** `security`<br>**Command-line option:** `-sc` or `-scan-policy` |

| Property name | Description |
|---|---|
| com.fortify.sca.SuppressLowSeverity | If set to true, Fortify Static Code Analyzer ignores low severity issues found in a scan.<br>**Value type:** Boolean<br>**Default:** true |
| com.fortify.sca.ThreadCount | Specifies the number of threads for parallel analysis mode. Add this property only if you need to reduce the number of threads used because of a resource constraint. If you experience an increase in scan time or problems with your scan, a reduction in the number of threads used might solve the problem.<br>**Value type:** Integer<br>**Default:** (number of available processor cores) |
| com.fortify.sca.TypeInferenceFunctionTimeout | The amount of time (in seconds) that type inference can spend to analyze a single function. Unlimited if set to zero or is not specified.<br>**Value type:** Long<br>**Default:** 60 |
| com.fortify.sca.TypeInferenceLanguages | Comma- or colon-separated list of languages that use type inference. This setting improves the precision of the analysis for dynamically-typed languages.<br>**Value type:** String<br>**Default:** javascript,python,ruby,typescript |
| com.fortify.sca.TypeInferencePhase0Timeout | Specifies the total amount of time (in seconds) that type inference can spend in phase 0 (the interprocedural analysis). Unlimited if set to zero or is not specified.<br>**Value type:** Long<br>**Default:** 300 |
| com.fortify.sca.UniversalBlacklist | Specifies a colon-separated list of functions to hide from all analyzers.<br>**Value type:** String<br>**Default:** .*yyparse.* |

# Regex analysis properties

The properties for the fortify-sca.properties file in the following table apply to regular expression analysis.

| Property name | Description |
|---|---|
| com.fortify.sca.regex.Enable | If set to true, regular expression analysis is enabled.<br>**Value type:** Boolean<br>**Default:** true |
| com.fortify.sca.regex.ExcludeBinaries | If set to true, binary files are excluded from a regular expression analysis.<br>**Value type:** Boolean |

| Property name | Description |
|---|---|
| | **Default:** `true` |
| `com.fortify.sca.regex.MaxSize` | Specifies the maximum size (in megabytes) for files that are scanned in a regular expression analysis. Files that exceed this file size maximum are excluded from a regular expression analysis.<br>**Value type:** Number<br>**Default:** `10` |

**See also**

"Regular expression analysis" on page 47

## LIM license properties

The properties for the `fortify-sca.properties` file in the following table apply to licensing with the LIM.

| Property name | Description |
|---|---|
| `com.fortify.sca.lim.Url` | Specifies the LIM server API URL. Do not edit this value directly with a text editor. Use the command-line option to change this value.<br>**Value type:** String<br>**Default:** (none)<br>**Command-line option:** `-store-license-pool-credentials`<br>**Example:** `https://<ip_address>:<port>` |
| `com.fortify.sca.lim.PoolName` | Specifies the LIM license pool name. Do not edit this value directly with a text editor. Use the command-line option to change this value.<br>**Value type:** String<br>**Default:** (none)<br>**Command-line option:** `-store-license-pool-credentials` |
| `com.fortify.sca.lim.PoolPassword` | Specifies the LIM license pool password (encrypted). Do not edit this value directly with a text editor. Use the command-line option to change this value.<br>**Value type:** String<br>**Default:** (none)<br>**Command-line option:** `-store-license-pool-credentials` |
| `com.fortify.sca.lim.ProxyUrl` | Specifies the proxy server used to connect to the LIM server.<br>**Value type:** String<br>**Default:** (none)<br>**Examples:** |

| Property name | Description |
|---|---|
| | `http://proxy.example.com:8080`<br>`https://proxy.example.com`<br>**Command-line option:** `-store-license-pool-credentials` |
| `com.fortify.sca.lim.ProxyUsername` | Specifies an encrypted user name for proxy authentication to connect to the LIM server. Do not edit this value directly with a text editor. Use the command-line option to change this value.<br>**Value type:** String<br>**Default:** (none)<br>**Command-line option:** `-store-license-pool-credentials` |
| `com.fortify.sca.lim.ProxyPassword` | Specifies an encrypted password for proxy authentication to connect to the LIM server. Do not edit this value directly with a text editor. Use the command-line option to change this value.<br>**Value type:** String<br>**Default:** (none)<br>**Command-line option:** `-store-license-pool-credentials` |
| `com.fortify.sca.lim.RequireTrustedSSLCert` | If set to true, any attempt to connect to the LIM server without a trusted certificate fails. If this property is set to false, a warning message displays for any attempt to connect to the LIM server without a trusted certificate.<br>**Value type:** Boolean<br>**Default:** `true` |
| `com.fortify.sca.lim.WaitForInitialLicense` | If set to true and LIM license pool credentials are stored, Fortify Static Code Analyzer waits for a LIM license to become available before starting a translation or scan. If this property is set to false, Fortify Static Code Analyzer aborts if it cannot obtain a LIM license.<br>**Value type:** Boolean<br>**Default:** `true` |

### See also

"LIM license directives" on page 145

## Rule properties

The properties for the `fortify-sca.properties` file in the following table apply to rules (and custom rules) and Rulepacks.

| Property name | Description |
|---|---|
| `com.fortify.sca.DefaultRulesDir` | Sets the directory used to search for the OpenText provided encrypted rules files.<br>**Value Type:** String (path) |

| Property name | Description |
|---|---|
| | **Default:**<br>`${com.fortify.Core}/config/rules` |
| `com.fortify.sca.RulesFile` | Specifies a custom Rulepack or directory. If you specify a directory, all of the files in the directory with the `.bin` and `.xml` extensions are included.<br>**Value Type:** String (path)<br>**Default:** (none)<br>**Command-line option:** `-rules` |
| `com.fortify.sca.CustomRulesDir` | Sets the directory used to search for custom rules.<br>**Value Type:** String (path)<br>**Default:**<br>`${com.fortify.Core}/config/customrules` |
| `com.fortify.sca.RulesFileExtensions` | Specifies a list of file extensions for rules files. Any files in *<sca_install_dir>*`/Core/config/rules` (or a directory specified with the `-rules` option) whose extension is in this list is included. The `.bin` extension is always included, regardless of the value of this property. The delimiter for this property is the system path separator.<br>**Value Type:** String<br>**Default:** `.xml` |
| `com.fortify.sca.NoDefaultRules` | If set to true, rules from the default Rulepacks are not loaded. Fortify Static Code Analyzer processes the Rulepacks for description elements and language libraries, but no rules are processed.<br>**Value Type:** Boolean<br>**Default:** (none)<br>**Command-line option:** `-no-default-rules` |
| `com.fortify.sca.NoDefaultIssueRules` | If set to true, disables rules in default Rulepacks that lead directly to issues. Fortify Static Code Analyzer still loads rules that characterize the behavior of functions. This can be helpful when creating custom issue rules.<br>**Value Type:** Boolean<br>**Default:** (none)<br>**Command-line option:** `-no-default-issue-rules` |
| `com.fortify.sca.NoDefaultSourceRules` | If set to true, disables source rules in the default Rulepacks. This can be helpful when creating custom source rules.<br>**Note:** Characterization source rules are not disabled.<br>**Value Type:** Boolean<br>**Default:** (none)<br>**Command-line option:** `-no-default-source-rules` |

| Property name | Description |
|---|---|
| com.fortity.sca.<br>NoDefaultSinkRules | If set to true, disables sink rules in the default Rulepacks. This can be helpful when creating custom sink rules.<br>**Note:** Characterization sink rules are not disabled.<br>**Value Type:** Boolean<br>**Default:** (none)<br>**Command-line option:** `-no-default-sink-rules` |

## Java and Kotlin properties

The properties for the `fortify-sca.properties` file in the following table apply to the translation of Java and Kotlin code.

| Property name | Description |
|---|---|
| com.fortify.sca.<br>JavaClasspath | Specifies the class path used to analyze Java or Kotlin source code. Separate multiple paths with semicolons (Windows) or colons (non-Windows).<br>**Value type:** String (paths)<br>**Default:** (none)<br>**Command-line option:** `-cp` or `-classpath` |
| com.fortify.sca.<br>JdkVersion | Specifies the Java source code version for Java or Kotlin translation.<br>**Value type:** String<br>**Default:** `11`<br>**Command-line option:** `-jdk` or `-source` |
| com.fortify.sca.<br>CustomJdkDir | Specifies a directory that contains a JDK version that is not included in the Fortify Static Code Analyzer installation (*<sca_install_dir>*`/Core/bootcp/`).<br>**Value type:** String (path)<br>**Default:** (none)<br>**Command-line option:** `-custom-jdk-dir` |
| com.fortify.sca.<br>JavaSourcepath | Specifies a semicolon- (Windows) or colon-separated (non-Windows) list of Java or Kotlin source file directories that are not included in the scan but are used for name resolution. The source path is similar to class path, except it uses source files rather than class files for resolution.<br>**Value type:** String (paths)<br>**Default:** (none)<br>**Command-line option:** `-sourcepath` |
| com.fortify.sca.<br>Appserver | Specifies the application server to process JSP files. The valid values are `weblogic` or `websphere`. |

| Property name | Description |
|---|---|
| | **Value type:** String<br>**Default:** (none)<br>**Command-line option:** `-appserver` |
| `com.fortify.sca.AppserverHome` | Specifies the application server's home directory. For WebLogic, this is the path to the directory that contains `server/lib`. For WebSphere, this is the path to the directory that contains the `JspBatchCompiler` script.<br>**Value type:** String (path)<br>**Default:** (none)<br>**Command-line option:** `-appserver-home` |
| `com.fortify.sca.AppserverVersion` | Specifies the version of the WebLogic or WebSphere application server.<br>**Value type:** String<br>**Default:** (none)<br>**Command-line option:** `-appserver-version` |
| `com.fortify.sca.JavaExtdirs` | Specifies directories to include implicitly on the class path for WebLogic and WebSphere application servers.<br>**Value type:** String<br>**Default:** (none)<br>**Command-line option:** `-extdirs` |
| `com.fortify.sca.JavaSourcepathSearch` | If set to true, Fortify Static Code Analyzer only translates Java source files that are referenced by the target file list. Otherwise, Fortify Static Code Analyzer translates all files included in the source path.<br>**Value type:** Boolean<br>**Default:** `true` |
| `com.fortify.sca.DefaultJarsDirs` | Specifies semicolon- or colon-separated list of directories of commonly used JAR files. JAR files located in these directories are appended to the end of the class path option (`-cp`).<br>**Value type:** String<br>**Default:** `default_jars` |
| `com.fortify.sca.DecompileBytecode` | If set to true, Java bytecode is decompiled for the translation.<br>**Value type:** Boolean<br>**Default:** `false` |
| `com.fortify.sca.jsp.UseSecurityManager` | If set to true, the JSP parser uses JSP security manager.<br>**Value type:** Boolean<br>**Default:** `true` |
| `com.fortify.sca.jsp.DefaultEncoding` | Specifies the encoding for JSPs. |

| Property name | Description |
|---|---|
| | **Value type:** String (encoding)<br>**Default:** `ISO-8859-1` |
| `com.fortify.sca.jsp.LegacyDataflow` | If set to true, enables additional filtering on JSP-related dataflow to reduce the amount of spurious false positives detected.<br>**Value type:** Boolean<br>**Default:** `false`<br>**Command-line option:** `-legacy-jsp-dataflow` |
| `com.fortify.sca.KotlinJvmDefault` | Specifies the generation of the `DefaultImpls` class for methods with bodies in Kotlin interfaces. The valid values are:<br>• `disable`—Specifies to generate the `DefaultImpls` class for each interface that contains methods with bodies.<br>• `all`—Specifies to generate the `DefaultImpls` class if an interface is annotated with `@JvmDefaultWithCompatibility`.<br>• `all-compatibility`—Specifies to generate the `DefaultImpls` class unless an interface is annotated with `@JvmDefaultWithoutCompatibility`.<br>**Value type:** String<br>**Default:** `disable` |
| `com.fortify.sca.ShowUnresolvedSymbols` | If set to true, displays any unresolved types, fields, and functions referenced in translated Java source files at the end of the translation.<br>**Value type:** Boolean<br>**Default:** `false`<br>**Command-line option:** `-show-unresolved-symbols` |

### See also

"Translating Java code" on page 50

"Translating Kotlin code" on page 58

## Visual Studio and MSBuild project properties

The properties for the `fortify-sca.properties` file in the following table apply to the translation of .NET projects and solutions.

| Property name | Description |
|---|---|
| `WinForms.TransformDataBindings`<br>`WinForms.TransformMessageLoops` | Sets various .NET options.<br>**Value type:** Boolean and String<br>**Defaults and examples:** |

| Property name | Description |
|---|---|
| `WinForms.TransformChangeNotificationPattern`<br>`WinForms.CollectionMutationMonitor.Label`<br>`WinForms.ExtractEventHandlers` | `WinForms.TransformDataBindings=true`<br>`WinForms.TransformMessageLoops=true`<br>`WinForms.TransformChangeNotificationPattern=true`<br>`WinForms.CollectionMutationMonitor.Label=WinFormsDataSource`<br>`WinForms.ExtractEventHandlers=true` |
| `com.fortify.sca.ASPVirtualRoots.<virtual_path>` | Specifies a semicolon-separated list of full paths to virtual roots used.<br>**Value type:** String<br>**Default:** (none)<br>**Example:**<br>`com.fortify.sca.ASPVirtualRoots.Library=c:\\WebServer\\CustomerTwo\\Stuff`<br>`com.fortify.sca.ASPVirtualRoots.Include=c:\\WebServer\\CustomerOne\\inc` |
| `com.fortify.sca.DisableASPExternalEntries` | If set to true, disables ASP external entries in the scan.<br>**Value type:** Boolean<br>**Default:** `false` |

### See also

"Translating Visual Studio projects" on page 62

## JavaScript and TypeScript properties

The properties for the `fortify-sca.properties` file in the following table apply to the translation of JavaScript and TypeScript code.

| Property name | Description |
|---|---|
| `com.fortify.sca.EnableDOMModeling` | If set to true, Fortify Static Code Analyzer generates JavaScript code to model the DOM tree that an HTML file generated during the translation phase and identifies DOM-related issues (such as cross-site scripting issues). Enable this property if the code you are translating includes HTML files that have embedded or referenced JavaScript code.<br>**Note:** Enabling this property can increase the translation time.<br>**Value type:** Boolean<br>**Default:** `false` |
| `com.fortify.sca.DOMModeling.tags` | If you set the `com.fortify.sca.EnableDOMModeling` property to true, you can specify additional coma-separated HTML tags names for Fortify Static Code Analyzer to include in the DOM modeling. |

| Property name | Description |
|---|---|
| | **Value type:** String<br>**Default:** `body`, `button`, `div`, `form`, `iframe`, `input`, `head`, `html`, and `p`.<br>**Example:**<br>`com.fortify.sca.DOMModeling.tags=ul,li` |
| `com.fortify.sca.JavaScript.src.domain.whitelist` | Specifies trusted domain names where Fortify Static Code Analyzer can download referenced JavaScript files for the scan. Delimit the URLs with vertical bars.<br>**Value type:** String<br>**Default:** (none)<br>**Example:** `com.fortify.sca.JavaScript.src.domain.whitelist=http://www.xyz.com\|http://www.123.org` |
| `com.fortify.sca.DisableJavascriptExtraction` | If set to true, JavaScript code embedded in JSP, JSPX, PHP, and HTML files is not extracted and not scanned.<br>**Value type:** Boolean<br>**Default:** `false` |
| `com.fortify.sca.EnableTranslationMinifiedJS` | If set to true, enables translation for minified JavaScript files.<br>**Value type:** Boolean<br>**Default:** `false` |
| `com.fortify.sca.skip.libraries.ES6`<br>`com.fortify.sca.skip.libraries.jQuery`<br>`com.fortify.sca.skip.libraries.javascript`<br>`com.fortify.sca.skip.libraries.typescript` | Specifies a list of comma- or colon-separated JavaScript or TypeScript technology library files that are not translated. You can use regular expressions in the file names. Note that the regular expression `'(-\d\.\d\.\d)?'` is automatically inserted before `.min.js` or `.js` for each file name included in the `com.fortify.sca.skip.libraries.jQuery` property value.<br>**Value type:** String<br>**Defaults:**<br>• ES6: `es6-shim.min.js,system-polyfills.js,shims_for_IE.js`<br>• jQuery: `jquery.js,jquery.min.js,jquery-migrate.js,jquery-migrate.min.js,jquery-ui.js,jquery-ui.min.js,jquery.mobile.js,jquery.mobile.min.js,jquery.color.js,jquery.color.min.js,jquery.color.svg-names.js,jquery.color.svg-names.min.js,jquery.color.plus-names.js,jquery.color.plus-names.min.js,jquery.tools.min.js`<br>• javascript: `bootstrap.js,bootstrap.min.js,` |

| Property name | Description |
|---|---|
| | `typescript.js`, `typescriptServices.js`<br>• typescript: `typescript.d.ts`, `typescriptServices.d.ts` |
| `com.fortify.sca.follow.imports` | If set to true, files included with an import statement are included in the translation.<br>**Value type:** Boolean<br>**Default:** `true` |
| `com.fortify.sca.exclude.node.modules` | If set to true, files in a node_modules directory are excluded from the analysis phase.<br>**Value type:** Boolean<br>**Default:** `true` |
| `com.fortify.sca.exclude.unimported.node.modules` | Specifies whether to exclude source code in a node_modules directory. If set to true, only imported node_modules are included in the translation.<br>**Note:** This property is only applied if `com.fortify.sca.exclude.node.modules` is set to false.<br>**Value type:** Boolean<br>**Default:** `true` |

**See also**

"Translating JavaScript and TypeScript code" on page 71

# Python properties

The properties for the `fortify-sca.properties` file in the following table apply to the translation of Python code.

| Property name | Description |
|---|---|
| `com.fortify.sca.PythonPath` | Specifies a semicolon-separated (Windows) or colon-separated (non-Windows) list of additional import directories. Fortify Static Code Analyzer does not respect PYTHONPATH environment variable that the Python runtime system uses to find import files. Use this property to specify the additional import directories.<br>**Value type:** String (path)<br>**Default:** (none)<br>**Command-line option:** `-python-path` |
| `com.fortify.sca.PythonVersion` | Specifies the Python source code version to scan. The valid values are `2` and `3`.<br>**Value type:** Number |

| Property name | Description |
|---|---|
| | **Default:** 3<br>**Command-line option:** `-python-version` |
| `com.fortify.sca.PythonNoAutoRootCalculation` | If set to true, disables the automatic calculation of a common root directory of all project files to use for importing modules and packages For more details, see "Including imported modules and packages" on page 80.<br>**Value type:** Boolean<br>**Default:** `false`<br>**Command-line option:** `-python-no-auto-root-calculation` |
| `com.fortify.sca.DjangoTemplateDirs` | Specifies semicolon-separated (Windows) or colon-separated (non-Windows) list of directories for Django templates. Fortify Static Code Analyzer does not use the TEMPLATE_DIRS setting from the Django `settings.py` file.<br>**Value type:** String (paths)<br>**Default:** (none)<br>**Command-line option:** `-django-template-dirs` |
| `com.fortify.sca.DjangoDisableAutodiscover` | Specifies that Fortify Static Code Analyzer does not automatically discover Django templates.<br>**Value type:** Boolean<br>**Default:** (none)<br>**Command-line option:** `-django-disable-autodiscover` |
| `com.fortify.sca.JinjaTemplateDirs` | Specifies semicolon-separated (Windows) or colon-separated (non-Windows) list of directories for Jinja2 templates.<br>**Value type:** String (paths)<br>**Default:** (none)<br>**Command-line option:** `-jinja-template-dirs` |
| `com.fortify.sca.DisableTemplateAutodiscover` | Specifies that Fortify Static Code Analyzer does not automatically discover Django or Jinja2 templates.<br>**Value type:** Boolean<br>**Default:** (none)<br>**Command-line option:** `-disable-template-autodiscover` |

### See also

"Translating Python code" on page 77

# Go properties

The properties for the `fortify-sca.properties` file in the following table apply to the translation of Go code.

| Property name | Description |
|---|---|
| `com.fortify.sca.gotags` | Specifies custom build tags for a Go project. This is equivalent to the `-tags` option for the `go` command.<br>**Value type:** String<br>**Default:** (none)<br>**Command-line option:** -gotags |
| `com.fortify.sca.GOPATH` | Specifies the root directory of your project/workspace.<br>**Value type:** String<br>**Default:** (GOPATH system environment variable) |
| `com.fortify.sca.GOROOT` | Specifies the location of the Go installation.<br>**Value type:** String<br>**Default:** (GOROOT system environment variable) |
| `com.fortify.sca.GOPROXY` | Specifies one or more comma-separated proxy URLs. You can also specify `direct` or `off`.<br>**Value type:** String<br>**Default:** (GOPROXY system environment variable) |

**See also**

"Translating Go code" on page 85

# Ruby properties

The properties for the `fortify-sca.properties` file in the following table apply to the translation of Ruby code.

| Property name | Description |
|---|---|
| `com.fortify.sca.RubyLibraryPaths` | Specifies one or more paths to directories that contain Ruby libraries.<br>**Value type:** String (path)<br>**Default:** (none)<br>**Command-line option:** `-ruby-path` |
| `com.fortify.sca.RubyGemPaths` | Specifies one or more paths to RubyGems locations. Set this value if the project has associated gems to scan.<br>**Value type:** String (path) |

| Property name | Description |
|---|---|
| | **Default:** (none)<br>**Command-line option:** `-rubygem-path` |

**See also**

"Translating Ruby code" on page 91

## COBOL properties

The properties for the `fortify-sca.properties` file in the following table apply to the translation of COBOL code.

| Property name | Description |
|---|---|
| `com.fortify.sca.CobolCopyDirs` | Specifies one or more semicolon- or colon-separated directories where Fortify Static Code Analyzer looks for copybook files.<br>**Value type:** String (path)<br>**Default:** (none)<br>**Command-line option:** `-copydirs` |
| `com.fortify.sca.CobolDialect` | Specifies the COBOL dialect. The valid values for dialect are `COBOL390` or `MICROFOCUS`. The dialect value is case-insensitive.<br>**Value type:** String<br>**Default:** COBOL390<br>**Command-line option:** `-dialect` |
| `com.fortify.sca.CobolCheckerDirectives` | Specifies one or more semicolon-separated COBOL checker directives.<br>**Value type:** String<br>**Default:** (none)<br>**Command-line option:** `-checker-directives` |
| `com.fortify.sca.CobolLegacy` | If set to true, enables legacy COBOL translation.<br>**Value type:** Boolean<br>**Default:** `false`<br>**Command-line option:** `-cobol-legacy` |
| `com.fortify.sca.CobolFixedFormat` | If set to true, specifies fixed-format COBOL to direct Fortify Static Code Analyzer to only look for source code between columns 8-72 in all lines of code (legacy COBOL translation only).<br>**Value type:** Boolean<br>**Default:** `false`<br>**Command-line option:** `-fixed-format` |

| Property name | Description |
|---|---|
| com.fortify.sca.CobolCopyExtensions | Specifies one or more semicolon- or colon-separated copybook file extensions (legacy COBOL translation only).<br>**Value type:** String<br>**Default:** (none)<br>**Command-line option:** -copy-extensions |

**See also**

"Translating COBOL code" on page 93

## PHP properties

The properties for the fortify-sca.properties file in the following table apply to the translation of PHP code.

| Property name | Description |
|---|---|
| com.fortify.sca.PHPVersion | Specifies the PHP version. For a list of valid versions, see the *Fortify Software System Requirements* document.<br>**Value type:** String<br>**Default:** 8.2<br>**Command-line option:** -php-version |
| com.fortify.sca.PHPSourceRoot | Specifies the PHP source root.<br>**Value type:** Boolean<br>**Default:** (none)<br>**Command-line option:** -php-source-root |

**See also**

"Translating PHP code" on page 101

## ABAP properties

The properties described in the following table apply to the translation of ABAP code.

| Property name | Description |
|---|---|
| com.fortify.sca.AbapDebug | If set to true, Fortify Static Code Analyzer adds ABAP statements to debug messages.<br>**Value type:** Boolean<br>**Default:** (none) |
| com.fortify.sca. | When Fortify Static Code Analyzer encounters an ABAP 'INCLUDE' directive, it looks in the named |

| Property name | Description |
|---|---|
| AbapIncludes | directory.<br>**Value type:** String (path)<br>**Default:** (none) |

## Flex and ActionScript properties

The properties for the fortify-sca.properties file in the following table apply to the translation of Flex and ActionScript code.

| Property name | Description |
|---|---|
| com.fortify.sca.FlexLibraries | Specifies a semicolon-separated (Windows) or colon-separated (non-Windows) of libraries to "link" to. This list must include flex.swc, framework.swc, and playerglobal.swc (which are usually located in the frameworks/libs directory in your Flex SDK root). Use this property primarily to resolve ActionScript.<br>**Value type:** String (path)<br>**Default:** (none)<br>**Command-line option:** -flex-libraries |
| com.fortify.sca.FlexSdkRoot | Specifies the root location of a valid Flex SDK. The folder must contain a frameworks folder that contains a flex-config.xml file. It must also contain a bin folder that contains an mxmlc executable.<br>**Value type:** String (path)<br>**Default:** (none)<br>**Command-line option:** -flex-sdk-root |
| com.fortify.sca.FlexSourceRoots | Specifies any additional source directories for a Flex project. Separate multiple directories with semicolons (Windows) or colons (non-Windows).<br>**Value type:** String (path)<br>**Default:** (none)<br>**Command-line option:** -flex-source-root |

## ColdFusion (CFML) properties

The properties for the fortify-sca.properties file in the following table apply to the translation of CFML code.

| Property name | Description |
|---|---|
| com.fortify.sca.CfmlUndefinedVariablesAreTainted | If set to true, Fortify Static Code Analyzer treats undefined variables in CFML pages as tainted. This serves as a hint to the Dataflow Analyzer to watch out for register-globals-style vulnerabilities. However, enabling this property |

| Property name | Description |
|---|---|
| | interferes with dataflow findings where a variable in an included page is initialized to a tainted value in an earlier-occurring included page.<br>**Value type:** Boolean<br>**Default:** `false` |
| `com.fortify.sca.CaseInsensitiveFiles` | If set to true, make CFML files case-insensitive for applications developed using a case-insensitive file system and scanned on case-sensitive file systems.<br>**Value type:** Boolean<br>**Default:** (not enabled) |
| `com.fortify.sca.SourceBaseDir` | Specifies the base directory for ColdFusion projects.<br>**Value type:** String (path)<br>**Default:** (none)<br>**Command-line option:** `-source-base-dir` |

**See also**

"Translating ColdFusion code" on page 112

## SQL properties

The properties for the `fortify-sca.properties` file in the following table apply to the translation of SQL code.

| Property name | Description |
|---|---|
| `com.fortify.sca.SqlLanguage` | Specifies the SQL language variant. The valid SQL language type values are `PLSQL` (for Oracle PL/SQL) and `TSQL` (for Microsoft T-SQL).<br>**Value type:** String<br>**Default:** `TSQL`<br>**Command-line option:** `-sql-language` |

**See also**

"Analyzing SQL" on page 113

# Output properties

The properties for the `fortify-sca.properties` file in the following table apply to the analysis output.

| Property name | Description |
|---|---|
| `com.fortify.sca.ResultsFile` | The file to which results are written.<br>**Value type:** String<br>**Default:** (none)<br>**Command-line option:** `-f`<br>**Example:** `com.fortify.sca.ResultsFile=MyResults.fpr` |
| `com.fortify.sca.Renderer` | Controls the output format. The valid values are `fpr`, `fvdl`, `text`, and `auto`. The default of `auto` selects the output format based on the extension of the file provided with the `-f` option.<br>**Value type:** String<br>**Default:** `auto`<br>**Command-line option:** `-format` |
| `com.fortify.sca.OutputAppend` | If set to true, Fortify Static Code Analyzer appends results to an existing results file.<br>**Value type:** Boolean<br>**Default:** `false`<br>**Command-line option:** `-append` |
| `com.fortify.sca.ResultsAsAvailable` | If set to true, Fortify Static Code Analyzer prints results as they become available. This is helpful if you do not specify the `-f` option (to specify an output file) and print to stdout.<br>**Value type:** Boolean<br>**Default:** `false` |
| `com.fortify.sca.BuildLabel` | Specifies a label for the scanned project. Fortify Static Code Analyzer does not use this label but includes it in the results.<br>**Value type:** String<br>**Default:** (none)<br>**Command-line option:** `-build-label` |
| `com.fortify.sca.BuildProject` | Specifies a name for the scanned project. Fortify Static Code Analyzer does not use this name but includes it in the results.<br>**Value type:** String<br>**Default:** (none)<br>**Command-line option:** `-build-project` |

| Property name | Description |
|---|---|
| com.fortify.sca.BuildVersion | Specifies a version number for the scanned project. Fortify Static Code Analyzer does not use this version number but it is included in the results.<br>**Value type:** String<br>**Default:** (none)<br>**Command-line option:** -build-version |
| com.fortify.sca.MachineOutputMode | Output information in a format that scripts or Fortify Static Code Analyzer tools can use rather than printing output interactively. Instead of a single line to display scan progress, a new line is printed below the previous one on the console to display updated progress.<br>**Value type:** Boolean<br>**Default:** (not enabled)<br>**Command-line option:** -machine-output |
| com.fortify.sca.SnippetContextLines | Sets the number of lines of code to display surrounding an issue. Snippets always include the two lines of code on each side of the line where the error occurs. By default, five lines of code are displayed.<br>**Value type:** Number<br>**Default:** 2 |
| com.fortify.sca.FVDLDisableDescriptions | If set to true, excludes Fortify security content descriptions from the analysis results file (FVDL).<br>**Value type:** Boolean<br>**Default:** false<br>**Command-line option:** -fvdl-no-descriptions |
| com.fortify.sca.FVDLDisableEngineData | If set to true, excludes engine data from the analysis results file (FVDL).<br>**Value type:** Boolean<br>**Default:** false<br>**Command-line option:**-fvdl-no-enginedata |
| com.fortify.sca.FVDLDisableLabelEvidence | If set to true, excludes label evidence from the analysis results file (FVDL).<br>**Value type:** Boolean<br>**Default:** false |
| com.fortify.sca.FVDLDisableProgramData | If set to true, excludes the ProgramData section from the analysis results file (FVDL).<br>**Value type:** Boolean<br>**Default:** false<br>**Command-line option:** -fvdl-no-progdata |
| com.fortify.sca.FVDLDisableSnippets | If set to true, excludes code snippets from the analysis results file (FVDL).<br>**Value type:** Boolean |

| Property name | Description |
|---|---|
| | **Default:** `false`<br>**Command-line option:** `-fvdl-no-snippets` |
| `com.fortify.sca.FVDLStylesheet` | Specifies location of the style sheet for the analysis results.<br>**Value type:** String (path)<br>**Default:** `${com.fortify.Core}/resources/sca/fvdl2html.xsl` |

## Mobile build session (MBS) properties

The properties for the `fortify-sca.properties` file in the following table apply to MBS files.

| Property name | Description |
|---|---|
| `com.fortify.sca.MobileBuildSessions` | If set to true, Fortify Static Code Analyzer copies source files into the build session directory.<br>**Value type:** Boolean<br>**Default:** `false` |
| `com.fortify.sca.ExtractMobileInfo` | If set to true, Fortify Static Code Analyzer extracts the build ID and the Fortify Static Code Analyzer version number from the mobile build session.<br>**Note:** Fortify Static Code Analyzer does not extract the mobile build with this property.<br>**Value type:** Boolean<br>**Default:** `false` |

**See also**

"Mobile build sessions" on page 44

## Proxy properties

The properties for the `fortify-sca.properties` file in the following table apply to proxy settings.

| Property name | Description |
|---|---|
| `com.fortify.sca.https.proxyHost` | Specifies a proxy host name.<br>**Value type:** String<br>**Default:** (none) |
| `com.fortify.sca.https.proxyPort` | Specifies a proxy port number.<br>**Value type:** Number<br>**Default:** (none) |

# Logging properties

The properties for the `fortify-sca.properties` file in the following table apply to log files.

| Property name | Description |
|---|---|
| `com.fortify.sca.LogFile` | Specifies the default log file name and location.<br>**Value type:** String (path)<br>**Default:** `${com.fortify.sca.ProjectRoot}/log/sca.log` and `${com.fortify.sca.ProjectRoot}/log/sca_FortifySupport.log`<br>**Command-line option:** `-logfile` |
| `com.fortify.sca.LogLevel` | Specifies the minimum log level for both log files. The valid values are `DEBUG`, `INFO`, `WARN`, `ERROR`, and `FATAL`. For more information, see "Locating the log files" on page 175 and "Configuring log files" on page 175.<br>**Value type:** String<br>**Default:** `INFO` |
| `com.fortify.sca.ClobberLogFile` | If set to true, Fortify Static Code Analyzer overwrites the log file for each run of sourceanalyzer.<br>**Value type:** Boolean<br>**Default:** `false`<br>**Command-line option:** `-clobber-log` |
| `com.fortify.sca.PrintPerformanceDataAfterScan` | If set to true, Fortify Static Code Analyzer writes performance-related data to the Static Code Analyzer Support log file after the scan is complete. This value is automatically set to true when in debug mode.<br>**Value type:** Boolean<br>**Default:** `false` |

### See also

"Configuring log files" on page 175

# Debug properties

The properties for the `fortify-sca.properties` file in the following table apply to debug settings.

| Property name | Description |
|---|---|
| `com.fortify.sca.Debug` | Includes debug information in the Static Code Analyzer Support log file, which is only useful for Customer Support to help troubleshoot.<br>**Value type:** Boolean |

| Property name | Description |
|---|---|
| | **Default:** `false`<br>**Command-line option:** `-debug` |
| `com.fortify.sca.DebugVerbose` | This is the same as the `com.fortify.sca.Debug` property, but it includes more details, specifically for parse errors.<br>**Value type:** Boolean<br>**Default:** (not enabled)<br>**Command-line option:** `-debug-verbose` |
| `com.fortify.sca.Verbose` | If set to true, includes verbose messages in the Static Code Analyzer Support log file.<br>**Value type:** Boolean<br>**Default:** `false`<br>**Command-line option:** `-verbose` |
| `com.fortify.sca.DebugTrackMem` | If set to true, additional performance information is written to the Static Code Analyzer Support log.<br>**Value type:** Boolean<br>**Default:** (not enabled)<br>**Command-line option:** `-debug-mem` |
| `com.fortify.sca.CollectPerformanceData` | If set to true, enables additional timers to track performance.<br>**Value type:** Boolean<br>**Default:** (not enabled) |
| `com.fortify.sca.Quiet` | If set to true, disables the command-line progress information.<br>**Value type:** Boolean<br>**Default:** `false`<br>**Command-line option:** `-quiet` |
| `com.fortify.sca.MonitorSca` | If set to true, Fortify Static Code Analyzer monitors its memory use and warns when JVM garbage collection becomes excessive.<br>**Value type:** Boolean<br>**Default:** `true` |

# fortify-sca-quickscan.properties

Fortify Static Code Analyzer offers a less in-depth scan known as a quick scan. This option scans the project in quick scan mode, using the property values in the `fortify-sca-quickscan.properties` file. By default, a quick scan reduces the depth of the analysis and applies the Quick View filter set. The Quick View filter set provides only critical and high priority issues.

**Note:** Properties in this file are only used if you specify the `-quick` option on the command line for your scan.

The following table provides two sets of default values: the default value for quick scans and the default value for normal scans. If only one default value is shown, the value is the same for both normal scans and quick scans.

| Property name | Description |
|---|---|
| `com.fortify.sca.CtrlflowMaxFunctionTime` | Sets the time limit (in milliseconds) for Control Flow analysis on a single function.<br>**Value type:** Integer<br>**Quick scan default:** `30000`<br>**Default:** `600000` |
| `com.fortify.sca.DisableAnalyzers` | Specifies a comma- or colon-separated list of analyzers to disable during a scan. The valid analyzer names are `buffer`, `content`, `configuration`, `controlflow`, `dataflow`, `nullptr`, `semantic`, and `structural`.<br>**Value type:** String<br>**Quick scan default:** `controlflow:buffer`<br>**Default:** (none) |
| `com.fortify.sca.FilterSet` | Specifies the filter set to use. You can use this property with an issue template to filter at scan-time instead of post-scan. See `com.fortify.sca.ProjectTemplate` described in "Translation and analysis phase properties" on page 186 to specify an issue template that contains the filter set to use.<br>When set to `Quick View`, this property runs rules that have a potentially high impact and a high likelihood of occurring and rules that have a potentially high impact and a low likelihood of occurring. Filtered issues are not written to the FPR and therefore this can reduce the size of an FPR. For more information about filter sets, see the *OpenText™ Fortify Audit Workbench User Guide*.<br>**Value type:** String<br>**Quick scan default:** `Quick View`<br>**Default:** (none) |
| `com.fortify.sca.FPRDisableMetatable` | Disables the creation of the metatable, which includes information for the Function view in Fortify Audit Workbench. This metatable enables right-click on a variable in the source window to show the declaration. If C/C++ scans take an extremely long time, setting this property to true can potentially reduce the scan time by hours.<br>**Value type:** Boolean<br>**Quick scan default:** `true`<br>**Default:** `false` |

| Property name | Description |
|---|---|
| | **Command-line option:** `-disable-metatable` |
| `com.fortify.sca.FPRDisableSourceBundling` | Disables source code inclusion in the FPR file. Prevents Fortify Static Code Analyzer from generating marked-up source code files during a scan. If you plan to upload FPR files that are generated as a result of a quick scan to Fortify Software Security Center, you must set this property to `false`.<br>**Value type:** Boolean<br>**Quick scan default:** `true`<br>**Default:** `false`<br>**Command-line option:** `-disable-source-bundling` |
| `com.fortify.sca.NullPtrMaxFunctionTime` | Sets the time limit (in milliseconds) for Null Pointer analysis for a single function. The standard default is five minutes. If this value is set to a shorter limit, the overall scan time decreases.<br>**Value type:** Integer<br>**Quick scan default:** `10000`<br>**Default:** `300000` |
| `com.fortify.sca.TrackPaths` | Disables path tracking for Control Flow analysis. Path tracking provides more detailed reporting for issues, but requires more scan time. To disable this for JSP only, set it to `NoJSP`. Specify `None` to disable all functions.<br>**Value type:** String<br>**Quick scan default:** (none)<br>**Default:** `NoJSP` |
| `com.fortify.sca.limiters.ConstraintPredicateSize` | Specifies the size limit for complex calculations in the Buffer Analyzer. Skips calculations that are larger than the specified size value in the Buffer Analyzer to improve scan time.<br>**Value type:** Integer<br>**Quick scan default:** `10000`<br>**Default:** `500000` |
| `com.fortify.sca.limiters.MaxChainDepth` | Controls the maximum call depth through which the Dataflow Analyzer tracks tainted data. Increase this value to increase the coverage of dataflow analysis, which results in longer scan times.<br>**Note:** Call depth refers to the maximum call depth on a dataflow path between a taint source and sink, rather than call depth from the program entry point, such as `main()`.<br>**Value type:** Integer<br>**Quick scan default:** `3`<br>**Default:** `5` |

| Property name | Description |
|---|---|
| com.fortify.sca.limiters.MaxFunctionVisits | Sets the number of times taint propagation analyzer visits functions.<br>**Value type:** Integer<br>**Quick scan default:** 5<br>**Default:** 50 |
| com.fortify.sca.limiters.MaxPaths | Controls the maximum number of paths to report for a single dataflow vulnerability. Changing this value does not change the results that are found, only the number of dataflow paths displayed for an individual result.<br>**Note:** OpenText does not recommend setting this property to a value larger than 5 because it might increase the scan time.<br>**Value type:** Integer<br>**Quick scan default:** 1<br>**Default:** 5 |
| com.fortify.sca.limiters.MaxTaintDefForVar | Sets a complexity limit for the Dataflow Analyzer. Dataflow incrementally decreases precision of analysis on functions that exceed this complexity metric for a given precision level.<br>**Value type:** Integer<br>**Quick scan default:** 250<br>**Default:** 1000 |
| com.fortify.sca.limiters.MaxTaintDefForVarAbort | Sets a hard limit for function complexity. If complexity of a function exceeds this limit at the lowest precision level, the analyzer skips analysis of the function.<br>**Value type:** Integer<br>**Quick scan default:** 500<br>**Default:** 4000 |

# fortify-rules.properties

This topic describes the properties available for use in the `fortify-rules.properties` file. Use these properties to modify behavior of individual rules or provide information that can improve how rules identify weaknesses.

| Property name | Description |
|---|---|
| com.fortify.sca.rules.password_regex.global | The regular expression to match password identifiers across all languages unless a language-specific rules property is set.<br>**Value type:** String<br>**Default:** (?i)(s\|_)?(user\|usr\|member\|admin\|guest\|login\|default\| |

| Property name | Description |
|---|---|
| | `new\|current\|old\|client\|server\|proxy\|sqlserver\|my\|mysql\|mongo\|mongodb\|db\|database\|ldap\|smtp\|email\|email(_)?smtp)?(_\|\.)?(pass(wd\|word\|phrase)\|secret)` |
| `com.fortify.sca.rules.password_regex.abap` | Regular expression to match password identifiers in ABAP code. Setting this property overrides the global regex password rules property.<br>**Value type:** String<br>**Default:** (value for `com.fortify.sca.rules.password_regex.global`) |
| `com.fortify.sca.rules.password_regex.actionscript` | Regular expression to match password identifiers in ActionScript code. Setting this property overrides the global regex password rules property.<br>**Value type:** String<br>**Default:** (value for `com.fortify.sca.rules.password_regex.global`) |
| `com.fortify.sca.rules.password_regex.apex` | Regular expression to match password identifiers in Salesforce Apex code. Setting this property overrides the global regex password rules property.<br>**Value type:** String<br>**Default:** (value for `com.fortify.sca.rules.password_regex.global`) |
| `com.fortify.sca.rules.password_regex.cfml` | Regular expression to match password identifiers in ColdFusion (CFML) code. Setting this property overrides the global regex password rules property.<br>**Value type:** String<br>**Default:** (none) |
| `com.fortify.sca.rules.password_regex.cobol` | Regular expression to match password identifiers in COBOL code. Setting this property overrides the global regex password rules property.<br>**Value type:** String<br>**Default:** (value for `com.fortify.sca.rules.password_regex.global`) |
| `com.fortify.sca.rules.password_regex.config` | Regular expression to match password identifiers in XML. Setting this property overrides the global regex password rules property. Do not use regular expression modifiers. The value is case-insensitive.<br>**Value type:** String<br>**Default:** `(s\|_)?(user\|usr\|member\|admin\|guest\|login\|default\|new\|current\|old\|client\|server\|proxy\|sqlserver\|my\|mysql\|mongo\|mongodb\|db\|database\|ldap\|smtp\|email\|email(_)?smtp)?(_\|\.)?pass(wd\|word\|phrase)` |
| `com.fortify.sca.rules.password_regex.cpp` | Regular expression to match password identifiers in C and C++ code. Setting this property overrides the global regex password rules property.<br>**Value type:** String<br>**Default:** (value for `com.fortify.sca.rules.password_regex.global`) |
| `com.fortify.sca.rules.password_regex.dart` | Regular expression to match password identifiers in Dart code. Setting this property overrides the global regex password rules property. |

| Property name | Description |
|---|---|
| | **Value type:** String<br>**Default:** (value for `com.fortify.sca.rules.password_regex.global`) |
| `com.fortify.sca.rules.password_regex.dotnet` | Regular expression to match password identifiers in .NET code. Setting this property overrides the global regex password rules property.<br>**Value type:** String<br>**Default:** (value for `com.fortify.sca.rules.password_regex.global`) |
| `com.fortify.sca.rules.password_regex.docker` | Regular expression to match password identifiers in Dockerfiles. Setting this property overrides the global regex password rules property.<br>**Value type:** String<br>**Default:** `.*pass(wd\|word\|phrase).*` |
| `com.fortify.sca.rules.password_regex.golang` | Regular expression to match password identifiers in Go code. Setting this property overrides the global regex password rules property.<br>**Value type:** String<br>**Default:** (value for `com.fortify.sca.rules.password_regex.global`) |
| `com.fortify.sca.rules.password_regex.java` | Regular expression to match password identifiers in Java code. Setting this property overrides the global regex password rules property.<br>**Value type:** String<br>**Default:** (value for `com.fortify.sca.rules.password_regex.global`) |
| `com.fortify.sca.rules.password_regex.javascript` | Regular expression to match password identifiers in JavaScript and TypeScript code. Setting this property overrides the global regex password rules property.<br>**Value type:** String<br>**Default:** (value for `com.fortify.sca.rules.password_regex.global`) |
| `com.fortify.sca.rules.password_regex.json` | Regular expression to match password identifiers in JSON. Setting this property overrides the global regex password rules property.<br>**Value type:** String<br>**Default:** `(?i).*pass(wd\|word\|phrase).*` |
| `com.fortify.sca.rules.password_regex.jsp` | Regular expression used to match password identifiers in JSP code. Setting this property overrides the global regex password rules property.<br>**Value type:** String<br>**Default:** (value for `com.fortify.sca.rules.password_regex.global`) |
| `com.fortify.sca.rules.password_regex.objc` | Regular expression to match password identifiers in Objective-C and Objective-C++ code. Setting this property overrides the global regex password rules property.<br>**Value type:** String<br>**Default:** `(?i)(s\|_)?(user\|usr\|member\|admin\|guest\|login\|default\|new\|current\|old\|client\|server\|proxy\|sqlserver\|my\|mysql\|mongo\|mongodb\|db\|database\|ldap\|smtp\|` |

| Property name | Description |
|---|---|
| | email\|email(_)?smtp)?(_\|\.)?(token\|pin\|pass(wd\|word\|phrase)) |
| com.fortify.sca.rules.password_regex.php | Regular expression to match password identifiers in PHP code. Setting this property overrides the global regex password rules property.<br>**Value type:** String<br>**Default:** (value for com.fortify.sca.rules.password_regex.global) |
| com.fortify.sca.rules.password_regex.powershell | Regular expression to match password identifiers in PowerShell files. Setting this property overrides the global regex password rules property.<br>**Value type:** String<br>**Default:** (?i)([a-z_]*\|\{.*)(pass(wd\|word\|phrase)\|pwd)(.*\}\|[a-z_]*) |
| com.fortify.sca.rules.password_regex.properties | Regular expression to match password identifiers in Properties files. Setting this property overrides the global regex password rules property.<br>**Value type:** String<br>**Default:** (value for com.fortify.sca.rules.password_regex.global) |
| com.fortify.sca.rules.password_regex.python | Regular expression to match password identifiers in Python code. Setting this property overrides the global regex password rules property.<br>**Value type:** String<br>**Default:** (value for com.fortify.sca.rules.password_regex.global) |
| com.fortify.sca.rules.password_regex.ruby | Regular expression to match password identifiers in Ruby code. Setting this property overrides the global regex password rules property.<br>**Value type:** String<br>**Default:** (value for com.fortify.sca.rules.password_regex.global) |
| com.fortify.sca.rules.password_regex.sql | Regular expression to match password identifiers in SQL code. Setting this property overrides the global regex password rules property.<br>**Value type:** String<br>**Default:** (value for com.fortify.sca.rules.password_regex.global) |
| com.fortify.sca.rules.password_regex.swift | Regular expression to match password identifiers in Swift code. Setting this property overrides the global regex password rules property.<br>**Value type:** String<br>**Default:** (?i)(s\|_)?(user\|usr\|member\|admin\|guest\|login\|default\|new\|current\|old\|client\|server\|proxy\|sqlserver\|my\|mysql\|mongo\|mongodb\|db\|database\|ldap\|smtp\|email\|email(_)?smtp)?(_\|\.)?(token\|pin\|pass(wd\|word\|phrase)) |
| com.fortify.sca.rules.password_regex.vb | Regular expression to match password identifiers in VB6 code. Setting this property overrides the global regex password rules property.<br>**Value type:** String<br>**Default:** (value for com.fortify.sca.rules.password_regex.global) |

| Property name | Description |
|---|---|
| com.fortify.sca.rules.password_regex.yaml | Regular expression to match password identifiers in YAML. Setting this property overrides the global regex password rules property.<br>**Value type:** String<br>**Default:** (?i).*pass(wd\|word\|phrase).* |
| com.fortify.sca.rules.key_regex.global | The regular expression to match key identifiers across all languages unless a language-specific regex key rules property is set.<br>**Value type:** String<br>**Default:** (?i)((enc\|dec)(ryption\|rypt)?\|crypto\|secret\|private)(_)?key |
| com.fortify.sca.rules.key_regex.abap | Regular expression to match key identifiers in ABAP code. Setting this property overrides the global regex key rules property.<br>**Value type:** String<br>**Default:** (value for com.fortify.sca.rules.key_regex.global) |
| com.fortify.sca.rules.key_regex.actionscript | Regular expression to match key identifiers in ActionScript code. Setting this property overrides the global regex key rules property.<br>**Value type:** String<br>**Default:** (value for com.fortify.sca.rules.key_regex.global) |
| com.fortify.sca.rules.key_regex.cfml | Regular expression to match key identifiers in CFML code. Setting this property overrides the global regex key rules property.<br>**Value type:** String<br>**Default:** (value for com.fortify.sca.rules.key_regex.global) |
| com.fortify.sca.rules.key_regex.cpp | Regular expression to match key identifiers in C and C++ code. Setting this property overrides the global regex key rules property.<br>**Value type:** String<br>**Default:** (value for com.fortify.sca.rules.key_regex.global) |
| com.fortify.sca.rules.key_regex.golang | Regular expression to match key identifiers in Go code. Setting this property overrides the global regex key rules property.<br>**Value type:** String<br>**Default:** (value for com.fortify.sca.rules.key_regex.global) |
| com.fortify.sca.rules.key_regex.java | Regular expression to match key identifiers in Java code. Setting this property overrides the global regex key rules property.<br>**Value type:** String<br>**Default:** (value for com.fortify.sca.rules.key_regex.global) |
| com.fortify.sca.rules.key_regex.javascript | Regular expression to match key identifiers in JavaScript and TypeScript code. Setting this property overrides the global regex key rules property.<br>**Value type:** String |

| Property name | Description |
|---|---|
| | **Default:** (value for `com.fortify.sca.rules.key_regex.global`) |
| `com.fortify.sca.rules.key_regex.jsp` | Regular expression to match key identifiers in JSP code. Setting this property overrides the global regex key rules property.<br>**Value type:** String<br>**Default:** (value for `com.fortify.sca.rules.key_regex.global`) |
| `com.fortify.sca.rules.key_regex.objc` | Regular expression used to match key identifiers in Objective-C and Objective-C++ code. Setting this property overrides the global regex key rules property.<br>**Value type:** String<br>**Default:** (value for `com.fortify.sca.rules.key_regex.global`) |
| `com.fortify.sca.rules.key_regex.php` | Regular expression to match key identifiers in PHP code. Setting this property overrides the global regex key rules property.<br>**Value type:** String<br>**Default:** (value for `com.fortify.sca.rules.key_regex.global`) |
| `com.fortify.sca.rules.key_regex.python` | Regular expression to match key identifiers in Python code. Setting this property overrides the global regex key rules property.<br>**Value type:** String<br>**Default:** (value for `com.fortify.sca.rules.key_regex.global`) |
| `com.fortify.sca.rules.key_regex.ruby` | Regular expression used to match key identifiers in Ruby code. Setting this property overrides the global regex key rules property.<br>**Value type:** String<br>**Default:** (value for `com.fortify.sca.rules.key_regex.global`) |
| `com.fortify.sca.rules.key_regex.sql` | Regular expression to match key identifiers in SQL code. Setting this property overrides the global regex key rules property.<br>**Default:** (value for `com.fortify.sca.rules.key_regex.global`) |
| `com.fortify.sca.rules.key_regex.swift` | Regular expression used to match key identifiers in Swift code. Setting this property overrides the global regex key rules property.<br>**Value type:** String<br>**Default:** (value for `com.fortify.sca.rules.key_regex.global`) |
| `com.fortify.sca.rules.key_regex.vb` | Regular expression to match key identifiers in Visual Basic 6 code. Setting this property overrides the global regex key rules property.<br>**Value type:** String<br>**Default:** (value for `com.fortify.sca.rules.key_regex.global`) |
| `com.fortify.sca.rules.GCPFunctionName` | Name of the serverless function called when no JSON/YAML cloud build config file exists.<br>**Value type:** String<br>**Default:** (none) |

| Property name | Description |
| --- | --- |
| `com.fortify.sca.rules.GCPHttpTrigger` | If set to true, the scanned cloud function is an HTTP trigger.<br>**Value type:** Boolean<br>**Default:** `false` |
| `com.fortify.sca.rules.enable_wi_correlation` | If set to true and Fortify Static Code Analyzer scans an application with a supported framework, produces a results file to be imported into OpenText™ Fortify WebInspect to improve results.<br>**Value type:** Boolean<br>**Default:** `false` |

# Appendix C: Fortify Java annotations

OpenText provides two versions of the Fortify Java annotations library.

- Annotations with the retention policy set to CLASS (`FortifyAnnotations-CLASS.jar`).
  With this version of the library, Fortify Java annotations are propagated to the bytecode during compilation.
- Annotations with the retention policy set to SOURCE (`FortifyAnnotations-SOURCE.jar`).
  With this version of the library, Fortify Java annotations are not propagated to the bytecode after the code that uses them is compiled.

If you use Fortify Software products to analyze bytecode of your applications (for example, with OpenText™ Fortify on Demand assessments), then use the version with the annotation retention policy set to CLASS. If you use Fortify Software products to analyze the source code of your applications, you can use either version of the library. However, OpenText strongly recommends that you use the library with a retention policy set to SOURCE.

**Important!** It is a security risk to leave Fortify Java annotations in production code because they can leak information about potential security problems in the code. OpenText recommends that you use annotations with the retention policy set to CLASS only for internalanalysis, and never use them in your application production builds.

This section outlines the annotations available. A sample application is included in the `Fortify_SCA_Samples_<version>.zip` archive in the `advanced/javaAnnotations` directory. A `README.txt` file included in the directory describes the sample application, problems that might arise from it, and how to fix these problems using Fortify Java annotations.

There are two limitations with Fortify Java annotations:

- Each annotation can specify only one input and/or one output.
- You can apply only one annotation of each type to the same target.

OpenText provides three main types of annotations:


- "Dataflow annotations" on the next page
- "Field and variable annotations" on page 226
- "Other annotations" on page 227


You also can write rules to support your own custom annotations. Contact Customer Support for more information.

# Dataflow annotations

There are four types of Dataflow annotations, similar to Dataflow rules: Source, Sink, Passthrough, and Validate. All are applied to methods and specify the inputs and/or outputs by parameter name or the strings `this` and `return`. Additionally, you can apply the Dataflow Source and Sink annotations to the function arguments.

## Source annotations

The acceptable values for the annotation parameter are `this`, `return`, or a function parameter name. For example, you can assign taint to an output of the target method.

```
@FortifyDatabaseSource("return")
String [] loadUserProfile(String userID) {
  ...
}
```

For example, you can assign taint to an argument of the target method.

```
void retrieveAuthCode(@FortifyPrivateSource String authCode) {
  ...
}
```

In addition to specific source annotations, OpenText provides a generic *untrusted* taint source called `FortifySource`.

The following is a complete list of source annotations:

- `FortifySource`
- `FortifyDatabaseSource`
- `FortifyFileSystemSource`
- `FortifyNetworkSource`
- `FortifyPCISource`
- `FortifyPrivateSource`
- `FortifyWebSource`

## Passthrough annotations

Passthrough annotations transfer any taint from an input to an output of the target method. It can also assign or remove taint from the output, in the case of `FortifyNumberPassthrough` and `FortifyNotNumberPassthrough`. The acceptable values for the `in` annotation parameter are `this`

or a function parameter name. The acceptable values for the `out` annotation parameter are `this`, `return`, or a function parameter name.

```
@FortifyPassthrough(in="a",out="return")
String toLowerCase(String a) {
  ...
}
```

Use `FortifyNumberPassthrough` to indicate that the data is purely numeric. Numeric data cannot cause certain types of issues, such as cross-site scripting, regardless of the source. Using `FortifyNumberPassthrough` can reduce false positives of this type. If a program decomposes character data into a numeric type (int, int[], and so on), you can use `FortifyNumberPassthrough`. If a program concatenates numeric data into character or string data, then use `FortifyNotNumberPassthrough`.

The following is a complete list of passthrough annotations:

- `FortifyPassthrough`
- `FortifyNumberPassthrough`
- `FortifyNotNumberPassthrough`

## Sink annotations

Sink annotations report an issue when taint of the appropriate type reaches an input of the target method. Acceptable values for the annotation parameter are `this` or a function parameter name.

```
@FortifyXSSSink("a")
void printToWebpage(int a) {
  ...
}
```

You can also apply the annotation to the function argument or the return parameter. In the following example, an issue is reported when taint reaches the argument `a`.

```
void printToWebpage(int b, @FortifyXSSSink String a) {
  ...
}
```

The following is a complete list of the sink annotations:

- `FortifySink`
- `FortifyCommandInjectionSink`
- `FortifyPCISink`
- `FortifyPrivacySink`

- `FortifySQLSink`
- `FortifySystemInfoSink`
- `FortifyXSSSink`

## Validate annotations

Validate annotations remove taint from an output of the target method. Acceptable values for the annotation parameter are `this`, `return`, or a function parameter name.

```
@FortifyXSSValidate("return")
String xssCleanse(String a) {
  ...
}
```

The following is a complete list of validate sink annotations:

- `FortifyValidate`
- `FortifyCommandInjectionValidate`
- `FortifyPCIValidate`
- `FortifyPrivacyValidate`
- `FortifySQLValidate`
- `FortifySystemInfoValidate`
- `FortifyXSSValidate`

# Field and variable annotations

You can apply these annotations to fields and (in most cases) variables.

## Password and private annotations

Use password and private annotations to indicate whether the target field or variable is a password or private data.

```
@FortifyPassword String x;
@FortifyNotPassword String pass;
@FortifyPrivate String y;
@FortifyNotPrivate String cc;
```

In the previous example, string x will be identified as a password and checked for privacy violations and hardcoded passwords. The string pass will not be identified as a password. Without the

annotation, it might cause false positives. The `FortifyPrivate` and `FortifyNotPrivate` annotations work similarly, only they do not cause privacy violation issues.

## Non-negative and non-zero annotations

Use these annotations to indicate disallowed values for the target field or variable.

```
@FortifyNonNegative int index;
@FortifyNonZero double divisor;
```

In the previous example, an issue is reported if a negative value is assigned to `index` or zero is assigned to `divisor`.

# Other annotations

## Check return value annotation

Use the `FortifyCheckReturnValue` annotation to add a target method to the list of functions that require a check of the return values.

```
@FortifyCheckReturnValue
int openFile(String filename) {
  ...
}
```

## Dangerous annotations

With the `FortifyDangerous` annotation, any use of the target function, field, variable, or class is reported. Acceptable values for the annotation parameter are `CRITICAL`, `HIGH`, `MEDIUM`, or `LOW`. These values indicat how to categorize the issue based on the Fortify Priority Order values).

```
@FortifyDangerous{"CRITICAL"}
public class DangerousClass {
  @FortifyDangerous{"HIGH"}
  String dangerousField;
  @FortifyDangerous{"LOW"}
  int dangerousMethod() {
    ...
  }
}
```

# Send Documentation Feedback

If you have comments about this document, you can contact the documentation team by email.

**Note:** If you are experiencing a technical issue with our product, do not email the documentation team. Instead, contact Customer Support at https://www.microfocus.com/support so they can assist you.

If an email client is configured on this computer, click the link above to contact the documentation team and an email window opens with the following information in the subject line:

**Feedback on User Guide (Fortify Static Code Analyzer 24.4.0)**

Just add your feedback to the email and click send.

If no email client is available, copy the information above to a new message in a web mail client, and send your feedback to fortifydocteam@opentext.com.

We appreciate your feedback!