Micro Focus
Fortify Static Code Analyzer

Software Version: 22.1.0

User Guide

Document Release Date: June 2022
Software Release Date: June 2022

MICRO FOCUS®

Legal Notices

Micro Focus
The Lawn
22-30 Old Bath Road
Newbury, Berkshire RG14 1QN
UK

https://www.microfocus.com

Warranty

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

Restricted Rights Legend

Confidential computer software. Except as specifically indicated otherwise, a valid license from Micro Focus is required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notice

© Copyright 2003 - 2022 Micro Focus or one of its affiliates

Trademark Notices

All trademarks, service marks, product names, and logos included in this document 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 on May 20, 2022. 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 Micro Focus Fortify 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 Micro Focus Product Documentation website:

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

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

https://community.microfocus.com/cyberres/fortify/w/fortify-product-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
22.1.0	Updated: • "Java Command-Line Options" on page 51 - New option added to display any unresolved types, fields, and functions referenced in translated Java source files • "Displaying Analysis Results Information from an FPR File" on page 140 - New option to display the lines of code counts for the analyzed files • "Sample Projects" on page 183 - Added two new samples: a Java 14 project and a Java web application
21.2.0	Added: • "Adding Trusted Certificates" on page 41 - Information about connecting Fortify Static Code Analyzer and its tools to other Fortify products that require communication over HTTPS • "Regular Expression Analysis" on page 48 - New ability to analyze files using regular expression rules • "Licensing" on page 19 and "LIM License Directives" on page 130 - New ability to use the Fortify License and Infrastructure Manager (LIM) to manage concurrent licenses Updated: • "Installing Fortify Static Code Analyzer and Applications" on page 28 - Added instructions for using a Fortify License and Infrastructure Manager (LIM) server, and added installers for AIX and Solaris operating systems • "Including Imported Modules and Packages" on page 74 - Fortify Static Code Analyzer can now import modules and packages from an automatically calculated common root directory • "Translating Scala Code" on page 108 - Removed the requirement of a

Software Release / Document Version	Changes
	Lightbend Enterprise Suite license purchase to analyze Scala code • "Translation Options" on page 119 - MSBuild and xcodebuild integrations now support the `-exclude` option • "Generating a BIRT Report" on page 147 - Added support for PCI SSF Compliance report version 1.1 • "Preparing to use Fortify Scan Wizard" on page 180 - New ability to have Fortify ScanCentral SAST perform the complete analysis remotely • "Advanced Samples" on page 185 - Removed three samples: webgoat, riches,java, and riches.net • "fortify-sca.properties" on page 189 - Added descriptions of new properties for using a license managed by the Fortify License and Infrastructure Manager (LIM) and for regular expression analysis
21.1.0	Added: • "Translating Kotlin Scripts" on page 60 - New support for translating Kotlin scripts • "Troubleshooting BIRTReportGenerator" on page 150 - Information added for resolving out of memory issues Updated: • "Translating Java Bytecode" on page 55 - New option to decompile Java bytecode for inclusion in the translation phase • "Translating Visual Studio and MSBuild Projects" on page 61 - Updates made for MSBuild integration • "Go Command-Line Options" on page 81 - Option added to specify Go proxy URLs • "Translating COBOL Code" on page 86 - Updated for Micro Focus Visual COBOL support • "Translating Dockerfiles" on page 108 - Updated the description of files that are analyzed as Dockerfiles • "Analysis Options" on page 121 and "Configuration Options" on page 187 - Removed options and properties related to FindBugs • "Working with FPR Files from the Command Line" on page 137 - Enhancements made to FPRUtility

Software Release / Document Version	Changes
	• "Generating a BIRT Report" on page 147 - New report versions available • "Configuring Scan Speed with Speed Dial" on page 158 - Added new precision levels 3 and 4 Removed: • Using FindBugs - No longer supported
20.2.0	Added: • "About Installing Fortify Static Code Analyzer and Applications" on page 27 and "Using Docker to Install and Run Fortify Static Code Analyzer" on page 33 • "Translating Dockerfiles" on page 108 • "Configuring Scan Speed with Speed Dial" on page 158 • "Fortify Java Annotations" on page 217 - Incorporated information previously available in the javaAnnotations sample `README.txt` to this guide Updated: • "Translating Visual Studio and MSBuild Projects" on page 61 - Updated to reflect the translation improvements made over the past couple releases (former chapter title: Translating .NET Code) • "Translating COBOL Code" on page 86 - Describes the changes introduced for analyzing COBOL code • "Running the Fortify ABAP Extractor" on page 98 - New option to export SAP standard code in addition to custom code • "Generating a BIRT Report" on page 147 - New supported report template added: OWASP ASVS 4.0 • "Sample Projects" on page 183 - Added two new samples • All references to Fortify ScanCentral were replaced with Fortify ScanCentral SAST (product name change)

Chapter 1: Introduction

This guide provides instructions for using Micro Focus 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. The Fortify Static Code Analyzer language technology provides rich data that enables the analyzers to pinpoint and prioritize violations so that fixes are fast and accurate. 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 *Micro Focus 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 reports

4. Auditing the results of the scan, either by opening the results (typically an FPR file) in Micro Focus Fortify Audit Workbench or uploading them to Micro Focus 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 *Micro Focus Fortify Audit Workbench User Guide* or the *Micro Focus Fortify Software Security Center User Guide* respectively.

About the Analyzers

Fortify Static Code Analyzer comprises seven vulnerability analyzers: Buffer, Configuration, Content, Control Flow, Dataflow, 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 lists and describes each analyzer.

Analyzer	Description
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 could 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.
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 48).
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.
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 uninitialized variables, and checks whether utilities, such as XML readers, are configured properly before being used.

Analyzer	Description
Dataflow	The Dataflow Analyzer detects potential vulnerabilities that involve tainted data (user-controlled input) put to potentially dangerous use. The Dataflow Analyzer uses global, interprocedural taint propagation analysis to detect the flow of data between a source (site of user input) and a sink (dangerous function call or operation). For example, the Dataflow Analyzer detects whether a user-controlled input string of unbounded length is copied into a statically sized buffer, and detects whether a user-controlled string is used to construct SQL query text.
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.
Semantic	The Semantic Analyzer detects potentially dangerous uses of functions and APIs at the intra-procedural level. Its specialized logic searches for buffer overflow, format string, and execution path issues, but is not limited to these categories. For example, the Semantic Analyzer detects deprecated functions in Java and unsafe functions in `C/C++`, such as `gets()`.
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 assignment to member variables in Java servlets, identifies the use of loggers that are not declared static final, and flags instances of dead code that is never executed because of a predicate that is always false.

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 43). For details on how to obtain a license for Fortify Static Code Analyzer, see the *Micro Focus Fortify Software System Requirements* document.

You must have a Fortify license file (`fortify.license`) and optionally you can use the Micro Focus Fortify License and Infrastructure Manager (LIM) 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 *Micro Focus 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 130.

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. Micro Focus Fortify Software Security Content 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 25). Alternatively, you can download or import previously downloaded Fortify Security Content with the fortifyupdate utility as a post-installation task (see "Manually Installing Fortify Security Content" on page 33).

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

- 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)

Fortify provides the ability to write custom rules that add to the functionality of Fortify Static Code Analyzer and the 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 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 *Micro Focus Fortify Static Code Analyzer Custom Rules Guide*.

Fortify recommends that you periodically update the security content. You can use the fortifyupdate utility to obtain the latest security content. For more information, see "Updating Security Content" on page 135.

Fortify ScanCentral SAST

You can use Micro Focus Fortify ScanCentral SAST to manage your resources by offloading the Fortify Static Code Analyzer scan 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 Micro Focus Fortify Software Security Center can direct Fortify ScanCentral SAST to output the FPR file directly to the server.

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 makes it easy to expand the system by adding more resources as needed, without having to interrupt the build process. In addition,

- 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 *Micro Focus Fortify Software System Requirements* document.

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

Fortify Scan Wizard

Micro Focus Fortify Scan Wizard is a utility that enables you to easily generate a script to perform Fortify Static Code Analyzer commands for Windows or Linux/macOS systems. You can run this generated script to analyze your code with Fortify Static Code Analyzer. You can specify to run your analysis locally, or use Micro Focus Fortify ScanCentral SAST to run all or part of the analysis remotely. For more information, see "Fortify Scan Wizard" on page 180.

Related Documents

This topic describes documents that provide information about Micro Focus Fortify software products.

> **Note:** You can find the Micro Focus Fortify Product Documentation at https://www.microfocus.com/support/documentation. Most guides are available in both PDF and HTML formats. Product help is available within the Fortify LIM product.

All Products

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

Document / File Name	Description
About Micro Focus Fortify Product Software Documentation About_Fortify_Docs_*<version>*.pdf	This paper provides information about how to access Micro Focus Fortify product documentation. **Note:** This document is included only with the product download.
Micro Focus Fortify License and Infrastructure Manager Installation and Usage Guide 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.
Micro Focus Fortify Software System Requirements Fortify_Sys_Reqs_*<version>*.pdf	This document provides the details about the environments and products supported for this version of Fortify Software.
Micro Focus Fortify Software Release Notes 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 Micro Focus Fortify Software <version> Fortify_Whats_New_*<version>*.pdf	This document describes the new features in Fortify Software products.

Micro Focus Fortify ScanCentral SAST

The following document provides information about Fortify ScanCentral SAST. Unless otherwise noted, these documents are available on the Micro Focus Product Documentation website at https://www.microfocus.com/documentation/fortify-software-security-center.

Document / File Name	Description
Micro Focus Fortify ScanCentral SAST	This document provides information about how to install,

Document / File Name	Description
Installation, Configuration, and Usage Guide SC_SAST_Guide_*<version>*.pdf	configure, and use Fortify ScanCentral SAST to streamline the static code analysis process. It is written for anyone 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.

Micro Focus Fortify Software Security Center

The following document provides information about Fortify Software Security Center. Unless otherwise noted, these documents are available on the Micro Focus Product Documentation website at https://www.microfocus.com/documentation/fortify-software-security-center.

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

Micro Focus Fortify Static Code Analyzer

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

Document / File Name	Description
Micro Focus Fortify Static Code Analyzer User Guide 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
Micro Focus Fortify Static Code Analyzer Custom Rules Guide 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. **Note:** This document is included only with the product download.
Micro Focus Fortify Audit Workbench User Guide 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.
Micro Focus Fortify Plugins for Eclipse User Guide Eclipse_Plugins_Guide_*<version>*.pdf	This document provides information about how to install and use the Fortify Complete and the Fortify Remediation Plugins for Eclipse.
Micro Focus Fortify Plugins for JetBrains IDEs and Android Studio User Guide JetBrains_AndStud_Plugins_Guide_*<version>*.pdf	This document describes how to install and use both the Fortify Analysis Plugin for IntelliJ IDEA and Android Studio and the Fortify Remediation Plugin for IntelliJ IDEA, Android Studio, and other JetBrains IDEs.
Micro Focus Fortify Extension for Visual Studio User Guide 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.
Micro Focus Fortify Static Code Analyzer Tools Properties Reference Guide SCA_Tools_Props_Ref_*<version>*.pdf	This document describes the properties used by Fortify Static Code Analyzer tools.

Chapter 2: Installing Fortify Static Code Analyzer

This chapter describes how to install and uninstall Fortify Static Code Analyzer and Fortify Static Code Analyzer tools. This chapter also describes basic post-installation tasks. See the *Micro Focus 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:

Fortify Static Code Analyzer Tools

The installation consists of Fortify Static Code Analyzer, which analyzes your build code according to a set of rules specifically tailored to provide the information necessary for the type of analysis performed. A Fortify Static Code Analyzer installation might also include one or more applications.

The following table describes the tools that are available for installation with the Fortify Static Code Analyzer and Applications installer.

Tool	Description
Micro Focus Fortify Audit Workbench	Provides a graphical user interface for Fortify Static Code Analyzer that helps you organize, investigate, and prioritize analysis results so that developers can fix security flaws quickly.
Micro Focus 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.

Tool	Description
Micro Focus Fortify Analysis Plugin for IntelliJ and Android Studio	Adds the ability to run Fortify Static Code Analyzer scans on the entire codebase of a project and apply software security rules that identify the vulnerabilities in your code from the IntelliJ and Android Studio IDEs.
Micro Focus 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 Micro Focus Fortify Software Security Center server.
Micro Focus Fortify Custom Rules Editor	A tool to create and edit custom rules.
Micro Focus Fortify Scan Wizard	A tool to quickly prepare a script that you can use to scan your code with Fortify Static Code Analyzer and optionally, upload the results directly to Fortify Software Security Center. **Note:** This tool is installed automatically with Fortify Static Code Analyzer.
Command-line utilities	There are several command-line utilities that are installed automatically with Fortify Static Code Analyzer. For more information, see "Command-Line Utilities" on page 133.

The following table describes other tools included in the Fortify Static Code Analyzer and Applications package. You install these tools separately from the Fortify Static Code Analyzer and Applications installer.

Tool	Description
Micro Focus Fortify Security Assistant Plugin for Eclipse	Provides alerts to potential security issues as you write your Java code. It provides detailed information about security risks and recommendations for how to secure the potential issue.

About Installing Fortify Static Code Analyzer and Applications

This section describes how to install Fortify SCA and Applications. You must provide a Fortify license file and optionally LIM license pool credentials for the Fortify Static Code Analyzer installation. The following table lists the different methods of installing Fortify SCA and Applications.

Installation Method	Instructions
Perform the installation using a standard install wizard	"Installing Fortify Static Code Analyzer and Applications" on the next page
Perform the installation silently (unattended)	"Installing Fortify Static Code Analyzer and Applications Silently (Unattended)" on page 30
Perform a text-based installation on non-Windows systems	"Installing Fortify Static Code Analyzer and Applications 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 SCA and Applications as a user that has a home directory with write permission. Do not install Fortify SCA and Applications as a non-root user that has no home directory.

After you complete the installation, see "Post-Installation Tasks" on page 39 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 and its components by updating the installed configuration files. For information about the configuration options for Fortify Static Code Analyzer, see "Configuration Options" on page 187. For information about configuration options for Fortify Static Code Analyzer component applications, see the *Micro Focus Fortify Static Code Analyzer Tools Properties Reference Guide*.

Installing Fortify Static Code Analyzer and Applications

To install Fortify Static Code Analyzer and Applications:

1. Run the installer file for your operating system:

   - Windows: `Fortify_SCA_and_Apps_<version>_windows_x64.exe`

   - Linux: `Fortify_SCA_and_Apps_<version>_linux_x64.run`

   - macOS: `Fortify_SCA_and_Apps_<version>_osx_x64.app.zip`

   - AIX: `Fortify_SCA_<version>_aix_x64.run`

   - Solaris: `Fortify_SCA_<version>_solaris_x86.run` or `Fortify_SCA_<version>_solaris10_sparc.run`

   where *<version>* is the software release version.

2. Accept the license agreement, and then click **Next**.

3. Choose where to install Fortify Static Code Analyzer and applications, and then click **Next**.

   > **Note:** If you are using Micro Focus Fortify ScanCentral SAST, you must specify a location that does not include spaces in the path.

4. (Optional) Select the components to install, and then click **Next**.

   > **Note:** Component selection is not available for all operating systems.

5. Specify the path to the `fortify.license` file, and then click **Next**.

6. (Optional) On the **LIM license** page, select **Yes** to use the Fortify License and Infrastructure Manager (LIM) for managing your concurrent licenses, and then click **Next**.

   > **Note:** When Fortify Static Code Analyzer performs a task that requires a license, Fortify Static Code Analyzer 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 Properties" on page 196).

   a. Type the LIM server 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**.

7. Specify the settings required to update your security content.

To update the security content for your installation:

> **Note:** For installations on non-Windows platforms and for deployment environments that do not have access to the Internet during installation, you can update the security content using the fortifyupdate utility. See "Manually Installing Fortify Security Content" on page 33.

    a. Type the URL of the update server. To use the Fortify Rulepack update server for security content updates, keep the URL as: `https://update.fortify.com`.

    b. (Optional) If connection to the update server requires a proxy server, type the proxy host and port number.

    c. Click **Next**.

8. Specify if you want to migrate from a previous installation of Fortify Static Code Analyzer on your system.

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

   > **Note:** You can also migrate Fortify Static Code Analyzer artifacts using the `scapostinstall` command-line utility. For information on how to use the post-install tool to migrate from a previous Fortify Static Code Analyzer installation, see "Migrating Properties Files" on page 39.

   To migrate artifacts from a previous installation:

       a. In the **SCA Migration** page, select **Yes**, and then click **Next**.

       b. Specify the location of the existing Fortify Static Code Analyzer installation on your system, and then click **Next**.

9. If you are installing the Fortify extension for Visual Studio, you are prompted to specify whether to install the extensions for the current install user or for all users.

   The default is to install the extensions for the current install user.

10. Specify if you want to install sample source code projects, and then click **Next**.

    See "Sample Projects" on page 183 for descriptions of these samples.

    > **Note:** If you do not install the samples and decide later that you want to install them, you must uninstall, and then re-install Fortify Static Code Analyzer and Applications.

11. Click **Next** to proceed to install Fortify Static Code Analyzer and applications.

12. After Fortify Static Code Analyzer is installed, select **Update security content after installation** if you want to update the security content, and then click **Finish**.

    The Security Content Update Result window displays the security content update results.

Installing Fortify Static Code Analyzer and Applications Silently (Unattended)

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. When you install Fortify Static Code Analyzer and Applications silently, the installer does not download the Micro Focus Fortify Software Security Content. For instructions on how to install the Fortify security content, see "Manually Installing Fortify Security Content" on page 33.

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. If you are using a different location for the Fortify Security Content updates than the default of `https://update.fortify.com`, add the following line:

      ```
      UpdateServer=<update_server_url>
      ```

      Note: As previously mentioned, Fortify security content is not downloaded with a silent installation. However, this information and the proxy information in the following step is added to the *<sca_install_dir>*`/Core/config/server.properties` file to use when you manually install Fortify security content.

   d. If you require a proxy server, add the following lines:

      ```
      UpdateProxyServer=<proxy_server>
      UpdateProxyPort=<port_number>
      ```

   e. If you do not want to install the sample source code projects, add the following line.

      ```
      InstallSamples=0
      ```

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.

For the `enable-components` option on Windows, you can specify the `AWB_group` parameter to install Fortify Audit Workbench, Fortify Custom Rules Editor, and associate FPR files with Fortify Audit Workbench. To install specific plugins, list each plugin by parameter name (the `Plugins_group` parameter does **not** install all plugins and you do not need to include it).

The following example Windows options file specifies the location of the license file, the location and proxy information for obtaining the Fortify Security Content, a request to migrate from a previous release, installation of Audit Workbench, installation of Micro Focus Fortify Extension for Visual Studio 2019 for all users, and the location of the Fortify SCA and Applications installation directory:

```
fortify_license_path=C:\Users\admin\Desktop\fortify.license
UpdateServer=https://internalserver.abc.com
UpdateProxyServer=webproxy.abc.company.com
UpdateProxyPort=8080
MigrateSCA=1
enable-components=AWB_group,VS2019
VS_all_users=1
installdir=C:\Fortify
```

The following options file example is for Linux and macOS:

```
fortify_license_path=/opt/Fortify/fortify.license
UpdateServer=https://internalserver.abc.com
UpdateProxyServer=webproxy.abc.company.com
UpdateProxyPort=8080
MigrateSCA=1
installdir=/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_and_Apps_<version>_windows_x64.exe --mode unattended --optionfile <full_path_to_options_file>`
Linux	`./Fortify_SCA_and_Apps_<version>_linux_x64.run --mode unattended --optionfile <full_path_to_options_file>`
macOS	You must uncompress the ZIP file before you run the command. `Fortify_SCA_and_Apps_<version>_osx_x64.app/Contents/MacOS/installbuilder.sh --mode unattended --optionfile <full_path_to_options_file>`
AIX and Solaris	`./Fortify_SCA_<version>_<platform>.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 depending on your operating system.

Windows	`C:\Users\<username>\AppData\Local\Temp\FortifySCAandApps-<version>-install.log`
Linux	`/tmp/FortifySCAandApps-<version>-install.log`
macOS	`/tmp/FortifySCAandApps-<version>-install.log`
AIX and Solaris	`/tmp/FortifySCA-<version>-install.log`

Installing Fortify Static Code Analyzer and Applications 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.

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

Linux	`./Fortify_SCA_and_Apps_<version>_linux_x64.run --mode text`
macOS	You must uncompress the provided ZIP file before you run the command. `Fortify_SCA_and_Apps_<version>_osx_x64.app/Contents/MacOS/installbuilder.sh --mode text`
AIX and Solaris	`./Fortify_SCA_<version>_<platform>.run --mode text`

Manually Installing Fortify Security Content

You can install Micro Focus Fortify Software Security Content (Secure Coding Rulepacks and metadata) automatically during the Windows installation procedure. However, you can also download Fortify security content from the Fortify Rulepack update server, and then use the fortifyupdate utility to install it. This option is provided for installations on non-Windows platforms and for deployment environments that do not have access to the Internet during installation.

Use the fortifyupdate utility 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 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 utility to update your security content. For more information about the fortifyupdate utility, see "Updating Security Content" on page 135.

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 112.

2. Copy the Fortify SCA and Applications 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 and Applications Silently (Unattended)" on page 30.

3. Run the Fortify SCA and Applications installer using the RUN instruction.

   You must run the installer in unattended mode. For more information, see "Installing Fortify Static Code Analyzer and Applications Silently (Unattended)" on page 30.

4. Run fortifyupdate to download the Fortify Security Content using the RUN instruction.

   For more information about this utility, see "Manually Installing Fortify Security Content" on the previous page.

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_and_Apps_<version>/bin/sourceanalyzer`.

The following is an example of a Dockerfile to install Fortify SCA and Applications:

```
FROM registry.suse.com/suse/sles12sp4
COPY fortify.license ./
COPY Fortify_SCA_and_Apps_22.1.0_linux_x64.run ./
COPY installerSettings ./
RUN zypper -n install rpm-build
RUN ./Fortify_SCA_and_Apps_22.1.0_linux_x64.run --mode unattended \
  --optionfile ./installerSettings && \
  /opt/Fortify/Fortify_SCA_and_Apps_22.1.0/bin/fortifyupdate && \
  rm Fortify_SCA_and_Apps_22.1.0_linux_x64.run fortify.license installerSettings

ENTRYPOINT [ "/opt/Fortify/Fortify_SCA_and_Apps_22.1.0/bin/sourceanalyzer" ]
```

Note: The rpm-build package in SUSE is needed by the installer.

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

```
docker build -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 SCA build session between the translate and scan phases and to share the output files (logs, FPR) 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`.

> **Important!** Include the Fortify Static Code Analyzer `–fcontainer` option in both the translate and scan commands so that Fortify Static Code Analyzer detects and uses only the memory dedicated to the container. Otherwise, by default Fortify Static Code Analyzer detects the total system memory because `-autoheap` is enabled.

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 -fcontainer
[<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 -
fcontainer [<sca_options>] –f /scratch_docker/results.fpr
```

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

About Upgrading Fortify Static Code Analyzer and Applications

To upgrade Fortify Static Code Analyzer and Applications, 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, Fortify 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 and Applications" on the next page.

> **Note:** You can leave the previous version installed. If you have multiple versions installed on the same system, the most recently installed version is invoked when you run the command from the command line. Scanning source code from the Fortify Secure Code Plugins also uses the most recently installed version of Fortify Static Code Analyzer.

Notes About Upgrading the Fortify Extension for Visual Studio

If you have administrative privileges and are upgrading from a previous version of the Fortify Static Code Analyzer for any supported version of Visual Studio, the installer will overwrite the existing Micro Focus Fortify Extension for Visual Studio. If the previous version was installed without administrative privileges, the installer will also overwrite the existing Fortify Extension for Visual Studio without requiring administrative privileges.

> **Note:** If you do not have administrative privileges and you are upgrading the Fortify Extension for Visual Studio that was previously installed using an administrative privileged user account, you must first uninstall the Fortify Extension for Visual Studio from Visual Studio using an administrative privilege account.

About Uninstalling Fortify Static Code Analyzer and Applications

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

Uninstalling Fortify Static Code Analyzer and Applications

Uninstalling on Windows Platforms

To uninstall the Fortify Static Code Analyzer and applications software:

1. Select **Start > Control Panel > Add or Remove Programs**.
2. From the list of programs, select **Fortify SCA and Applications <*version*>**, and then click **Remove**.
3. You are prompted to indicate whether to remove all application settings. Do one of the following:
   - Click **Yes** to remove the application setting folders for the tools 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 on Other Platforms

To uninstall Fortify Static Code Analyzer software non-Windows platforms:

1. Back up your configuration, including any important files you created.
2. Run the uninstall command located in the <*sca_install_dir*> for your operating system:

Linux	Uninstall_FortifySCAandApps_<*version*>
macOS	Uninstall_FortifySCAandApps_<*version*>.app
AIX and Solaris	./Uninstall_FortifySCA_<*version*>

3. You are prompted to indicate whether to remove all application settings. Do one of the following:
   - Click **Yes** to remove the application setting folders for the tools 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 and Applications 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_FortifySCAandApps_<version>.exe --mode unattended`
Linux	`./Uninstall_FortifySCAandApps_<version> --mode unattended`

macOS	`Uninstall_FortifySCAandApps_`
	`<version>.app/Contents/MacOS/installbuilder.sh`
	`--mode unattended`

AIX and Solaris	`./Uninstall_FortifySCA_<version> --mode unattended`

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

Uninstalling Fortify Static Code Analyzer and Applications 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	`./Uninstall_FortifySCAandApps_<version> --mode text`
macOS	`Uninstall_FortifySCAandApps_` `<version>.app/Contents/MacOS/installbuilder.sh --mode text`
AIX and Solaris	`./Uninstall_FortifySCA_<version> --mode text`

Post-Installation Tasks

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

Running the Post-Install Tool

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 a 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 `SCA 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 for Security Content Updates

Specify how you want to obtain Micro Focus Fortify Software 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 Micro Focus 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 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 Security Content update server or Micro Focus Fortify Software Security Center and it is no longer required, you can remove these settings.

To remove the proxy settings for Fortify Security Content updates or 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 - (hyphen) 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 and its tools to other Fortify 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 Properties" on page 196.

• The fortifyupdate utility uses an HTTPS connection either automatically during a Windows system installation or manually (see "Manually Installing Fortify Security Content" on page 33) to update

Fortify security content.

- The Fortify Static Code Analyzer Tools (such as Fortify Audit Workbench, Fortify Scan Wizard, and the Fortify Extension for Visual Studio) typically require an HTTPS connection to communicate with Fortify Software Security Center. By default, these tools do not trust self- or locally-signed certificates.
- Fortify Static Code Analyzer configured as a ScanCentral SAST sensor uses an HTTPS connection to communicate with the Controller.

When using HTTPS, Fortify Static Code Analyzer and its tools 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 the Fortify Static Code Analyzer Tools need to trust connections where the server presents a certificate issued by this CA, you must configure the Fortify Static Code Analyzer Tools 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 containing 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 into a Build" on page 112.

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 results file (typically in the Fortify Project Results (FPR) format). FPR files have the `.fpr` file 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 49.

The following is an example of the sequence of commands you use to translate and analyze code:

```
sourceanalyzer -b <build_id> -clean
sourceanalyzer -b <build_id> ...
sourceanalyzer -b <build_id> -scan -f MyResults.fpr
```

The three commands in the previous example illustrate the following steps in the analysis process:

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

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

2. Translate the project code.

   This step can consist of multiple calls to sourceanalyzer with the same build ID (except for dynamic languages including JavaScript/TypeScript, PHP, Python, and Ruby).

3. Analyze the project code and produce the Fortify Project Results file (FPR).

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. The chapters for each source code type describe any specific requirements.

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.

> **Caution!** 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.

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
```

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 and MSBuild Projects" on page 61

- "Translating C and C++ Code" on page 67
- "Translating JavaScript and TypeScript Code" on page 70
- "Translating Python Code" on page 74
- "Translating Code for Mobile Platforms" on page 78
- "Translating Go Code" on page 81
- "Translating Ruby Code" on page 84
- "Translating COBOL Code" on page 86
- "Translating Apex and Visualforce Code" on page 91
- "Translating Other Languages and Configurations" on page 94

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.

> **Note:** Fortify recommends that you do not use MBS for regular expression analysis (see "Regular Expression Analysis" on page 48) unless both systems have the same version of the source code (with the files stored on the same paths on each system). If the same source code does not exist on both systems, the quality of the regular expression analysis results is unpredictable. If the source code files do not exist on the scan machine, then vulnerabilities normally detected by regex analysis would not be reported.

You must have the same version of Fortify Security Content (Rulepacks) installed on both the system where you are performing the translation and the system where you are performing 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, 22.1.0.0240). 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, 21.2.0 and 21.2.x are compatible.

To determine the Fortify Static Code Analyzer version number, type `sourceanalyzer -v` on the command line.

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.

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.

Note: If necessary, you can obtain the build ID and Fortify Static Code Analyzer version from an MBS file with the following command:

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

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

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 and).

An example of the basic command-line syntax for the analysis phase is:

```
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
```

The use of antivirus software can negatively impact Fortify Static Code Analyzer performance. If you notice long scan times, Fortify 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 Fortify 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.

Higher-Order Analysis

Higher-Order Analysis (HOA) improves the ability to track dataflow through higher-order code. Higher-order code manipulates functions as values, generating them with anonymous function expressions (lambda expressions), passing them as arguments, returning them as values, and assigning 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 189.

Modular Analysis

This release includes a technology preview of modular analysis. With modular analysis, you can pre-scan libraries (and sublibraries) separately from your core project. You can then include these pre-scanned libraries when you scan the core project. Doing this might improve the core project analysis performance because you are not rescanning the libraries every time you scan the core project. Modular analysis also enables you to scan a project that references a library without requiring the library's source code, Fortify Static Code Analyzer translated files, or custom rules used to scan the library. This has the added benefit that you only need to audit issues in your core application. The analysis results are more streamlined to code that you directly control and therefore you do not need to worry about issues in code that you do not own.

Modular analysis is currently available for libraries and applications developed in Java and Java EE.

Note: In this release, you might not see any performance improvements from modular analysis. Fortify is working to optimize the performance of modular analysis in future releases.

You must rescan your libraries whenever you:

- Update to a new version of Fortify Static Code Analyzer
- Update your Fortify security content
- Change the libraries

Modular Command-Line Examples

To translate and scan a library separately, type:

```
sourceanalyzer -b LibA MyLibs/A/*.java

sourceanalyzer -b LibA -scan-module
```

To translate and scan the core project and include multiple pre-scanned libraries:

```
sourceanalyzer -b MyProj MyProj/*.java

sourceanalyzer -b MyProj -scan -include-modules LibA,LibB
```

For a description of the options shown in the previous examples, see "Analysis Options" on page 121.

Regular Expression Analysis

Regular expression (regex) analysis provides the ability to use regular expression rules for detecting 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.

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's 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 119.
- 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
```

Translation and Analysis Phase Verification

Micro Focus Fortify Audit Workbench result 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 *Micro Focus 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 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 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 integrating Fortify Static Code Analyzer with Ant, see "Ant Integration" on page 114.

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
  ```

  Fortify 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 Java EE).

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

Java/Java EE Option	Description
	contains the `JspBatchCompiler` script. **Equivalent Property Name:** `com.fortify.sca.AppServerHome`
`-appserver-version` `<version>`	Specifies the version of the application server. **Equivalent Property Name:** `com.fortify.sca.AppServerVersion`

`-cp <dirs>`|
`-classpath <dirs>`
	Specifies the class path to use for analyzing 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:
	
	`-cp "build/classes:lib/*.jar"`
	
	For information about file specifiers, see "Specifying Files and Directories" on page 131.
	
	Equivalent Property Name:
	`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. **Equivalent Property Name:** `com.fortify.sca.JavaExtdirs`
`-java-build-dir <dirs>`	Specifies one or more directories that contain compiled Java sources.
`-source <version>`| `-jdk <version>`	Indicates the JDK version for which the Java code is written. See the *Micro Focus Fortify Software System Requirements* document for supported versions. The default is Java 1.8. **Equivalent Property Name:** `com.fortify.sca.JdkVersion`
`-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

Java/Java EE Option	Description
	each class, field, and function with the source information of the first translated occurrence in the code. This information is also written to the log file.
	Equivalent Property Name: `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.
	Equivalent Property Name: `com.fortify.sca.JavaSourcePath`

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 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
```

Java Warnings

You might see the following warnings for Java:

```
Unable to resolve type...
```

```
Unable to resolve function...
```

```
Unable to resolve field...
```

```
Unable to locate import...
```

```
Unable to resolve symbol...
```

```
Multiple definitions found for function...
```

```
Multiple definitions found for class...
```

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. To resolve the warnings, make sure that you include all the required files that your application uses.

Translating Java EE Applications

To translate Java EE applications, Fortify Static Code Analyzer processes Java source files and Java EE components such as JSP files, deployment descriptors, and configuration files. While you can process all the pertinent files in a Java 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 Java EE applications, use the same procedure used to translate Java files. For examples, see "Java Command-Line Examples" on the previous page.

Translating JSP Projects, Configuration Files, and Deployment Descriptors

In addition to translating the Java files in your 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.

Java EE Translation Warnings

You might see the following warning in the translation of Java 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 containing 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

Fortify 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.

In addition to translating source code, you can translate the Java bytecode in your project. To translate bytecode, you have the following two options:

- Request that Fortify Static Code Analyzer decompile the bytecode classes to regular Java files for inclusion in the translation.

  To have the bytecode decompiled for the translation, 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.DecompileBytecode=true
  ```

- Request that Fortify Static Code Analyzer translate bytecode without decompilation.

  For best results, Fortify recommends that the bytecode be compiled with full debug information (`javac -g`).

To include bytecode in the Fortify Static Code Analyzer translation:

a. 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.

b. In the Fortify Static Code Analyzer translation phase, specify 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 translating and scanning JSP.

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 scanning Java EE applications, see "Translating Java EE Applications" on page 54.

Increased Issues Counts 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
```

  Fortify 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>`|
`-classpath <dirs>`
	Specifies the class path to use for translating 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:
	
	```-cp "build/classes:lib/*.jar"```
	
	For information about file specifiers, see "Specifying Files and Directories" on page 131.
	
	Equivalent Property Name:
	`com.fortify.sca.JavaClasspath`

`-source <version>`| `-jdk <version>`	Indicates the JDK version for which the Kotlin code is written. See the *Micro Focus Fortify Software System Requirements* document for supported versions. The default is Java 1.8. **Equivalent Property Name:** `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. **Equivalent Property Name:** `com.fortify.sca.JavaSourcePath`

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 and scan a gradle project using gradlew, type:

```
sourceanalyzer -b MyProject gradlew clean assemble

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

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 the 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` file extension. Fortify Static Code Analyzer translates `*.kts` files but does not apply these templates.

Chapter 6: Translating Visual Studio and MSBuild Projects

Fortify Static Code Analyzer supports translation of the following types of projects built with Visual Studio or MSBuild:

- C/C++ projects
- .NET projects written in C# or Visual Basic (VB.NET)

  This includes projects that target .NET, .NET Framework, .NET Core, and .NET Standard
- ASP.NET applications

  This includes applications that make use of the ASP.NET Core framework
- Xamarin applications that target Android and iOS platforms

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

This section contains the following topics:

Visual Studio and MSBuild Project Translation Prerequisites

> **Important!** Fortify Static Code Analyzer supports translation of Visual Studio and MSBuild projects on Windows systems only.

Fortify recommends that each project you translate is complete and that you perform the translation in an environment where you can build it without errors. 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 and MSBuild Project Translation Command-Line Syntax

The basic syntax to translate Visual Studio or MSBuild projects is to append an MSBuild command that builds the project to the Fortify Static Code Analyzer command. The following command translates a Visual Studio solution called `Sample.sln`:

```
sourceanalyzer –b <build_id> msbuild /t:rebuild Sample.sln
```

This command builds and translates the solution or project. Fortify strongly recommends you run this command from the Developer Command Prompt for Visual Studio to ensure an optimal environment for the translation.

> **Important!** When you translate from the Developer Command Prompt for Visual Studio environment, Fortify recommends that you run the `dotnet restore` command before you run the Fortify Static Code Analyzer translation. You must run this command from the top-level folder of the project. This ensures that all required reference libraries are downloaded and installed in the project.

After the translation is complete, you can perform the analysis phase as shown in the following example:

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

Handling Special Cases for Translating Visual Studio and MSBuild Projects

Running Translation from a Script

As stated previously, Fortify recommends that you run the Fortify Static Code Analyzer translation of Visual Studio and MSBuild projects from the Developer Command Prompt for Visual Studio. 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 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 Micro Focus 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 do the following in the MSBuild command:

- Enclose the setting value in quotes (in addition to the quotes around the appropriate command-line option)
- Quotes are escaped

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
```

Working with Multiple Targets and Projects for MSBuild Command

Recent versions of MSBuild enable you to build multiple targets and specific projects using an extended syntax of the `/t` option. However, Fortify recommends that you avoid this syntax for translating Visual Studio or MSBuild projects. Specifying a single `rebuild` target is sufficient to translate any supported Visual Studio and MSBuild project.

If you cannot use a single `rebuild` target to translate your project, Fortify provides limited support of multiple targets and projects specified for the MSBuild command. To use this feature, add the `-multiple-msbuild-targets` option to the Fortify Static Code Analyzer command, as shown in the following example:

```
sourceanalyzer –b MySampleProj -multiple-msbuild-targets msbuild
/t:Project1:build;Project2:build Sample.sln
```

Note: Support of this translation mode is limited, and Fortify Static Code Analyzer might not properly handle all possible combinations of targets and projects.

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`), Fortify 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 `-binary-name` option when running 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.

```
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 -f Sample2.fpr
```

For more information about the `-binary-name` option, see "Analysis Options" on page 121.

Alternative Ways to Translate Visual Studio and MSBuild Projects

This section describes alternative methods of translating Visual Studio and MSBuild projects.

Alternative Translation Options for Visual Studio Solutions

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

- Use the Micro Focus 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 a Visual Studio solution called `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 or MSBuild project without invoking Fortify Static Code Analyzer directly. This requires the `Fortify.targets` file, which is located in the *<sca_install_dir>*`\Core\private-bin\sca\MSBuildPlugin` directory. You can specify the file using an absolute or relative path in the MSBuild command line that builds your project. For example:

```
msbuild /t:rebuild
/p:CustomAfterMicrosoftCommonTargets=<sca_install_dir>\Core\private-
bin\sca\MSBuildPlugin\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. This is equivalent to the Fortify Static Code Analyzer `-b` option.	None
FORTIFY_ MSBUILD_	Enables debug mode. This is equivalent to the Fortify Static Code Analyzer `–debug`	False

Environment Variable	Description	Default Value
DEBUG	option.	
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 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. 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 translating Visual Studio or MSBuild projects, see "Translating Visual Studio and MSBuild Projects" on page 61.

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'
}
```

For more information about integrating with Gradle, see "Build Integration" on page 112.

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 *Micro Focus 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 invoke 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. This generated code is optimized., minimized, or both. Therefore, you might want to 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 manually exclude this type of code.

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

> **Note:** 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.

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 the 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 119.

Excluding NPM Dependencies

By default, Fortify Static Code Analyzer translates only the NPM dependences that are imported in the code. You can change this behavior with the following two properties:

- The `com.fortify.sca.follow.imports` property directs Fortify Static Code Analyzer to resolve all imported files and include them in the translation.

  This property is enabled by default. Setting this property to false prevents NPM dependencies that are not explicitly included on the command-line from being included in the translation.

- The `com.fortify.sca.exclude.unimported.node.modules` property directs Fortify Static Code Analyzer to exclude all files in any node_modules directory from the translation except files that are specifically imported by the `com.fortify.sca.follow.imports` property.

  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.

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 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 them 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.

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.

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 Fortify Static Code Analyzer is translating. 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 in the `-python-path` option. For example, if you have two subpackages for a namespace package `package_name` in multiple folders as in this example:

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

Include the following with the `-python-path` option: `/path_1;/path_2`.

Using the Django Framework with Python

Fortify Static Code Analyzer supports the Django framework. To translate code created using the Django 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 templates in the project root folder. Any Django templates found are automatically added to the translation. If you do not want Fortify Static Code Analyzer to automatically discover Django templates, use the `-django-disable-autodiscover` option. If your project requires Django templates, but the project is configured such that Django templates are in an unexpected location, use the `-django-template-dirs` option to specify the directories that contain the templates in addition to the `-django-disable-autodiscover` option.

You can specify additional locations of Django template files by adding the `-django-template-dirs` option to the sourceanalyzer command:

```
-django-template-dirs <dirs>
```

Python Command-Line Options

The following table describes the Python options.

Python Option	Description
`-python-version <version>`	Specifies the Python source code version you want to scan. The valid values for `<version>` are 2 and 3. The default value is 2. **Equivalent Property Name:** `com.fortify.sca.PythonVersion`
`-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. **Equivalent Property Name:** `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. **Equivalent Property Name:** `com.fortify.sca.PythonPath`
`-django-disable-autodiscover`	Specifies that Fortify Static Code Analyzer does not automatically discover Django templates. **Equivalent Property Name:** `com.fortify.sca.DjangoDisableAutodiscover`

Python Option	Description
`-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. **Equivalent Property Name:** `com.fortify.sca.DjangoTemplateDirs`

Python Command-Line Examples

To translate Python 3 code, type:

```
sourceanalyzer -b Python3Proj -python-version 3 -python-path
/usr/lib/python3.4:/usr/local/lib/python3.4/site-packages src/*.py
```

To translate Python 2 code, type:

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

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 and MSBuild Projects" on page 61.

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.

> **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 119). 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
```

You can then perform the analysis phase, 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 Micro Focus Fortify Analysis Plugin for IntelliJ and Android Studio. For more information, see the *Micro Focus Fortify Plugins for JetBrains IDEs 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 "Gradle Integration" on page 115.

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.

Fortify 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 177. You can filter out the issues based on the Instance ID.

Chapter 11: Translating Go Code

This section describes how to translate Go code.

This section contains the following topics:

Go Command-Line Syntax

For 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
`-gopath <dir>`	Specifies the root directory of your project. Make sure that the directory structure adheres to the Go workspace hierarchy (https://golang.org/doc/gopath_code.html). If this option is not specified, then the GOPATH system environment variable is used. You must specify the gopath directory as an absolute path. The following examples are valid values for `<dir>`: `/home/projects/go_workspace/my_proj`

Go Option	Description

	`C:\projects\go_workspace\my_proj`
	
	The following example is an invalid value for *<dir>*:
	
	`go_workspace/my_proj`
	
	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.
	
	Equivalent Property Name
	`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. 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. **Equivalent Property Name** `com.fortify.sca.GOROOT`
`-goproxy <url>`	Specifies one or more comma-separated proxy URLs. You can also specify `direct` or `off` (to disable network usage). 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`. **Equivalent Property Name** `com.fortify.sca.GOPROXY`

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 81.

  - 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 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) **Equivalent Property Name:** `com.fortify.sca.RubyLibraryPaths`

Ruby Option	Description
`-rubygem-path <dirs>`	Specifies the path(s) to a RubyGems location (see "Adding Gem Paths" below) **Equivalent Property Name:** `com.fortify.sca.RubyGemPaths`

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 13: Translating COBOL Code

In the previous release, Fortify Static Code Analyzer introduced updated COBOL code translation, which is now the default translation method. The previous translation method, referred to now as legacy COBOL translation is still available for use with a command-line option. Use the legacy COBOL translation method if either of the following is true:

- You run Fortify Static Code Analyzer on a non-Windows operating system
- Your COBOL dialect is unsupported.

The following sections describe the default COBOL code translation. Information that pertains only to the legacy COBOL translation is indicated as such.

For a list of supported technologies for translating COBOL code, see the *Micro Focus 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 Micro Focus Fortify Customer Support for more information about scanning COBOL and the required license file.

This section contains the following topics:

Preparing COBOL Source and Copybook Files for Translation

Fortify Static Code Analyzer supports translation of COBOL source files on Windows systems only.

> **Legacy COBOL Translation:** Fortify Static Code Analyzer supports translation of COBOL source files on the supported platforms and architectures listed in the *Micro Focus Fortify Software System Requirements* document.

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

  Fortify recommends that your COBOL source code files have extensions `.CBL`, `.cbl`, `.COB` or `.cob`. Therefore, there is no need to manage extension specification from the command line as described in "Translating COBOL Source Files Without File Extensions" on the next page and "Specifying Files and Directories" on page 131.

  > **Note:** Fortify Static Code Analyzer translates COBOL source code files with or without file extensions.

- All copybook files that the COBOL source code uses
- 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 file extension `.CPY` or `.cpy`.

> **Legacy COBOL Translation:** Fortify Static Code Analyzer translates copybook files with or without file extensions.

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`.

> **Legacy COBOL Translation:**
>
> - The corresponding copybook file has the name `FOO` with or without a file extension. If the copybook files have file extensions, use the `-copy-extensions` command-line option. For more information, see "Legacy COBOL Translation Command-Line Options" on page 89.
> - The `COPY` command can also accept a directory-file-path structure instead of a file name.

Fortify 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 is:

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

Legacy COBOL Translation: Free-format COBOL is the default translation mode. Fortify Static Code Analyzer supports the translation of fixed-format COBOL. To translate fixed-format COBOL, you must specify the `-fixed-format` command-line option. For more information, see "Legacy COBOL Translation Command-Line Options" on the next page.

See Also

"Specifying Files and Directories" on page 131

Translating COBOL Source Files Without File Extensions

If you have COBOL source 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 131).

COBOL Command-Line Options

The following table describes the COBOL command-line options.

COBOL Option	Description
`-copydirs <dirs>`	Specifies one or more semicolon-separated directories where Fortify Static Code Analyzer looks for copybook files. **Equivalent Property Name:** `com.fortify.sca.CobolCopyDirs`
`-cobol-dialect <dialect>`	Specifies the COBOL dialect. The valid values for dialect are `COBOL390` or `MICROFOCUS`. The dialect value is case insensitive. The default value is `COBOL390`. **Equivalent Property Name:** com.fortify.sca.CobolDialect
`-checker-directives <directives>`	Specifies one or more semicolon-separated COBOL checker directives. **Note:** This option is intended for advanced users of Micro Focus Server Express. **Equivalent property name:** `com.fortify.sca.CobolCheckerDirectives`

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. **Equivalent Property Name:** `com.fortify.sca.CobolLegacy`
`-copydirs <dirs>`	Specifies one or more semicolon- or colon-separated directories where Fortify Static Code Analyzer looks for copybook files. **Equivalent Property Name:**

Legacy COBOL Option	Description
	`com.fortify.sca.CobolCopyDirs`
`-copy-extensions <ext>`	Specifies one or more semicolon- or colon-separated copybook file extensions. **Equivalent Property Name:** `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. IBM Enterprise COBOL code is typically fixed-format. The following are indications that you might need the `-fixed-format` option: • The COBOL translation appears to hang indefinitely • Fortify Static Code Analyzer reports numerous parsing errors in the COBOL translation **Equivalent Property Name:** `com.fortify.sca.CobolFixedFormat`

Chapter 14: Translating Apex and Visualforce Code

This section contains the following topics:

Apex Translation Prerequisites

- 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 are in files with the `.trigger` extension

  Use the Force.com Migration Tool available on the Salesforce website to download your app from your org in the Salesforce cloud to your local computer.

- If you customized the standard Salesforce database structures to support your app, then you must also download a description of the changes so that Fortify Static Code Analyzer knows how your modified version of Salesforce interacts with your app. See "Downloading Customized Salesforce Database Structure Information" on the next page.

Apex and Visualforce Command-Line Syntax

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

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

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

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

For descriptions of all the Apex- and Visualforce-specific command-line options, see "Apex and Visualforce Command-Line Options" below.

Apex and Visualforce Command-Line Options

The following table describes the Apex and Visualforce translation command-line options.

Apex or Visualforce Option	Description
`-apex`	Directs Fortify Static Code Analyzer to use the Apex and Visualforce translation for files with the `.cls` extension. Without this option, Fortify Static Code Analyzer translates `*.cls` files as Visual Basic code. **Note:** Alternatively, you can set the `com.fortify.sca.fileextensions.cls` property to APEX either on the command line (include `-Dcom.fortify.sca.fileextensions.cls=APEX`) or in the `<sca_install_dir>/Core/config/fortify-sca.properties` file. **Equivalent Property Name:** `com.fortify.sca.Apex`
`-apex-sobject-path <path>`	Specifies the location of the custom sObject JSON file `sobjects.json`. For instructions on how to use the sf_extractor tool, see "Downloading Customized Salesforce Database Structure Information" below. **Equivalent Property Name:** `com.fortify.sca.ApexObjectPath`

Downloading Customized Salesforce Database Structure Information

Use the sf_extractor tool to download a description of any customized Salesforce database structures. Fortify Static Code Analyzer requires this information to perform a more complete analysis. The sf_extractor creates a custom sObject JSON file that you include with the sourceanalyzer translation phase. (For information about how to provide this information to Fortify Static Code Analyzer, see "Apex and Visualforce Command-Line Options" above.)

The following table describes the contents of the `sf_extractor.zip` file, which is located in `<sca_install_dir>`/Tools.

Folder or File Name	Description
lib	Folder containing JAR dependencies
src	Source code
partner.wsdl	Partner WSDL file version 37.0
sf_extractor.jar	Compiled JAR file (dependencies included)

The command-line syntax to run sf_extractor is:

```
java -jar sf_extractor.jar <username> <password> <security_token> <org>
```

where:

- `<username>` is your Salesforce cloud user name. For example, `test@test.test`.
- `<password>` is your Salesforce cloud password.
- `<security_token>` is the 25 alphanumeric character security token.
- `<org>` is `y` if you are using a sandbox org or `n` if you are using a production org.

The sf_extractor tool uses the credentials to access the Salesforce SOAP API. It downloads all the sObjects with additional information from the current org, and then it downloads information about fields in the sObjects. This is required to properly resolve types represented in current org.

This tool produces an `sobjects.json` file that you provide to Fortify Static Code Analyzer in the translation command using the `-apex-sobject-path` option.

Chapter 15: Translating Other Languages and Configurations

This section contains the following topics:

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. **Equivalent Property Name:** `com.fortify.sca.PHPSourceRoot`
`-php-version` `<version>`	Specifies the PHP version. The default version is 7.4. For a list of valid versions, see the *Micro Focus Fortify Software System Requirements* document. **Equivalent Property Name:** `com.fortify.sca.PHPVersion`

Translating ABAP Code

Translating ABAP code is similar to translating other operating language code. However, it requires additional 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, invokes 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 does read 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. This set of files is what Fortify Static Code Analyzer translates and scans, producing an FPR file that you can open with Micro Focus Fortify Audit Workbench.

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 Fortify 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 are running 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 is running on an SAP version with a different table structure, Fortify 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 launch 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.
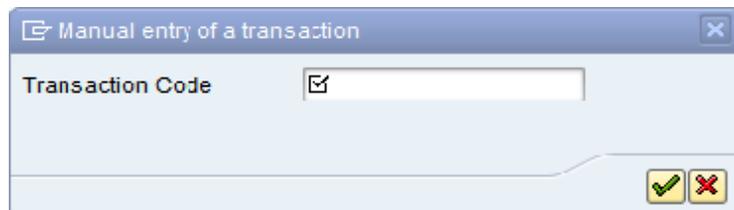
3. Type YSCA in the **Transaction Code** box.

4. Click the green check mark button ✅.
   The **Extract ABAP code and launch SCA** item appears in the **Favorites** list.
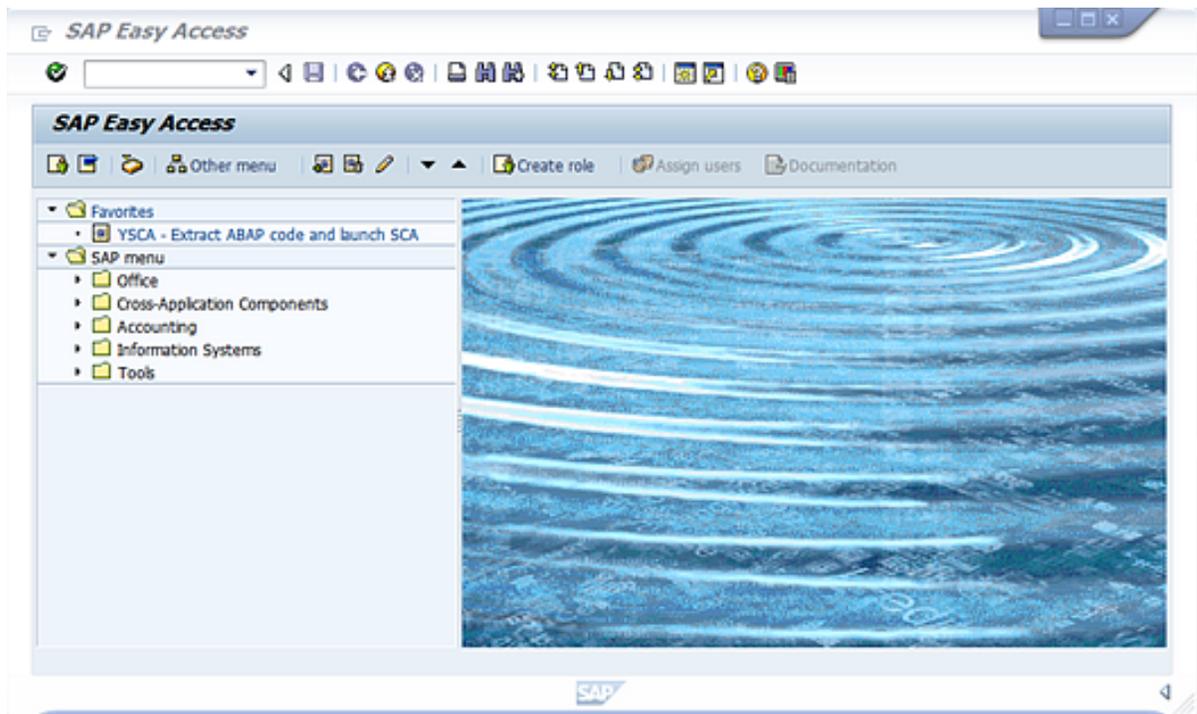
5. Click the **Extract ABAP code and launch SCA** link to launch the Fortify ABAP Extractor.

Running the Fortify ABAP Extractor

To run the Fortify ABAP Extractor:

1. Start the program from the **Favorites** link, the transaction code, or manually start the Extractor object.
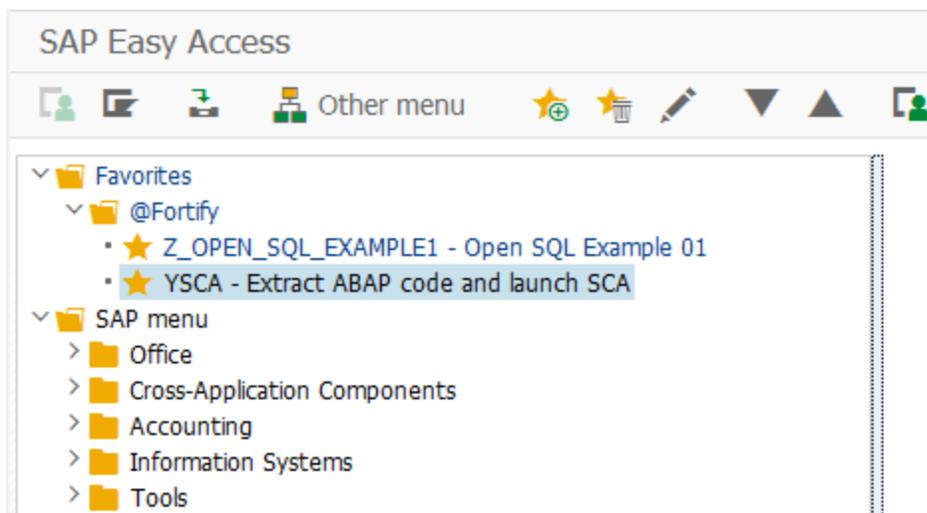
This opens the Fortify ABAP Extractor.
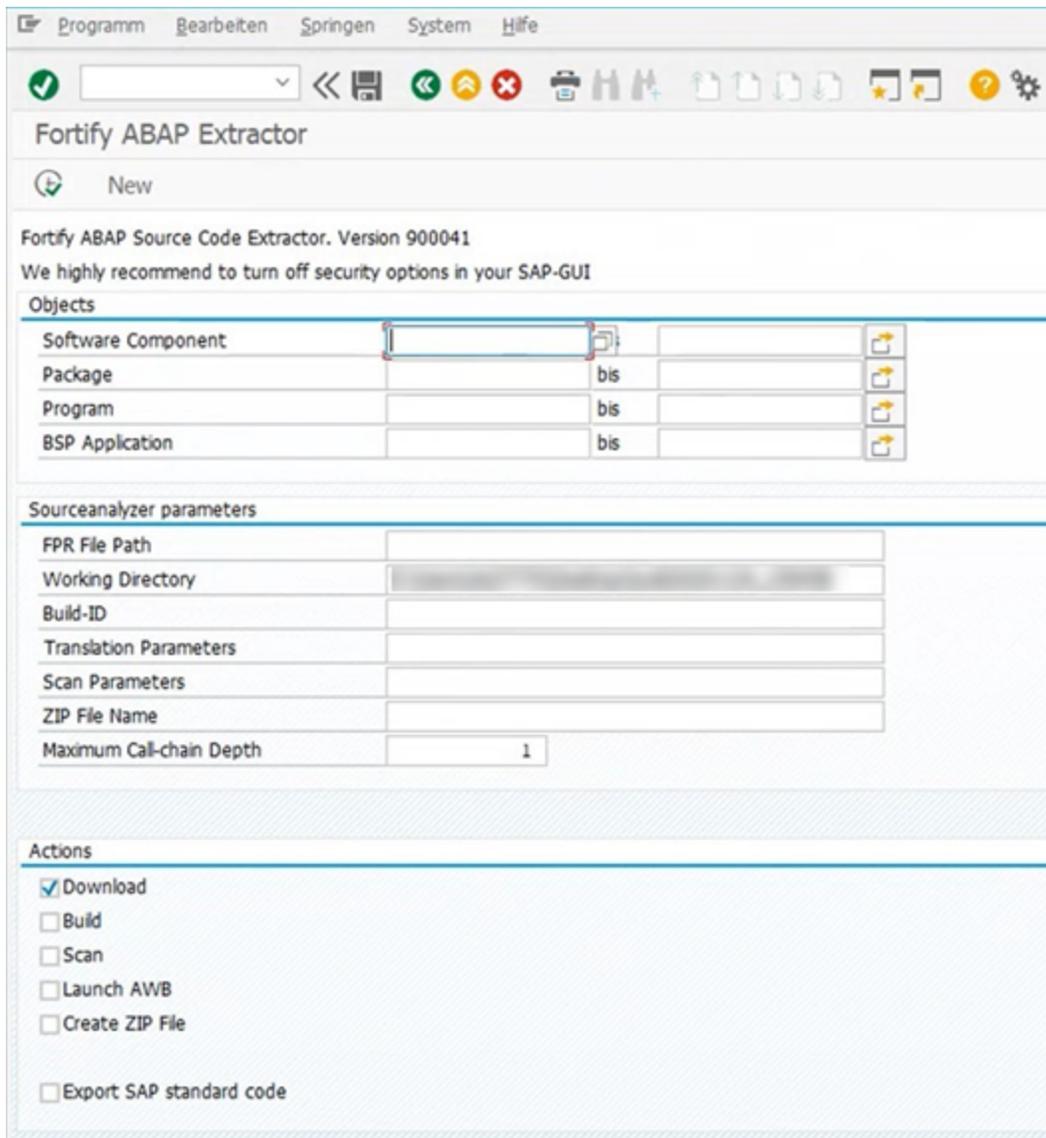
2. Select the code to download.

   Provide the start and end name for the range of software components, packages, programs, or BSP applications that you want to scan.
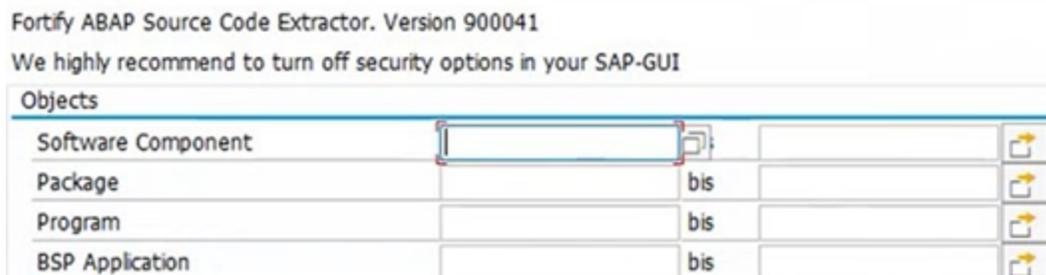
   Note: You can specify multiple objects or ranges.
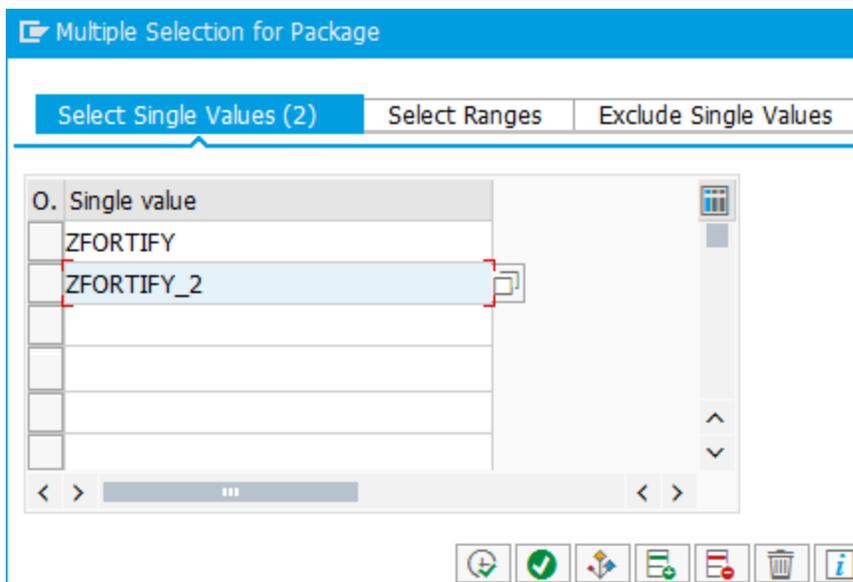
3. Provide the Fortify Static Code Analyzer-specific information described in the following table.

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 if you want 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 if you want to automatically translate

Field	Description
	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 parameters if you want to automatically translate the downloaded code on the same machine where you are running the extraction process or you want to customize the translation options.
Scan Parameters	(Optional) Type any Fortify Static Code Analyzer command-line scan options. You must specify scan parameters if you want to scan the downloaded code automatically on the same machine where you are running the process or you want to customize the scan options.
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. Fortify recommends that you do not specify a value greater than 2 unless directed to do so by Micro Focus Fortify Customer Support.

4. Provide action information described in the following table.

Field	Description
Download	Select this check box to have Fortify Static Code Analyzer download the

Field	Description
	source code extracted from your SAP database.
Build	Select this 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 predefined Fortify Static Code Analyzer machine.
Scan	Select this 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 this check box to start Micro Focus Fortify Audit Workbench and open the specified FPR file.
Create ZIP File	Select this 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 this check box to export SAP standard code in addition to custom code.

5. Click **Execute**.

Fortify ABAP Extractor
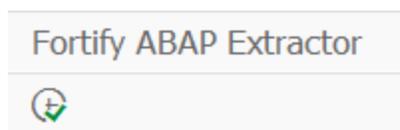
Uninstalling the Fortify ABAP Extractor

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 for translating ActionScript is:

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

where:

`<libs>` is a semicolon-separated (Windows) or a colon-separated (non-Windows systems) 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. **Equivalent Property Name:** `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). **Note:** You can specify SWC or SWF files as Flex libraries (SWZ is not currently supported). **Equivalent Property Name:** `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. Normally, these contain a subfolder named `com`.

Flex and ActionScript Option	Description
	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`). **Equivalent Property Name:** `com.fortify.sca.FlexSourceRoots`

Note: `-flex-sdk-root` and `–flex-source-roots` 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 could fail, translation to ActionScript could fail, and the parsing of the resulting ActionScript could also fail. If you see any errors that do not have a clear connection to the original source code, notify Micro Focus Fortify Customer Support.

ActionScript Command-Line Examples

The following examples illustrate command-line syntax for typical scenarios for translating 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 translating 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 Fortify Static Code Analyzer 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 a properties file because typing in the data is time consuming and the data is generally constant. 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

Type the following to translate ColdFusion source code:

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

where:

- *<build_id>* specifies the 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 131.

> **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. **Equivalent Property Name:** `com.fortify.sca.SourceBaseDir`

Translating SQL

On Windows platforms, 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.

To specify the SQL type for translation on Windows platforms, type one of the following translation commands:

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

or

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

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 `TSQL` or `PLSQL`.

PL/SQL Command-Line Example

The following example shows the syntax to translate two PL/SQL files:

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

The following example shows how to translate all PL/SQL files in the `sources` directory:

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

T-SQL Command-Line Example

The following example shows the command to translate two T-SQL files:

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

The following example shows how to translate 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`.

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 the Lightbend documentation at https://developer.lightbend.com/guides/fortify.

> **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 Dockerfiles

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

> **Note:** You can modify the file extension used to detect Dockerfiles using the
> `com.fortify.sca.fileextensions` property. See "Translation and Analysis Phase Properties" on page 189.

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 indicating 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 16: Integrating 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 in 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 *Micro Focus 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 integrating with Xcodebuild, see "iOS Code Analysis Command-Line Syntax" on page 79. For information about integration with MSBuild, see "Translating Visual Studio and MSBuild Projects" on page 61.

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 Invoke Fortify Static Code Analyzer" 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 Invoke Fortify Static Code Analyzer

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 Fortify Static Code Analyzer options.

```
CC=sourceanalyzer -b mybuild gcc
CXX=sourceanalyzer -b mybuild g++
LD=sourceanalyzer -b mybuild ld
AR=sourceanalyzer -b mybuild 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.

Touchless Build Integration

Fortify Static Code Analyzer includes a generic build tool called `touchless` that enables translation of projects using build systems that Fortify Static Code Analyzer does not directly support. The command-line syntax for touchless build integration is:

```
sourceanalyzer -b <build_id> touchless <build_command>
```

For example, you might use a python script called `build.py` to compute dependencies and execute appropriately-ordered C compiler operations. Then to execute your build, run the following command:

```
python build.py
```

Fortify Static Code Analyzer does not have native support for such a build design. However, you can use the touchless build tool to translate and build the entire project with the single command:

```
sourceanalyzer -b <build_id> touchless python build.py
```

The same requirements for successful build integration with supported build systems described earlier in this chapter (see) apply to touchless integration with unsupported build systems.

Ant Integration

Fortify Static Code Analyzer provides an easy way to 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 on the system `PATH`.

Prepend your Ant command-line with the `sourceanalyzer` command as follows:

```
sourceanalyzer -b <build_id> ant [<ant_options>]
```

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. See the *Micro Focus 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 on the system PATH. Prepend the Gradle command line with the `sourceanalyzer` command as follows:

```
sourceanalyzer -b <build_id> <sca_options> gradle [<gradle_options>]
<gradle_tasks>
```

For example:

```
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>]
```

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 translating the project with gradle or gradlew, you can then perform the analysis phase as shown in the following example:

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

Including Verbose and Debug Options

If you use the Fortify Static Code Analyzer `-verbose` option, then you must also include the `-gradle` option. Use of this option applies to both Gradle and the Gradle Wrapper. For example:

```
sourceanalyzer -b MyProject -gradle -verbose gradle assemble
```

As part of the gradle integration, Fortify Static Code Analyzer temporarily updates the original build file `build.gradle`. If you include the `-debug` option, Fortify Static Code Analyzer saves a copy of the original build file as `build.gradle.orig`). After the analysis with the `-debug` option is complete, rename the `build.gradle.orig` file back to `build.gradle` and run sourceanalyzer again without the `-debug` option.

Maven Integration

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 Micro Focus Fortify ScanCentral SAST
- Upload results to Micro Focus 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 *Micro Focus Fortify Software System Requirements* document.

If you have a previous version of the Fortify Maven Plugin installed, and 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 a Fortify analysis on a maven project:

- As a Maven Plugin

  In this method, you perform the Fortify 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:<ver>: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>]
  ```

  For example:

  ```
  sourceanalyzer -b MyProject mvn package
  ```

  See "Command-Line Interface" on page 119 for descriptions of available Fortify Static Code Analyzer options.

  c. Complete the analysis by running the scan as shown in the following example:

  ```
  sourceanalyzer -b MyProject [<sca_scan_options>] -scan -f
  MyResults.fpr
  ```

Chapter 17: 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. **Equivalent Property Name:** `com.fortify.sca.BuildID`
`-disable-language`	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`, `dotnet`, `golang`,`java`, `javascript`, `jsp`, `kotlin`, `objc`, `php`, `plsql`, `python`, `ruby`, `scala`, `sql`, `swift`, `tsql`, `typescript`, and `vb`. **Equivalent Property Name:** `com.fortify.sca.DISabledLanguages`

Translation Option	Description
`-enable-language`	Specifies a colon-separated list of languages to translate. The valid language values are `abap`, `actionscript`, `apex`, `cfml`, `cobol`, `configuration`, `cpp`, `dotnet`, `golang`, `java`, `javascript`, `jsp`, `kotlin`, `objc`, `php`, `plsql`, `python`, `ruby`, `scala`, `sql`, `swift`, `tsql`, `typescript`, and `vb`. **Equivalent Property Name:** `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). For example: ```sourceanalyzer -cp "**/*.jar" "**/*" -exclude "**/Test/*.java"``` This example excludes all Java files in any `Test` subdirectory. See "Specifying Files and Directories" on page 131 for more information on how to use file specifiers. **Note:** When you integrate the translation with most compilers or build tools, Fortify Static Code Analyzer translates all source files that the compiler or build tool processes even if this option specifies to exclude them. However, the Fortify Static Code Analyzer xcodebuild and MSBuild integrations do support the `-exclude` option. **Equivalent Property Name:** `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. Valid encoding names are from the `java.nio.charset.Charset`. Typically, if you do not specify the encoding type, Fortify Static Code Analyzer uses `file.encoding` from the

Translation Option	Description
	`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. **Equivalent Property Name:** `com.fortify.sca.InputFileEncoding`
`-nc`	When specified before a compiler command line, Fortify Static Code 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 file extension. The valid file type values are ABAP, ACTIONSCRIPT, APEX, APEX_TRIGGER, ARCHIVE, ASPNET, ASP, ASPX, BITCODE, BSP, BYTECODE, CFML, COBOL, CSHARP, DOCKERFILE, 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, and XML.

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. **Equivalent Property Name:** `com.fortify.sca.BuildID`
`-scan`	Causes Fortify Static Code Analyzer to perform analysis for the specified build ID. **Note:** Do not use this option together with the `-scan-module` option in the same sourceanalyzer command.
`-scan-module`	Causes Fortify Static Code Analyzer to perform analysis for the specified build ID as a separate module. **Note:** Do not use this option together with the `-scan` option in the

Analysis Option	Description
	same sourceanalyzer command.
	Equivalent Property Name:
	`com.fortify.sca.ScanScaModule`
`-include-modules`	Specifies the libraries previously scanned as separate modules in a comma- or colon-separated list of build IDs to include in the project scan.
	Equivalent Property Name:
	`com.fortify.sca.IncludeScaModules`
`-analyzers`	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.
	Equivalent Property Name:
	`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 158.
	Equivalent Property Name:
	`com.fortify.sca.PrecisionLevel`
`-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 157.
	Equivalent Property Name:
	`com.fortify.sca.QuickScanMode`
`-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.
	Equivalent Property Name:

Analysis Option	Description
	`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. 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`. The `<type>` parameter is case-insensitive.
`-filter <file>`	Specifies a results filter file. For more information, see "Filtering the Analysis" on page 177. **Equivalent Property Name:** `com.fortify.sca.FilterFile`
`-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. **Note:** This is equivalent to disabling the following rule types: DataflowSink, Semantic, Controlflow, Structural, Configuration, Content, Statistical, Internal, and Characterization:Issue. **Equivalent Property Name:** `com.fortify.sca.NoDefaultIssueRules`
`-no-default-rules`	Specifies not to load rules from the default Rulepacks. Fortify Static Code Analyzer processes the Rulepacks for description elements and language libraries, but processes no rules. **Equivalent Property Name:** `com.fortify.sca.NoDefaultRules`
`-no-default-source-rules`	Disables source rules in the default Rulepacks. **Note:** Characterization source rules are not disabled. **Equivalent Property Name:** `com.fortify.sca.NoDefaultSourceRules`

Analysis Option	Description
`-no-default-sink-rules`	Disables sink rules in the default Rulepacks. **Note:** Characterization sink rules are not disabled. **Equivalent Property Name:** `com.fortify.sca.NoDefaultSinkRules`
`-project-template`	Specifies the issue template file to use for the scan. This only affects scans on the local machine. If you upload the FPR to Micro Focus Fortify Software Security Center, it uses the issue template assigned to the application version. **Equivalent Property Name:** `com.fortify.sca.ProjectTemplate`
`-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. **Equivalent Property Name:** `com.fortify.sca.RulesFile`

Output Options

The following table describes the output options.

Output Option	Description

`-f <file>|`
`-output-file`
`<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 the terminal.
	
	 Equivalent Property Name:
	 `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 extension of the file provided with the `-f` option. The FVDL is an XML file that contains the detailed Fortify Static Code Analyzer analysis results. This includes vulnerability details, rule

Output Option	Description
	descriptions, code snippets, command-line options used in the scan, and any scan errors or warnings.
	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). Micro Focus Fortify Audit Workbench is automatically associated with the `.fpr` file extension.
	Note: If you use result certification, you must specify the `fpr` format. See the *Micro Focus Fortify Audit Workbench User Guide* for information about result certification.
	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 161.
	Equivalent Property Name:
	`com.fortify.sca.Renderer`
-append	Appends results to the file specified with the `-f` option. The resulting FPR 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.
	The engine data, which includes Fortify 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 the `-append` option, Fortify does not certify results generated with `-append`.
	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.
	In general, only use the `-append` option when it is impossible to analyze an entire application at once.
	Equivalent Property Name:
	`com.fortify.sca.OutputAppend`

Output Option	Description
`-build-label` `<label>`	Specifies the label of the project being scanned. Fortify Static Code Analyzer does not use this label but includes it in the analysis results. **Equivalent Property Name:** `com.fortify.sca.BuildLabel`
`-build-project` `<project>`	Specifies the name of the project being scanned. Fortify Static Code Analyzer does not use the name but includes it in the analysis results. **Equivalent Property Name:** `com.fortify.sca.BuildProject`
`-build-version` `<version>`	Specifies the version of the project being scanned. Fortify Static Code Analyzer does not use the version but includes it in the analysis results. **Equivalent Property Name:** `com.fortify.sca.BuildVersion`
`-disable-source-bundling`	Excludes source files from the FPR file. **Equivalent Property Name:** `com.fortify.sca.FPRDisableSourceBundling`
`-fvdl-no-descriptions`	Excludes the Fortify security content descriptions from the analysis results file. **Equivalent Property Name:** `com.fortify.sca.FVDLDisableDescriptions`
`-fvdl-no-enginedata`	Excludes engine data from the analysis results file. The engine data includes Fortify security content information, command-line options, system properties, warnings, errors, and other information about the Fortify Static Code Analyzer execution. **Equivalent Property Name:** `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. **Equivalent Property Name:** `com.fortify.sca.FVDLDisableProgramData`
`-fvdl-no-snippets`	Excludes the code snippets from the analysis results file.

Output Option	Description
	Equivalent Property Name: `com.fortify.sca.FVDLDisableSnippets`

Other Options

The following table describes other options.

Other Option	Description
`@<file>`	Reads command-line options from the specified file. **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 189.

`-h |`
`-? |`
`-help`
	Prints a summary of command-line options.

`-debug`	Includes debug information in the Fortify Support log file, which is only useful for Micro Focus Fortify Customer Support to help troubleshoot. **Equivalent Property Name:** `com.fortify.sca.Debug`
`-debug-verbose`	This is the same as the `-debug` option, but it includes more details, specifically for parse errors. **Equivalent Property Name:** `com.fortify.sca.DebugVerbose`
`-verbose`	Sends verbose status messages to the console and to the Fortify Support log file. **Equivalent Property Name:** `com.fortify.sca.Verbose`
`-logfile <file>`	Specifies the log file that Fortify Static Code Analyzer creates.

Other Option	Description
	Equivalent Property Name: `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. **Equivalent Property Name:** `com.fortify.sca.ClobberLogFile`
`-quiet`	Disables the command-line progress information. **Equivalent Property Name:** `com.fortify.sca.Quiet`
`-version|` `-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.
`-Xmx<size>M|G`	Specifies the maximum amount of memory Fortify Static Code Analyzer uses. 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 probably 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. 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. **Note:** Specifying this option overrides the default memory allocation you would get 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. **Note:** This option can generate an extensive amount of information.
`-show-build-warnings`	Use with `-b` *`<build_id>`* to show any errors and warnings that occurred in the translation phase on the console. **Note:** Fortify Audit Workbench also displays these errors and warnings in the results Certification tab.
`-show-files`	Lists the files in the specified build ID. When the `-bin` option is present, displays only the source files that went into the binary.
`-show-loc`	Displays the number of lines in the code being translated.

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 41.

Use the directives described in the following table for a license managed by the 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.
	
	Example:
	
	```sourceanalyzer -store-license-pool-credentials "https://<ip_address>/LIM.REST.API|TeamA|mypassword"```
	
	Associated Property Names:
	`com.fortify.sca.lim.Url`
	`com.fortify.sca.lim.PoolName`
	`com.fortify.sca.lim.PoolPassword`
	`com.fortify.sca.lim.ProxyUrl`
	`com.fortify.sca.lim.ProxyUsername`
	`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.

Directive	Description
	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>

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> *<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 *<dir>*/*.jar	Matches any file with the specified extension found in the named directory.
<dir>/**/*.kt *<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.

File / Directory Specifier	Description
`**/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. Matches: • `/src/main/java/org/owasp/webgoat` • `/test/java/org/owasp/webgoat` Does not match (`assignments` directory does not match) • `/test/java/org/owasp/webgoat/assignments`

Note: 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 (/).

Chapter 18: Command-Line Utilities

This section contains the following topics:

Fortify Static Code Analyzer Utilities

Fortify Static Code Analyzer command-line utilities enable you to manage Fortify Security Content and FPR files, run reports, perform post-installation configurations, and monitor scans. These utilities are located in `<sca_install_dir>/bin`. The utilities for Windows are provided as `.bat` or `.cmd` files. The following table describes the utilities.

> **Note:** By default, log files for most Fortify Static Code Analyzer utilities are written to the following directory:
>
> - Windows: `C:\Users\<username>\AppData\Local\Fortify\<utility_name>-<version>\log`
> - Non-Windows: `<userhome>/.fortify/<utility_name>-<version>/log`

Utility	Description	More Information
fortifyupdate	Compares installed security content to the current version and makes any required updates	"About Updating Security Content" on the next page
FPRUtility	With this utility you can: - Merge audited projects - Verify FPR signatures - Display information from an FPR file including: - Any errors associated with the analysis	"Working with FPR Files from the Command Line" on page 137

Utility	Description	More Information
	- Number of issues - Filtered lists of issues in different formats - Lines of code for analyzed files - List of analyzed functions - Mappings for a migrated project - Combine or split source code files and audit projects into FPR files - Alter an FPR	
BIRTReportGenerator ReportGenerator	Generates BIRT reports and legacy reports from FPR files	"Generating Reports from the Command Line" on page 146
scapostinstall	After you install Fortify Static Code Analyzer, this utility 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 39
SCAState	Provides state analysis information on the JVM during the scan phase	"Checking the Fortify Static Code Analyzer Scan Status" on page 151

About Updating Security Content

You can use the `fortifyupdate` utility to download the latest Fortify Secure Coding Rulepacks and metadata from Fortify.

The `fortifyupdate` utility gathers information about the existing security content in your Fortify 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 Security Content

Use the `fortifyupdate` utility to either download security content or import a local copy of the security content. This utility is located in the `<sca_install_dir>/bin` directory.

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 [<options>]
```

The default read timeout for this utility is 180 seconds. To change the timeout setting, add the `rulepackupdate.SocketReadTimeoutSeconds` property in the `server.properties` configuration file. For more information, see the *Micro Focus Fortify Static Code Analyzer Tools Properties Reference Guide*.

fortifyupdate Command-Line Options

The following table lists the `fortifyupdate` options.

fortifyUpdate Option	Description
`-import <file>.zip`	Imports the ZIP file that contains security content. Rulepacks are extracted to 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 makes the update in the *<sca_install_dir>*. **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: • en (English)

fortifyUpdate Option	Description
	• es (Spanish) • ja (Japanese) • ko (Korean) • pt_BR (Brazilian Portuguese) • zh_CN (Simplified Chinese) • zh_TW (Traditional Chinese) **Note:** The values are ***not*** case-sensitive. Alternatively, you can specify a default locale for security content updates in the `fortify.properties` configuration file. For more information, see the *Micro Focus Fortify Static Code Analyzer Tools Properties Reference 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 or 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. For more information about the `server.properties` configuration file, see the *Micro Focus Fortify Static Code Analyzer Tools Properties Reference Guide*.

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 are updating from a non-standard location (that you specify with the `-url` option).
`-acceptSSLCertificate`	Specifies to use the SSL certificate provided by the server.

Working with FPR Files from the Command Line

Use the FPRUtility located in the `bin` directory of your Fortify Static Code Analyzer installation to perform the following tasks:

- "Merging FPR Files" on the next page
- "Displaying Analysis Results Information from an FPR File" on page 140
- "Extracting a Source Archive from an FPR File" on page 144
- "Altering FPR Files" on page 146
- "Allocating More Memory for FPRUtility" on page 146

Merging FPR Files

The FPRUtility `-merge` option combines the analysis results from two FPR files into a single FPR file. The values of the primary project are used to resolve conflicts. When you merge two FPR files, copies of both the primary analysis results and the secondary analysis results are stored in the merged FPR. When you open a merged FPR in Micro Focus Fortify Audit Workbench or Micro Focus Fortify Software Security Center, *removed issues* are determined as those that exist in the secondary analysis results but not in the primary analysis results. Similarly, *new issues* are those that exist in the primary analysis results, but not in the secondary analysis results.

<<If merge is primarily used to merge an new FPR with an older FPR (analysis of the same application), then can we rename "primary" as "new" and "secondary" as "old" in this topic?>>

To merge FPR files:

```
FPRUtility -merge -project <primary>.fpr -source <secondary>.fpr \
 -f <merged>.fpr
```

To merge FPR files and set instance ID migrator options:

```
FPRUtility -merge -project <primary>.fpr -source <secondary>.fpr \
 -f <merged>.fpr -iidmigratorOptions "<iidmigrator_options>"
```

FPRUtility Data Merge Options

The following table lists the FPRUtility options that apply to merging data.

FPRUtility Option	Description
-merge	Merges the specified project and source FPR files.
-project <primary>.fpr	Specifies the primary FPR file to merge. Conflicts are resolved using the values in this file.
-source <secondary>.fpr	Specifies the secondary FPR file to merge. The primary project overrides values if conflicts exist.
-f <merged>.fpr	Specifies the name of the merged FPR file to contain the result of the merged files. **Note:** When you specify this option, neither of the original FPR files are modified. If you do not use this option, the primary FPR is overwritten with the merged results.
-forceMigration	Forces the migration, even if Fortify Static Code Analyzer and

FPRUtility Option	Description
	the Rulepack versions of the two projects are the same.
-ignoreAnalysisDates	Specifies to ignore the analysis dates in the primary and secondary FPR files for the merge. Otherwise, the secondary FPR is always updated with the primary FPR .
-useSourceIssueTemplate	Specifies to use the filter sets and folders from the issue template in the secondary FPR.
-useMigrationFile <mapping_file>	Specifies an instance ID mapping file. This enables you to modify mappings manually rather than using the migration results. Supply your own instance ID mapping file.
-iidmigratorOptions <iidmigrator_options>	Specifies instance ID migrator options. Separate included options with spaces and enclosed them in quotes. Some valid options are: • -i provides a case-sensitive file name comparison of the merged files • -u <scheme_file> tells iidmigrator to read the matching scheme from <scheme_file> for instance ID migration **Note:** Wrap <-iidmigrator_options> in single quotes ('-u <scheme_file>') when working from a Cygwin command prompt. Windows example: ``` FPRUtility -merge -project <primary>.fpr -source <secondary>.fpr -f <merged>.fpr -iidmigratorOptions "-u scheme_file" ```
-debug	Displays debug information that can be helpful to troubleshoot issues with FPRUtility.

FPRUtility Data Merge Exit Codes

Upon completion of the `-merge` command, FPRUtility provides one of the exit codes described in the following table.

Exit Code	Description
0	The merge completed successfully.
5	The merge failed.

Displaying Analysis Results Information from an FPR File

The FPRUtility `-information` option displays information about the analysis results. You can obtain information to:

- Validate signatures
- Examine any errors associated with the FPR
- Obtain the number of issues for each analyzer, vulnerability category, or custom grouping
- Obtain lists of issues (including some basic information). You can filter these lists.
- Obtain the list of analyzed files and the number of lines of code (LOC) for each file. You can also compare the LOC with another FPR.

To display signature information for the analysis:

```
FPRUtility -information -signature -project <project>.fpr -f <output>.txt
```

To display a full analysis error report for the FPR:

```
FPRUtility -information -errors -project <project>.fpr -f <output>.txt
```

To display the number of issues per vulnerability category or analyzer:

```
FPRUtility -information -categoryIssueCounts -project <project>.fpr
FPRUtility -information -analyzerIssueCounts -project <project>.fpr
```

To display the number of issues for a custom grouping based on a search:

```
FPRUtility -information -search -query <search_expression> \
[-categoryIssueCounts] [-analyzerIssueCounts] \
[-includeSuppressed] [-includeRemoved] \
-project <project>.fpr -f <output>.txt
```

Note: By default, the result does not include suppressed and removed issues. To include suppressed or removed issues, use the `-includeSuppressed` or `-includeRemoved` options.

To display information for issues in CSV format:

```
FPRUtility -information -listIssues \
-search [-queryAll | -query <search_expression>] \
[-categoryIssueCounts] [-analyzerIssueCouts] \
[-includeSuppressed] [-includeRemoved] \
-project <project>.fpr -f <output>.csv -outputFormat CSV
```

To display information for all issues from the most recent scan (excluding unsuppressed and removed issues) using the Quick View filter set:

```
FPRUtility -information -listIssues \
-search -queryAllExistingUnsuppressed \
-filterSet "Quick View" \
[-categoryIssueCounts] [-analyzerIssueCouts] \
-project <project>.fpr -f <output>.txt
```

To display a comparison of the number of lines of code for analyzed files in two FPRs:

```
FPRUtility -information -loc -project <project>.fpr \
-compareTo <oldproject>.fpr -f <output>.txt
```

FPRUtility Information Options

The following table lists the FPRUtility options that apply to project information.

FPRUtility Option	Description
-information	Displays information for the project.
Specify one of the following options to indicate what information to display:	
-signature	Displays the signature for analysis results and rules.
-mappings	Displays the migration mappings report.
-errors	Displays a full error report for the FPR.
-versions	Displays the Fortify Static Code Analyzer and the Fortify Secure Coding Rulepacks versions used in the static scan.
-functionsMeta	Displays all functions that the static analyzer encountered in CSV format. To filter which functions are displayed, include -excludeCoveredByRules, and -excludeFunctionsWithSource.
-categoryIssueCounts	Displays the number of issues for each vulnerability category.
-analyzerIssueCounts	Displays the number of issues for each analyzer.

FPRUtility Option	Description
`-search <query_option>`	• Use `-search -query <search_expression>` to display the number of issues in the result of your specified search expression. To display the number of issues per vulnerability category or analyzer, add the optional `-categoryIssueCounts` and `-analyzerIssueCounts` options to the search option. Use the `-includeSuppressed` and `-includeRemoved` options to include suppressed or removed issues. • Use `-search -queryAll` to search all the issues in the FPR including suppressed and removed issues. • Use `-search -queryAllExistingUnsuppressed` to search all the issues in the FPR excluding suppressed and removed issues.

`-loc`	Displays the list of analyzed files each with the number of lines of code (LOC) in the following format:
	
	`<filename>: <total_loc> (<executable_loc>)`
	
	where *<total_loc>* is the approximate number of lines that contain code constructs (comments are excluded).
	
	Use `-compareTo <project>.fpr` with this option to compare the number of lines of code with another FPR. The comparison output includes the following information:
	
	• + indicates new analyzed files
	
	• – indicates removed analyzed files
	
	• * indicates files with a different number of lines of code. The difference in the number of lines of code is shown next to the executable LOC number as in (+N or -N). For example:
	
	`* ProjectA/main.jsp: 115 +15 (85 +7)`
	
	In the previous example, the comparison shows that the number of lines of code in `main.jsp` is different between the two FPR files. There are 15 additional total LOC and 7 additional executable LOC.

`-project <project>.fpr`	Specifies the FPR from which to extract the results information.
`-listIssues`	Displays the location for each issue in one of the following formats: `<sink_filename>:<line_num>` or `<sink_filename>:<line_num> (<category> | <analyzer>)` You can also use the `-listIssues` option with `-search` and with both issueCounts grouping options. If you group by `-categoryIssueCounts`, then the output includes (*<analyzer>*) and if you group by

FPRUtility Option	Description
	`-analyzerIssueCounts`, then the output includes (`<category>`).
	If you specify the `-outputFormat` `CSV` option, then each issue is displayed on one line in the format:
	```"<instanceid>", "<category>",``` ```"<sink_filename>:<line_num>", "<analyzer>"```
`-filterSet <filterset_name>`	Displays only the issues and counts that pass the filters specified in the filter set. Filter sets are ignored without this option. **Important!** You must use `-search` with this option.
`-f <output>`	Specifies the output file. The default is `System.out`.
`-outputformat <format>`	Specifies the output format. The valid values are `TEXT` and `CSV`. The default value is `TEXT`.
`-debug`	Displays debug information that can be helpful to troubleshoot issues with FPRUtility.

FPRUtility Signature Exit Codes

Upon completion of the `-information -signature` command, FPRUtility provides one of the exit codes described in the following table.

Exit Code	Description
0	The project is signed, and all the signatures are valid.
1	The project is signed, and some, but not all, of the signatures passed the validity test.
2	The project is signed but none of the signatures are valid.
3	The project had no signatures to validate.

Extracting a Source Archive from an FPR File

The FPRUtility -sourceArchive option creates a source archive (FSA) file from a specified FPR file and removes the source code from the FPR file. You can extract the source code from an FPR file, merge an existing source archive (FSA) back into an FPR file, or recover source files from a source archive.

To archive data:

```
FPRUtility -sourceArchive -extract -project <project>.fpr -f <output_
archive>.fsa
```

To archive data to a directory:

```
FPRUtility -sourceArchive -extract -project <project>.fpr \
-recoverSourceDirectory -f <output_dir>
```

To add an archive to an FPR file:

```
FPRUtility -sourceArchive -mergeArchive -project <project>.fpr \
-source <old_source_archive>.fsa -f <project_with_archive>.fpr
```

To recover files that are missing from an FPR file:

```
FPRUtility -sourceArchive -fixSecondaryFileSources \
-payload <source_archive>.zip -project <project>.fpr -f <output>.fpr
```

FPRUtility Source Archive Options

The following table lists the FPRUtility options that apply to working with the source archive.

FPRUtility Option	Description
-sourceArchive	Creates an FSA file so that you can extract a source archive.
One of: -extract -mergeArchive -fixSecondaryFileSources	Use the -extract option to extract the contents of the FPR file.
	Use the -mergeArchive option to merge the contents of the FPR file with an existing archived file (-source option).

FPRUtility Option	Description
	Use the `-fixSecondaryFileSources` option to recover source files from a source archive (`-payload` option) missing from an FPR file.
`-project <project>.fpr`	Specifies the FPR to archive.
`-recoverSourceDirectory`	Use with the `-extract` option to extract the source as a directory with restored source files.
`-source <old_source_archive>.fsa`	Specifies the name of the existing archive. Use only if you are merging an FPR file with an existing archive (`-mergeArchive` option).
`-payload <source_archive>.zip`	Use with the `-fixSecondaryFileSources` option to specify the source archive from which to recover source files.
`-f <project_with_archive>.fpr | <output_archive>.fsa | <output_dir>`	Specifies the output file. You can generate an FPR, a directory, or an FSA file.
`-debug`	Displays debug information that can be helpful to troubleshoot issues with FPRUtility.

Altering FPR Files

Use the FPRUtility `-trimToLastScan` option to remove the previous scan results from a merged project (FPR). This reduces the size of the FPR file when you no longer need the previous scan results. This can also reduce the time it takes to open an FPR in Fortify Audit Workbench.

To remove the previous scan from the FPR:

```
FPRUtility -trimToLastScan -project <merged_project>.fpr [-f <output>.fpr]
```

FPRUtility Alter FPR File Options

FPRUtility Option	Description
`-trimToLastScan`	Removes the previous scan results from a merged project.
`-project <merged_project>.fpr`	Specifies the merged FPR to alter. If this project is not a merged project, then the FPR file remains unchanged.
`-f <output>.fpr`	Specifies the name of the altered output file. If you do not specify this option, then the merged FPR is altered.

Allocating More Memory for FPRUtility

Performing tasks with large and complex FPR files might trigger out-of-memory errors. By default, 1000 MB is allocated for FPRUtility. To increase the memory, add the `-Xmx` option to the command line. For example, to allocate 2 GB for FPRUtility, use the following command:

```
FPRUtility -Xmx2G -merge -project <primary>.fpr -source <secondary>.fpr \
-f <output>.fpr
```

Generating Reports from the Command Line

There are two command-line utilities to generate reports:

- BIRTReportGenerator—Produces reports that are based on the Business Intelligence and Reporting Technology (BIRT) system from FPR files.

  Note: If you are using a text-based Linux system running OpenJDK, you must install DejaVu Sans and DejaVu Serif fonts to successfully generate BIRT reports.

- ReportGenerator—Generates legacy reports from FPR files. You can specify a report template, otherwise a default report template is used. See the *Micro Focus Fortify Audit Workbench User Guide* for a description of the available report templates.

Generating a BIRT Report

The basic command-line syntax to generate a BIRT report is:

```
BIRTReportGenerator -template <template_name>
-source <audited_proj>.fpr -format <format>
-output <report_file>
```

The following is an example of how to generate an OWASP Top 10 2017 report with additional options:

```
BIRTReportGenerator -template "owasp top 10" -source auditedProj.fpr
-format pdf -showSuppressed --Version "owasp top 10 2017"
--UseFortifyPriorityOrder -output MyOWASP_Top10_Rpt.pdf
```

BIRTReportGenerator Command-Line Options

The following table lists the BIRTReportGenerator options.

BIRTReportGenerator Option	Description
`-template <template_name>`	(Required) Specifies the report template name. The valid values for `<template_name>` are `"CWE Top 25"`, `"CWE/SANS Top 25"`, `"Developer Workbook"`, `"DISA CCI 2"`, `"DISA STIG"`, `"FISMA Compliance"`, `GDPR`, `MISRA`, `"OWASP ASVS 4.0"`, `"OWASP Mobile Top 10"`, `"OWASP Top 10"`, `"PCI DSS Compliance"`, and `"PCI SSF Compliance"`. **Note:** You only need to enclose the report template name in quotes if a space exists in the *<template_ name>*. The template name values are ***not*** case-sensitive.
`-source <audited_proj>.fpr`	(Required) Specifies the audited project on which to base the report.
`-format pdf|doc|html|xls`	(Required) Specifies the generated report format.

BIRTReportGenerator Option	Description
	Note: The format values are ***not*** case-sensitive.
`-output <report_file.***>`	(Required) Specifies the file to which the report is written. **Note:** If you specify a file that already exists, that file is overwritten.
`-searchQuery <query>`	Specifies a search query to filter issues before generating the report. For example: `-searchQuery audited:false` For a description of the search query syntax, see the *Micro Focus Fortify Audit Workbench User Guide*.
`-showSuppressed`	Include issues that are marked as suppressed.
`-showRemoved`	Include issues that are marked as removed.
`-showHidden`	Include issues that are marked as hidden.
`-filterSet <filterset_name>`	Specifies a filter set to use to generate the report (for example, `-filterSet "Quick View"`).
`--Version <version>`	Specifies the version for the template. The valid values for the template versions are listed below. The template version values are case-insensitive. **Note:** Templates that are not listed here have only one version available. If you do not specify a version and multiple versions are available, BIRTReportGenerator uses the most recent version based on the external metadata used when the FPR was created.

BIRTReportGenerator Option	Description
	• For the "CWE Top 25" template, the version is "`CWE Top 25 <year>`" (for example, "`CWE Top 25 2020`") • For the "CWE/SANS Top 25" template, the version is "`<year> CWE/SANS Top 25`" (for example, "`2010 CWE/SANS Top 25`") • For the "DISA STIG" template, the version is "`DISA STIG <version>`" (for example, "`DISA STIG 4.11`") • For the "FISMA Compliance" template, the version is "`NIST 800-53 Rev <version>`" (for example, "`NIST 800-53 Rev 4`") • For the MISRA template, the available versions are "`MISRA C 2012`" or "`MISRA C++ 2008`" • For the "OWASP Top 10" template, the version is "`OWASP Top 10 <year>`" (for example, "`OWASP Top 10 2017`") • For the "PCI DSS Compliance" template, the version is "`<version> Compliance`" (for example, "`3.2 Compliance`") • For the "PCI SSF Compliance" template, the version is "`PCI SSF <version>`" (for example, "`PCI SSF 1.0`")
`--IncludeDescOfKeyTerminology`	Include the *Description of Key Terminology* section in the report.
`--IncludeAboutFortify`	Include the *About Fortify Solutions* section in the report.
`--SecurityIssueDetails`	Provide detailed descriptions of reported issues. This option is not available for the Developer Workbook template.
`--UseFortifyPriorityOrder`	Use Fortify Priority Order instead of folder names to categorize issues. This option is not available for the Developer Workbook and PCI Compliance templates.

BIRTReportGenerator Option	Description
`-h|-help`	Displays detailed information about the options.
`-debug`	Displays debug information that can be helpful to troubleshoot issues with BIRTReportGenerator.

Troubleshooting BIRTReportGenerator

Occasionally, you might encounter an out of memory error when you generate a report. You might see a message similar to the following in the command-line output:

```
java.lang.OutOfMemoryError: GC overhead limit exceeded
```

To increase the memory allocated for BIRTReportGenerator:

1. Make a copy of the *<sca_install_dir>*/bin/BIRTReportGenerator file as a backup.
2. Open the *<sca_install_dir>*/bin/BIRTReportGenerator file in a text editor.

   The last line in the file includes `-Xmx1500M`, which is the total heap memory provided to BIRTReportGenerator. This line looks similar to the following:

   ```
   "%FORTIFY_CORE%\private-bin\awb\eclipse\eclipsec.exe" -vm "%JAVA_CMD%" ...
   -Xmx1500M -Dcom.fortify.log.console=%CONSOLE_LOG% %USER_VM_OPTS%
   ```

3. Change the value for the `-Xmx` option in that line to increase the amount of heap memory for BIRTReportGenerator. For example, to allocate 32 GB to BIRTReportGenerator, set the option to `-Xmx32G`.

   > **Note:** Do not allocate more memory that is physically available.

4. Save and close the file, and then rerun the report.

Generating a Legacy Report

To generate a PDF report, type the following command:

```
ReportGenerator -format pdf -f <myreport>.pdf -source <myresults>.fpr
```

To generate an XML report, type the following command:

```
ReportGenerator -format xml -f <myreport>.xml -source <myresults>.fpr
```

ReportGenerator Command-Line Options

The following table lists the ReportGenerator options.

ReportGenerator Option	Description
`-format pdf | xml`	(Required) Specifies the generated report format.
`-source <audited_proj>.fpr`	(Required) Specifies the audited project on which to base the report.
`-f <report_file.***>`	(Required) Specifies the file to which the report is written.
`-template <template_name>`	Specifies the report template. If not specified, ReportGenerator uses the default template. The default template is located in `<sca_install_dir>/Core/config/reports/DefaultReportDefinition.xml`. **Note:** Enclose the *<template_name>* in quotes if it contains any spaces.
`-user <username>`	Specifies a user name to add to the report.
`-showSuppressed`	Include issues marked as suppressed.
`-showRemoved`	Include issues marked as removed.
`-showHidden`	Include issues marked as hidden.
`-filterSet <filterset_name>`	Specifies a filter set to use to generate the report (for example, `-filterset "Quick View"`).
`-verbose`	Displays status messages to the console.
`-debug`	Displays debug information that can be helpful to troubleshoot issues with ReportGenerator.
`-h`	Displays detailed information about the options.

Checking the Fortify Static Code Analyzer Scan Status

Use the SCAState utility to see up-to-date state analysis information during the scan phase.

To check Fortify Static Code Analyzer state:

1. Run a Fortify Static Code Analyzer scan.
2. Open another command window.
3. Type the following at the command prompt:

```
SCAState [<options>]
```

SCAState Utility Command-Line Options

The following table lists the SCAState utility options.

SCAState Option	Description
-a| --all	Displays all available information.
-debug	Displays information that is useful to debug SCAState behavior.
-ftd| --full-thread-dump	Prints a thread dump for every thread.
-h|--help	Displays the help information for the SCAState utility.
-hd <*filename*>| --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 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. To obtain the process ID on Windows systems:

SCAState Option	Description
	1. Open a command window. 2. Type `tasklist` at the command prompt. A list of processes is displayed. 3. Find the `java.exe` process in the list and note its PID. To find the process ID on Linux systems: • Type `ps aux | grep sourceanalyzer` at the command prompt.
`-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.
`-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. Fortify recommends that you run SCAState as an administrator on Windows.

Chapter 19: 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:

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)

Fortify 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 utility.

Application Complexity	CPU Cores	RAM (GB)	Description
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 *Micro Focus 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. Fortify 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 169 for more information about how to determine the amount of memory required for optimal performance.
- **CPU**—Fortify recommends a 2.1 GHz or faster processor.

Sample Scans

These sample scans were performed using Fortify Static Code Analyzer version 22.1.0 on dedicated virtual machines. These scans were run using Micro Focus Fortify Software Security Content 2022 Update 1. The following table shows the scan times you can expect for several common open-source projects.

Project Name	Language	Translation Time (mm:ss)	Analysis (Scan) Time (mm:ss)	Total Issues	LOC	System Configuration
nasm 0.98.38	C/C++	00:30	03:52	1,104	21,449	Linux VM with 8 CPUs and 32 GB of RAM

Project Name	Language	Translation Time (mm:ss)	Analysis (Scan) Time (mm:ss)	Total Issues	LOC	System Configuration
WebGoat 8	Java	00:36	01:07	515	5,223	Linux VM with 8 CPUs and 32 GB of RAM
WordPress for Android	Java	00:25	01:37	431	10,034	
CakePHP	PHP	00:29	01:49	2,403	55,048	
phpBB 3	PHP	00:39	02:00	1,298	39,610	
SharpZipLib	.NET (C#)	02:19	02:03	1,061	12,185	Windows VM with 8 CPUs and 32 GB of RAM
Hackademic-next	JavaScript	01:33	03:27	789	44,586	Linux VM with 8 CPUs and 32 GB of RAM
prisma	TypeScript	01:02	02:34	67	23,740	
numpy-1.13.3	Python 3	02:52	11:10	309	90,921	
MediaBrowser	Swift	00:33	01:01	26	6,225	macOS VM with 2 CPUs and 8 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 cause a long time to audit and upload to Micro Focus 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` `-import-build-session`	Translate and scan on different machines	"Mobile Build Sessions" on page 45
Analysis	`-quick`	Run a quick scan	"Quick Scan" on the next page
Analysis	`-scan-`	Set the scan precision	"Configuring Scan Speed with

Phase	Option	Description	More Information
	`precision`		Speed Dial" on page 158
Analysis	`-bin`	Scan the files related to a binary	"Breaking Down Codebases" on page 159
Analysis	`-Xmx<size>M|G`	Set maximum heap size	"Memory Tuning" on page 169
Analysis	`-Xss<size>M|G`	Set stack size for each thread	"Memory Tuning" on page 169
Analysis Audit/Upload	`-filter <file>`	Apply a filter using a filter file	"Filter Files" on page 161
Analysis Audit/Upload	`-disable-source-bundling`	Exclude source files from the FPR file	"Excluding Source Code from the FPR" on page 163

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 214.

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. Fortify 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 214 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. Fortify 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 "Fortify Static Code Analyzer Properties Files" on page 187.

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 Micro Focus 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 Micro Focus Fortify Software Security Center server**—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 *Micro Focus 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. Fortify 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.

> **Important!** Be aware that if you modify the configuration files they might be overwritten with an upgrade of Fortify Static Code Analyzer.

> **Notes:**
>
> - By default, Micro Focus 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 if you are scanning 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.
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 very little 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`, `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. This includes reducing the scan time, reducing FPR file size, and tips for opening large FPR files.

Filter Files

Filter files are flat files that you can specify with a scan using the `-filter` option. Use a filter file to filter out specific vulnerability instances, rules, and vulnerability categories. 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 flagging these types of issues and adding them to the FPR. Using a filter file can reduce both the scan time and results file size.

For example, if you are scanning a simple program that just reads a specified file, you might not want to see path manipulation issues, because these are 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 for the scan as shown in the following example:

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

The `MyResults.fpr` does not include any issues with the category Path Manipulation.

For more information about filter files, see .

Excluding Issues from the FPR with Filter Sets

Filters in an issue template determine how the results from Fortify Static Code Analyzer are shown. For example, you can have a filter to put any detected SQL Injection issues into a separate directory called `SQL Injections`, or you might have a filter that hides issues with a confidence below a certain threshold. 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 Micro Focus Fortify Audit Workbench to create a filter and a filter set, and then run the Fortify Static Code Analyzer scan with the filter set. For more detailed instructions about how to create filters and filter sets in Fortify Audit Workbench, see the *Micro Focus Fortify Audit Workbench User Guide*. The following example describes the basic steps for how to create and use a scan-time filter:

1. In this example, suppose you use OWASP Top 10 2017 and you only want to see issues categorized within this standard. Create a filter in Fortify Audit Workbench such as:

   ```
   If [OWASP Top 10 2017] 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 2017 category with 'A' in the name, then it hides it. Because all OWASP Top 10 2017 categories start with 'A' (A1, A2, ..., A10), then any category without the letter 'A' is not in the OWASP Top 10 2017. The filter hides the issues from view in Fortify Audit Workbench, but they are still in the FPR.

2. In Fortify Audit Workbench, create a new filter set called `OWASP_Filter_Set` that contains the previous filter, and then export the issue template to a file called `IssueTemplate.xml`.

3. You can then specify this filter at scan-time with the following command:

   ```
   sourceanalyzer -b MyProject -scan -f myFilteredResults.fpr
   -project-template IssueTemplate.xml -Dcom.fortify.sca.FilterSet=OWASP_
   Filter_set
   ```

In the previous example, the inclusion of the `-Dcom.fortify.sca.FilterSet` property tells Fortify Static Code Analyzer to use the `OWASP_Filter_Set` filter set from the issue template `IssueTemplate.xml`. Any filters that hide issues from view are removed and are not written to the FPR. Therefore, you can reduce the visible number of issues, make the scan very targeted, and reduce the size of the resulting FPR file.

> **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`
 	
Command-Line Option:
`-disable-source-bundling`
	Excludes source code from the FPR.

`com.fortify.sca.`
`FVDLDisableSnippets=true`
 	
Command-Line Option:
`–fvdl-no-snippets`
	Excludes code snippets from the FPR.

The following command-line example uses both options:

```
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 by removing the previous scan results as described in "Altering FPR Files" on page 146.

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`
`=true`
	Excludes the metatable from the FPR. Micro Focus Fortify Audit Workbench uses the metatable to map

Property Name	Description
Command-Line Option: `-disable-metatable`	information in Functions view.

`com.fortify.sca.`
`FVDLDisableDescriptions`
`=true`
Command-Line Option:
`-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`
Command-Line Option:
`-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.
	
	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 Micro Focus 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`
Command-Line Option:
`-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 Micro Focus 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 *Micro Focus Fortify Static Code Analyzer Tools Properties Reference 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 Micro Focus Fortify Audit Workbench.

Property Name	Description

`com.fortify.`
`model.IssueCutOffStartIndex`
`=<num>` (inclusive)
 	
`com.fortify.`
`model.IssueCutOffEndIndex`
`=<num>` (exclusive)
	Sets the start and end index for issue cutoff.
	
	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:
	
	`com.fortify.model.`
	`IssueCutOffStartIndex=0`
	
	`com.fortify.model.`
	`IssueCutOffEndIndex=101`
	
	Because the `IssueCutOffStartIndex` is 0 by default, you do not need to specify this property.

`com.fortify.`
`model.IssueCutOffByCategoryStartIndex=`
`<num>` (inclusive)
 	
`com.fortify.`
`model.IssueCutOffByCategoryEndIndex=`
`<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:
	
	`com.fortify.model.`
	`IssueCutOffByCategoryEndIndex=6`

`com.fortify.` `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.`
`model.MaxEngineErrorCount=`
`<num>`
	Specifies the number of Fortify Static Code Analyzer reported warnings to load from the FPR. For projects with many scan warnings, this can reduce both load time in Fortify Audit Workbench and the amount of memory required to open the FPR.

`com.fortify.` `model.ExecMemorySetting`	Specifies the JVM heap memory size for Fortify Audit Workbench to launch external utilities 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 Fortify recommends that you provide your debug logs to the Micro Focus Fortify 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 Utility

The SCAState command-line utility enables you to see up-to-date state analysis information during the analysis phase. The SCAState utility 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 scan. For more information about how to use the SCAState utility, see the *"Checking the Fortify Static Code Analyzer Scan Status" on page 151*.

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 Micro Focus 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 20: 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 (this could indicate that an attempt was made to translate a file that has a file 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 124, 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 probably 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 tradeoff 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 anything 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 a very long time when analyzing very 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 Micro Focus Fortify 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 two 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 two log files.

Default File Name	Description
sca.log	The standard log provides a log of informational messages, warnings, and errors that occurred in the run of sourceanalyzer.
sca_FortifySupport.log	The Fortify Support log provides: • The same log messages as the standard log file, but with additional details • Additional detailed messages that are not included in the standard log file This log file is only helpful to Micro Focus Fortify Customer Support or the development team to troubleshoot any possible issues.

If you encounter warnings or errors that you cannot resolve, provide the Fortify Support log file to Micro Focus Fortify 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 Micro Focus Fortify 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 Micro Focus Fortify 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 contains the following topics:

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.

> **Note:** Fortify 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.

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. Each filter item is on a separate line in the filter file. You can specify the following filter types:

- Category
- Instance ID
- Rule ID

The filters are applied at different times in the analysis process, based on the type of filter. Fortify Static Code Analyzer applies category and rule ID filters in the initialization phase before any analysis has taken place, whereas an instance ID filter is applied after the analysis phase.

Filter File Example

As an example, the following output is from a scan of the `EightBall.java`, located in the `<sca_install_dir>`/Samples/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 example filter file content 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
```

Appendix B: Fortify Scan Wizard

This section contains the following topics:

Preparing to use Fortify Scan Wizard

Fortify Scan Wizard uses the information you provide to create a script with the commands for Fortify Static Code Analyzer to scan project code and optionally upload the analysis results to Micro Focus Fortify Software Security Center. You can use Fortify Scan Wizard to create a script that runs your scans locally or sends them to Micro Focus Fortify ScanCentral SAST for all or part of the analysis.
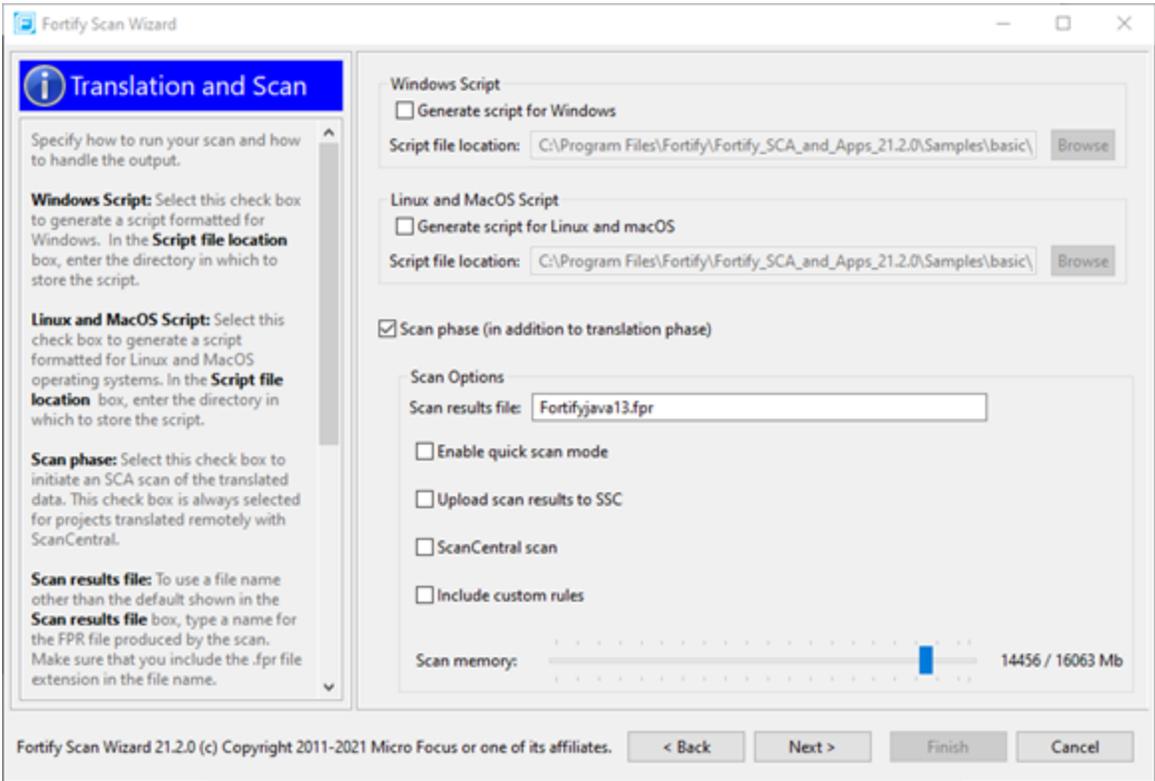
To use Fortify Scan Wizard, you need access to the build directory of the projects you want to scan. The following table describes some of the required information you will need, depending on how you will analyze the project and if you want to upload the scan results to Fortify Software Security Center.

> **Important!** If Fortify Software Security Center or the Fortify ScanCentral SAST Controller uses an SSL connection from an internal certificate authority or a self-signed certificate, you must add the certificate to the Java keystore for Fortify Static Code Analyzer (see "Adding Trusted Certificates" on page 41).

Task	Requirements
Perform a remote analysis (translation and scan phases) with Fortify ScanCentral SAST	Fortify ScanCentral SAST Controller URL **Note:** If you are also uploading analysis results to Fortify Software Security Center, then you do not need to specify a Controller URL. The Fortify ScanCentral SAST that is integrated with the Fortify Software Security Center server is used in this case.
	Make sure that your project is in a language that Fortify ScanCentral SAST supports for translation. See the *Micro Focus Fortify Software System Requirements* for a list of supported languages.
Perform a local Fortify Static Code Analyzer translation and a remote scan with Fortify ScanCentral SAST	Fortify ScanCentral SAST Controller URL
Upload analysis results to Fortify Software Security Center	Fortify Software Security Center server URL **Note:** If you are using Fortify ScanCentral SAST, the Fortify Software Security Center server must be integrated with the Fortify ScanCentral SAST Controller.
	Your Fortify Software Security Center login credentials **Note:** If you do not have Fortify Software Security Center login credentials, you must have an application name and version that exists in Fortify Software Security Center.
	An authentication token of type ToolsConnectToken **Note:** If you do not have a token, you can use Fortify Scan

Task	Requirements
	Wizard to generate one. To do this, you must have Fortify Software Security Center login credentials.

Note: If you generate a script for a Windows system, you cannot run that script on a non-Windows system. Likewise, if you generate a script for a non-Windows system, you cannot run it on a Windows system.

Starting Fortify Scan Wizard

To start Fortify Scan Wizard with Fortify SCA and Applications installed locally, do one of the following, based on your operating system:

- On Windows, select **Start > All Programs > Fortify SCA and Applications *<version>* > Scan Wizard**.

  You can also open a Command Prompt window, and then type `scanwizard`.

- On Linux, navigate to the `<sca_install_dir>`/bin directory, and then run `ScanWizard` from the command line.

- On macOS, navigate to the `<sca_install_dir>`/bin directory, and then double-click `ScanWizard`.

Appendix C: Sample Projects

The Fortify SCA and Applications installation might include several code samples that you can use to when learning to use Fortify Static Code Analyzer. If you installed the sample files, they are in the following directory:

`<sca_install_dir>`/Samples

The `Samples` directory contains two subdirectories: `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 Micro Focus Fortify Audit Workbench.

The `basic` subdirectory includes an assortment of simple language-specific code samples. The `advanced` subdirectory includes more advanced samples including source code to help you integrate Fortify Static Code Analyzer with your bug tracker application. For information on integrating bug tracker applications with Fortify Audit Workbench, see *Micro Focus Fortify Audit Workbench User Guide*.

This section contains the following topics:

Basic Samples

The following table describes the sample files in the `<sca_install_dir>`/Samples/basic directory and provides a list of the vulnerabilities that the samples demonstrate. Many of the samples includes a `README.txt` file that provides details and instructions on its use.

Folder Name	Description	Vulnerabilities
cpp	A C++ sample file and instructions to analyze code that has a simple dataflow vulnerability. Fortify analysis requires a gcc or cl compiler.	Command Injection Memory Leak
database	A `database.pks` sample file. This SQL sample includes issues in SQL code.	Access Control: Database

Folder Name	Description	Vulnerabilities
eightball	A Java application (`EightBall.java`) that exhibits bad error handling. It requires an integer argument. If you supply a file name instead of an integer as the argument, it displays the file contents.	Path Manipulation Unreleased Resource: Streams Unchecked Return Value J2EE Bad Practices: Leftover Debug Code
formatstring	The `formatstring.c` file. Fortify analysis requires a gcc or cl compiler.	Format String
java13	A Java 13 `Sample.java` file.	Privacy Violation Insecure Randomness: Hardcoded Seed J2EE Bad Practices: Leftover Debug Code
java14	A Java 14 `Sample.java` file.	Privacy Violation J2EE Bad Practices: Leftover Debug Code
javascript	The `sample.js` JavaScript file.	Cross-Site Scripting Open Redirect Privacy Violation
nullpointer	The `NullPointerSample.java` file.	Null Dereference
php	Two PHP files: `sink.php` and `source.php`. Analyzing `source.php` reveals simple dataflow vulnerabilities and a dangerous function.	Cross-Site Scripting SQL Injection
sampleOutput	A sample output file (`WebGoat5.0.fpr`) from a WebGoat project.	Various
stackbuffer	The `stackbuffer.cpp` file. Analysis of this sample with Fortify Static Code Analyzer requires a g++ or cl compiler.	Buffer Overflow

Folder Name	Description	Vulnerabilities
toctou	The `toctou.c` file. Analysis of this sample with Fortify Static Code Analyzer requires a gcc or cl compiler.	Time-of-Check/Time-of-Use (Race Condition)
vb6	The `command-injection.bas` file.	Command Injection SQL Injection
vbscript	The `source.asp` and `sink.asp` files.	SQL Injection

Advanced Samples

The following table describes the samples in the `<sca_install_dir>`/Samples/advanced directory. Many of the samples include a `README.txt` file that describes how to analyze the sample and the vulnerabilities that are detected in the analysis.

Folder Name	Description
BugTrackerPlugin `<bug_tracker>`	Includes source code for supported bug tracker plugins.
c++	A sample solution for supported versions of Visual Studio. To use this sample, you must have the following installed: • A supported version of Visual Studio Visual C/C++ • Fortify Static Code Analyzer • To analyze the sample from Visual Studio as described in the README.txt file, you must have the Micro Focus Fortify Extension for Visual Studio installed for your Visual Studio version The code includes a Command Injection issue and an Unchecked Return Value issue.
configuration	A sample Java EE application that has vulnerabilities in its web module deployment descriptor `web.xml`.
crosstier	A sample that has vulnerabilities that span multiple application technologies (Java, PL/SQL, JSP, and struts). The analysis results contain several issues of different types, including two

Folder Name	Description
	Access Control vulnerabilities. One of these is a cross-tier result. It has a dataflow trace from user input in Java code that can affect a `SELECT` statement in PL/SQL.
csharp	A simple C# program that has SQL Injection vulnerabilities. Versions are included for each supported version of Visual Studio. Analysis of the Visual Studio 2017 and 2019 samples reveal SQL Injection, Unreleased Resource, Password Management: Hardcoded Password, and Path Manipulation vulnerabilities. Analysis of the Visual Studio 2022 sample reveals Command Injection vulnerabilities. Other vulnerability categories might also be present, depending on the Rulepack version used in the scan.
customrules	Several simple source code samples and Rulepack files that illustrate how four different analyzers: Semantic, Dataflow, Control Flow, and Configuration interpret rules. This folder also includes several miscellaneous samples of real-world rules that you can use to scan real applications.
ejb	A sample Java EE cross-tier application with servlets and EJBs.
filters	A sample that uses the Fortify Static Code Analyzer `-filter` option.
javaAnnotations	A sample application that illustrates problems that might arise from its use and how to fix the problems with the Fortify Java Annotations. This sample illustrates how the use of Fortify Annotations can result in increased accuracy in the reported vulnerabilities. The `README.txt` file describes the potential problems and solutions associated with the sample application.
JavaDoc	JavaDoc directory for the `public-api` and `WSClient`.
javaWebApp	A sample Java web application.
swift	The iGoat-Swift folder contains the iGoat-Swift source provided by the Open Web Application Security Project (OWASP). To analyze this project, you must have a supported xcodebuild version installed.

Appendix D: Configuration Options

The Fortify SCA and Applications installer places a set of properties files on your system. Properties files contain configurable settings for Micro Focus Fortify Static Code Analyzer runtime analysis, output, and performance.

This section contains the following topics:

Fortify Static Code Analyzer Properties Files

The properties files are located in the *<sca_install_dir>*/Core/config directory.

The installed properties files contain default values. Fortify 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 describes the primary properties files. Additional properties files are described in *Micro Focus Fortify Static Code Analyzer Tools Properties Reference Guide*.

Properties File Name	Description
fortify-sca.properties	Defines the Fortify Static Code Analyzer configuration properties.
fortify-sca-quickscan.properties	Defines the configuration properties applicable for a Fortify Static Code Analyzer quick scan.

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. 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.
	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 table summarizes the properties available for use in the `fortify-sca.properties` file. See for additional properties that you can use in this properties file. The description for each property includes the value type, the default value, the equivalent command-line option (if applicable), and an example.

Translation and Analysis Phase Properties

The properties described in the following table are general properties that apply to the translation and/or analysis (scan) phase.

Property Name	Description
`com.fortify.sca. BuildID`	Specifies the build ID of the build. **Value Type:** String **Default:** (none) **Command-Line Option:** `-b`
`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. **Value Type:** String (path) **Default (Windows):** `${win32.LocalAppdata}\Fortify` **Note:** `${win32.LocalAppdata}` is a special variable that points to the windows Local Application Data shell folder. **Default (Non-Windows):** `$home/.fortify` **Command-Line Option:** `-project-root` **Example:** `com.fortify.sca.ProjectRoot= C:\Users\<username>\AppData\Local\`

`com.fortify.sca. fileextensions.java`
 	
`com.fortify.sca. fileextensions.cs`
 	
`com.fortify.sca. fileextensions.js`
 	
`com.fortify.sca. fileextensions.py`
 	
`com.fortify.sca. fileextensions.rb`
	Specifies how to translate specific file extensions for languages that do not require build integration. The valid file extension types are ABAP, ACTIONSCRIPT, APEX, APEX_TRIGGER, ARCHIVE, ASPNET, ASP, ASPX, BITCODE, BSP, BYTECODE, CFML, COBOL, CSHARP, DOCKERFILE, 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, and XML.
	
	Value Type: String (valid language type)
	
	Default: See the `fortify-sca.properties` file for the complete list.

Property Name	Description

`com.fortify.sca.`
`fileextensions.aspx`
 	
`com.fortify.sca.`
`fileextensions.php`
 	
Note: This is a partial list. For the complete list, see the properties file.
	Examples:
	
	`com.fortify.sca.fileextensions.java=JAVA`
	`com.fortify.sca.fileextensions.cs=CSHARP`
	`com.fortify.sca.fileextensions.js=TYPESCRIPT`
	`com.fortify.sca.fileextensions.py=PYTHON`
	`com.fortify.sca.fileextensions.rb=RUBY`
	`com.fortify.sca.fileextensions.aspx=ASPNET`
	`com.fortify.sca.fileextensions.php=PHP`
	
	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 file 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.
	
	Example:
	`com.fortify.sca.fileextensions.jsp=`
	`oracle:<path_to_script>`

`com.fortify.sca.`
`compilers.javac=`
`com.fortify.sca.`
`util.compilers.JavacCom`
`piler`
 	
`com.fortify.sca.`
`compilers.c++=`
`com.fortify.sca.`
`util.compilers.GppCompi`
`ler`
 	
`com.fortify.sca.`
`compilers.make=`
`com.fortify.sca.`
`util.compilers.Touchles`
`sCompiler`
 	
`com.fortify.sca.`
`compilers.mvn=`
`com.fortify.sca.`
`util.compilers.MavenAda`
`pter`
 	
Note: This is a partial list. For the complete list, see the properties file.
	Specifies custom-named compilers.
	
	Value Type: String (compiler)
	
	Default: See the Compilers section in the `fortify-sca.properties` file for the complete list.
	
	Example:
	
	To tell Fortify Static Code Analyzer that "my-gcc" is a gcc compiler:
	
	`com.fortify.sca.`
	`compilers.my-gcc=`
	`com.fortify.sca.util.compilers.`
	`GccCompiler`
	
	Notes:
	
	• Compiler names can begin or end with an asterisk (*), which matches zero or more characters.
	• 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`

`com.fortify.sca.` `UseAntListener`	If set to true, Fortify Static Code Analyzer includes `com.fortify.dev.ant.SCAListener` in the compiler options. **Value Type:** Boolean **Default:** `false`

Property Name	Description
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, dotnet, golang,java, javascript, jsp, kotlin, objc, php, plsql, python, ruby, scala, sql, swift, tsql, typescript, and vb. **Value Type:** String **Default:** (none) **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, dotnet, golang,java, javascript, jsp, kotlin, objc, php, plsql, python, ruby, scala, sql, swift, tsql, typescript, and vb. **Value Type:** String **Default:** All languages in the specified source are translated unless explicitly excluded with the com.fortify.sca.DISabledLanguages property. **Command-Line Option:** -enable-language
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 131 for more information on how to use file specifiers. **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. **Value Type:** String **Default:** Not enabled **Command-Line Option:** -exclude **Example:** com.fortify.sca.exclude= file1.x;file2.x
com.fortify.sca. CmdlineOptionsFileEncod ing	Specifies the encoding of the command-line options file provided with @*<filename>* (see "Other Options" on page 127). 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 **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. **Value Type:** String **Default:** JVM system default encoding **Example:** com.fortify.sca.CmdlineOptionsFileEncoding=UTF-8

Property Name	Description
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. 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`. **Value Type:** String **Default:** (none) **Command-Line Option:** `-encoding` **Example:** `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.
	
	Value Type: String (path)
	
	Default: `reg`
	
	Example:
	`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). 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. **Value Type:** Boolean **Default:** `false`
com.fortify.sca.AddImpliedMethods	If set to true, Fortify Static Code Analyzer generates implied methods when it encounters implementation by inheritance. **Value Type:** Boolean **Default:**`true`
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`.

Property Name	Description
	Value Type: String **Default:** This property is commented out and all analysis types are used in scans. **Command-Line Option:** `-analyzers`
`com.fortify.sca.DeadCodeFilter`	If set to true, Fortify Static Code Analyzer removes dead code issues, for example because the compiler generated dead code and it does not appear in the source code. **Value Type:** Boolean **Default:** `true`
`com.fortify.sca.DisableDeadCodeElimination`	Dead code is code that can never be executed, such as code inside the body of an `if` statement that always evaluates to false. If this property is set to true, then Fortify Static Code Analyzer does not identify dead code, does not report dead code issues, and reports other vulnerabilities in the dead code, even though they are unreachable during execution. **Value Type:** Boolean **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`. **Value Type:** String **Default:** (none)
`com.fortify.sca.ExitCodeLevel`	Extends the default exit code options. See "Exit Codes" on page 168 for a description of the exit codes and the valid values for this property.
`com.fortify.sca.hoa.Enable`	If set to true, higher-order analysis is enabled. **Value Type:** Boolean **Default:** `true`
`com.fortify.sca.Phase0HigherOrder.Languages`	Specifies a comma-separated list of languages for which to run higher-order analysis. Valid values are `python`, `swift`, `ruby`, `javascript`, and `typescript`. **Value Type:** String **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. Fortify recommends this timeout limit in case some issue causes the analysis to run too long. Fortify 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. **Value Type:** Number **Default:** `2700`
`com.fortify.sca.PrecisionLevel`	Specifies the scan precision. Scans with a lower precision level are performed faster. The valid values are `1` and `2`.

Property Name	Description
	Value Type: Number **Default:** (none) **Command-Line Option:** `-scan-precision`|`-p`
`com.fortify.sca.MaxPassthroughChainDepth`	Specifies the length of a taint path between input and output parameters in a function call. **Value Type:** Integer **Default:** 4
`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. **Value Type:** String **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. **Value Type:** Long **Default:** `300`
`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. **Value Type:** Long **Default:** `60`
`com.fortify.sca.DisableFunctionPointers`	If set to true, disables function pointers during the scan. **Value Type:** Boolean **Default:** `false`
`com.fortify.sca.SuppressLowSeverity`	If set to true, Fortify Static Code Analyzer ignores low severity issues found in a scan. **Value Type:** Boolean **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. **Value Type:** Number **Default:** `1.0`
`com.fortify.sca.analyzer.controlflow.EnableTimeOut`	Specifies whether to enable Control Flow Analyzer timeouts. **Value Type:** Boolean **Default:**`true`
`com.fortify.sca.FilterFile`	Specifies the path to a filter file for the scan. See "Filter Files" on page 177 for more information. **Value Type:** String (path)

Property Name	Description
	Default: (none)
	Command-Line Option: `-filter`
`com.fortify.sca.FilteredInstanceIDs`	Specifies a comma-separated list of IIDs to be filtered out using a filter file.
	Value Type: String
	Default: (none)
	Example:
	`com.fortify.sca.FilteredInstanceIDs=CA4E1623A2424919B98EC19FCA279FFA,` `4418B3DC072647158B3758E6183C14CD`
`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.
	Value Type: String (path)
	Default: (none)
	Command-Line Option: `-bin` or `-binary-name`
`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.
	Value Type: Boolean
	Default: (not enabled)
	Command-Line Option: `-quick`
`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 Micro Focus Fortify Software Security Center server, it uses the issue template assigned to the application version.
	Value Type: String
	Default: (none)
	Command-Line Option: `-project-template`
	Example:
	`com.fortify.sca.ProjectTemplate=` `test_issuetemplate.xml`
`com.fortify.sca.ScanScaModule`	If set to true, Fortify Static Code Analyzer performs modular scan of this project, which enables use of this library's build ID with the `include-modules` option (or the `com.fortify.sca.IncludeScaModules` property) in subsequent scans.
	This property is ignored if the -scan command-line option is specified.
	Value Type: Boolean
	Default: false
	Command-Line Option: `-scan-module`
`com.fortify.sca.IncludeScaModules`	Specifies a comma- or colon-separated list of build IDs for libraries pre-scanned as separate modules to use in the project scan. Each build ID must denote an existing

Property Name	Description
	scanned library.
	Value Type: String (build IDs)
	Default: (none)
	Command-Line Option: `-include-modules`
	Example: `com.fortify.sca.IncludeScaModules=LibA,LibB`
`com.fortify.sca.alias.Enable`	If set to true, enables alias analysis.
	Value Type: Boolean
	Default: `true`
`com.fortify.sca.UniversalBlacklist`	Specifies a colon-separated list of functions to hide from all analyzers.
	Value Type: String
	Default: `.*yyparse.*`
`com.fortify.sca.MultithreadedAnalysis`	Specifies whether Fortify Static Code Analyzer runs in parallel analysis mode.
	Value Type: Boolean
	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.
	Value type: Integer
	Default: (number of available processor cores)

LIM Properties

The properties described in the following table apply to licensing with the LIM.

Property Name	Description
`com.fortify.sca.lim.Url`	Specifies the LIM server URL. Do not edit this value directly with a text editor. Use the command-line option to change this value.
	Value Type: String
	Default: (none)
	Command-Line Option: `-store-license-pool-credentials`
	Example: `https://<ip_address>/LIM.REST.API`
`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.
	Value Type: String

Property Name	Description
	Default: (none)
	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.
	Value Type: String
	Default: (none)
	Command-Line Option: `-store-license-pool-credentials`
`com.fortify.sca.` `lim.ProxyUrl`	Specifies the proxy server used to connect to the LIM server.
	Value Type: String
	Default: (none)
	Examples:
	`http://proxy.example.com:8080` `https://proxy.example.com`
	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.
	Value Type: String
	Default: (none)
	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.
	Value Type: String
	Default: (none)
	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.
	Value Type: Boolean
	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 exits if it cannot obtain a LIM license.
	Value Type: Boolean
	Default: `true`

Rule Properties

The properties described 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 Fortify provided encrypted rules files. **Value Type:** String (path) **Default:** `${com.fortify.Core}/config/rules`
`com.fortify.sca.RulesFile`	Specifies a custom Rulepack or directory. If you specify a directory, all of the files in the directory with the `.bin` and `.xml` extensions are included. **Value Type:** String (path) **Default:** (none) **Command-Line Option:** `-rules`
`com.fortify.sca.CustomRulesDir`	Sets the directory used to search for custom rules. **Value Type:** String (path) **Default:** `${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. **Value Type:** String **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. **Value Type:** Boolean **Default:** (none) **Command-Line Option:** `-no-default-rules`
`com.fortify.sca.NoDefaultIssueRules`	If set to true, disables rules in default Rulepacks that lead directly to issues. Still loads rules that characterize the behavior of functions. This can be helpful when creating custom issue rules. **Value Type:** Boolean **Default:** (none) **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

Property Name	Description
	creating custom source rules.
	Note: Characterization source rules are not disabled.
	Value Type: Boolean
	Default: (none)
	Command-Line Option: `-no-default-source-rules`
`com.fortity.sca.` `NoDefaultSinkRules`	If set to true, disables sink rules in the default Rulepacks. This can be helpful when creating custom sink rules.
	Note: Characterization sink rules are not disabled.
	Value Type: Boolean
	Default: (none)
	Command-Line Option: `-no-default-sink-rules`

Java and Kotlin Properties

The properties described in the following table apply to the translation of Java and Kotlin code.

Property Name	Description
`com.fortify.sca.` `JavaClasspath`	Specifies the class path used to analyze Java or Kotlin source code. Separate multiple paths with semicolons (Windows) or colons (non-Windows).
	Value Type: String (paths)
	Default: (none)
	Command-Line Option: `-cp` or `-classpath`
`com.fortify.sca.` `JdkVersion`	Specifies the Java source code version for Java or Kotlin translation.
	Value Type: String
	Default: `1.8`
	Command-Line Option: `-jdk` or `-source`
`com.fortify.sca.` `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.
	Value Type: String (paths)
	Default: (none)
	Command-Line Option: `-sourcepath`
`com.fortify.sca.` `Appserver`	Specifies the application server to process JSP files. The valid values are `weblogic` or `websphere`.

Property Name	Description
	Value Type: String
	Default: (none)
	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. **Value Type:** String (path) **Default:** (none) **Command-Line Option:** `-appserver-home`
`com.fortify.sca.` `AppserverVersion`	Specifies the version of the WebLogic or WebSphere application server. **Value Type:** String **Default:** (none) **Command-Line Option:** `-appserver-version`
`com.fortify.sca.` `JavaExtdirs`	Specifies directories to include implicitly on the class path for WebLogic and WebSphere application servers. **Value Type:** String **Default:** (none) **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. **Value Type:** Boolean **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`). **Value Type:** String **Default:** (none)
`com.fortify.sca.` `DecompileBytecode`	If set to true, Java bytecode is decompiled for the translation. **Value Type:** Boolean **Default:** `false`
`com.fortify.sca` `jsp.UseSecurityManager`	If set to true, the JSP parser uses JSP security manager. **Value Type:** Boolean **Default:** `true`
`com.fortify.sca.` `jsp.DefaultEncoding`	Specifies the encoding for JSPs.

Property Name	Description
	Value Type: String (encoding)
	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.
	Value Type: Boolean
	Default: `false`
	Command-Line Option: `-legacy-jsp-dataflow`
`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.
	Value Type: Boolean
	Default: `false`
	Command-Line Option: `-show-unresolved-symbols`

Visual Studio and MSBuild Projects Properties

The properties described in the following table apply to the translation of .NET projects.

Property Name	Description

`WinForms.TransformDataBindings`
 	
`WinForms.TransformMessageLoops`
 	
`WinForms.TransformChangeNotificationPattern`
 	
`WinForms.CollectionMutationMonitor.Label`
 	
`WinForms.ExtractEventHandlers`
	Sets various .NET options.
	
	Value Type: Boolean and String
	
	Defaults and Examples:
	
	`WinForms.TransformDataBindings=true`
	
	`WinForms.TransformMessageLoops=true`
	
	`WinForms.TransformChangeNotificationPattern=true`
	
	`WinForms.CollectionMutationMonitor.Label=WinFormsDataSource`
	
	`WinForms.ExtractEventHandlers=true`

`com.fortify.sca.ASPVirtualRoots.<virtual_path>`	Specifies a semicolon-separated list of full paths to virtual roots used.
	
	Value Type: String
	
	Default: (none)
	
	Example:
	
	`com.fortify.sca.ASPVirtualRoots.Library=`
	`c:\\WebServer\\CustomerTwo\\Stuff`
	`com.fortify.sca.ASPVirtualRoots.Include=`
	`c:\\WebServer\\CustomerOne\\inc`

`com.fortify.sca.DisableASPExternalEntries`	If set to true, disables ASP external entries in the scan.

Property Name	Description
	Value Type: Boolean **Default:** `false`

JavaScript and TypeScript Properties

The properties described 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. **Note:** Enabling this property can increase the translation time. **Value Type:** Boolean **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. **Value Type:** String **Default:** `body`, `button`, `div`, `form`, `iframe`, `input`, `head`, `html`, and `p`. **Example:** `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.
	
	Value Type: String
	
	Default: (none)
	
	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. **Value Type:** Boolean **Default:** `false`
`com.fortify.sca.` `skip.libraries.ES6`	Specifies a list of comma- or colon-separated JavaScript technology library files that are not translated. You can use regular expressions in the file names.

Property Name	Description

`com.fortify.sca.`
`skip.libraries.jQuery`
 	
`com.fortify.sca.`
`skip.libraries.javascript`
 	
`com.fortify.sca.`
`skip.libraries.typescript`
	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.
	
	Value Type: String
	
	Defaults:
	
	• ES6: `es6-shim.min.js,system-polyfills.js,`
	`shims_for_IE.js`
	• 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`
	• javascript: `bootstrap.js,`
	`bootstrap.min.js,`
	`typescript.js,`
	`typescriptServices.js`
	• 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 JavaScript translation. **Value Type:** Boolean **Default:** `true`
`com.fortify.sca.` `exclude.unimported.node.modules`	If set to true, only imported node_modules are included in the JavaScript translation. **Value Type:** Boolean **Default:** `true`

Python Properties

The properties described 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. **Value Type:** String (path)

Property Name	Description
	Default: (none)
	Command-Line Option: `-python-path`
`com.fortify.sca.PythonVersion`	Specifies the Python source code version you want to scan. The valid values are 2 and 3.
	Value Type: Number
	Default: 2
	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 74.
	Value Type: Boolean
	Default: `false`
	Command-Line Option: `-python-no-auto-root-calculation`
`com.fortify.sca.DjangoTemplateDirs`	Specifies semicolon-separated (Windows) or colon-separated (non-Windows) list of paths for Django templates. Fortify Static Code Analyzer does not use the TEMPLATE_DIRS setting from the Django `settings.py` file.
	Value Type: String (paths)
	Default: (none)
	Command-Line Option: `-django-template-dirs`
`com.fortify.sca.DjangoDisableAutodiscover`	Specifies that Fortify Static Code Analyzer does not automatically discover Django templates.
	Value Type: Boolean
	Default: (none)
	Command-Line Option: `-django-disable-autodiscover`

Go Properties

The properties described in the following table apply to the translation of Go code.

Property Name	Description
`com.fortify.sca.GOPATH`	Specifies the root directory of your project/workspace.
	Value Type: String
	Default: (GOPATH system environment variable)
`com.fortify.sca.GOROOT`	Specifies the location of the Go installation.
	Value Type: String

Property Name	Description
	Default: (GOROOT system environment variable)
com.fortify.sca. GOPROXY	Specifies one or more comma-separated proxy URLs. You can also specify `direct` or `off`. **Value Type:** String **Default:** (GOPROXY system environment variable)

Ruby Properties

The properties described 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. **Value Type:** String (path) **Default:** (none) **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. **Value Type:** String (path) **Default:** (none) **Command-Line Option:** `-rubygem-path`

COBOL Properties

The properties described 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. **Value Type:** String (path) **Default:** (none) **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. **Value Type:** String **Default:** COBOL390 **Command-Line Option:** `-cobol-dialect`
com.fortify.sca.	Specifies one or more semicolon-separated COBOL checker directives.

Property Name	Description
`CobolCheckerDirectives`	**Value Type:** String **Default:** (none) **Command-Line Option:** `-checker-directives`
`com.fortify.sca.CobolLegacy`	If set to true, enables legacy COBOL translation. **Value Type:** Boolean **Default:** `false` **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). **Value Type:** Boolean **Default:** `false` **Command-Line Option:** `-fixed-format`
`com.fortify.sca.CobolCopyExtensions`	Specifies one or more semicolon- or colon-separated copybook file extensions (legacy COBOL translation only). **Value Type:** String **Default:** (none) **Command-Line Option:** `-copy-extensions`

Apex Properties

The properties described in the following table apply to the translation of Apex and Visualforce code.

Property Name	Description
`com.fortify.sca.Apex`	If set to true, Fortify Static Code Analyzer uses Apex translation for files with the `.cls` extension and Visualforce translation for files with the `.component` extension. **Value Type:** Boolean **Default:** `false` **Command-Line Option:** `-apex`
`com.fortify.sca.ApexObjectPath`	Specifies the absolute path of the custom sObject JSON file `sobjects.json`. **Value Type:** String **Default:** (none) **Command-Line Option:** `-apex-sobject-path`

PHP Properties

The properties described 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 *Micro Focus Fortify Software System Requirements* document. **Value Type:** String **Default:**`7.4` **Command-Line Option:**`-php-version`
`com.fortify.sca.PHPSourceRoot`	Specifies the PHP source root. **Value Type:** Boolean **Default:** (none) **Command-Line Option:**`-php-source-root`

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. **Value Type:** Boolean **Default:** (none)
`com.fortify.sca.AbapIncludes`	When Fortify Static Code Analyzer encounters an ABAP 'INCLUDE' directive, it looks in the named directory. **Value Type:** String (path) **Default:** (none)

Flex and ActionScript Properties

The properties described 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. **Value Type:** String (path)

Property Name	Description
	Default: (none)
	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.
	Value Type: String (path)
	Default: (none)
	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).
	Value Type: String (path)
	Default: (none)
	Command-Line Option: `-flex-source-root`

ColdFusion (CFML) Properties

The properties described 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 interferes with dataflow findings where a variable in an included page is initialized to a tainted value in an earlier-occurring included page.
	Value Type: Boolean
	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.
	Value Type: Boolean
	Default: (not enabled)
`com.fortify.sca.` `SourceBaseDir`	Specifies the base directory for ColdFusion projects.
	Value Type: String (path)
	Default: (none)
	Command-Line Option: `-source-base-dir`

SQL Properties

The properties described 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). **Value Type:** String **Default:**`TSQL` **Command-Line Option:**`-sql-language`

Regex Analysis Properties

The properties described 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. **Value Type:** Boolean **Default:** `true`
`com.fortify.sca.regex.ExcludeBinaries`	If set to true, binary files are excluded from a regular expression analysis. **Value Type:** Boolean **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. **Value Type:** Number **Default:** `10`

Output Properties

The properties described in the following table apply to the analysis output.

Property Name	Description
`com.fortify.sca.ResultsFile`	The file to which results are written. **Value Type:** String **Default:** (none) **Command-Line Option:** `-f`

Property Name	Description
	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 file extension of the file provided with the `-f` option. **Value Type:** String **Default:** `auto` **Command-Line Option:** `-format`
`com.fortify.sca.` `OutputAppend`	If set to true, Fortify Static Code Analyzer appends results to an existing results file. **Value Type:** Boolean **Default:** `false` **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. **Value Type:** Boolean **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. **Value Type:** String **Default:** (none) **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. **Value Type:** String **Default:** (none) **Command-Line Option:** `-build-project`
`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. **Value Type:** String **Default:** (none) **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. **Value Type:** Boolean

Property Name	Description
	Default: (not enabled)
	Command-Line Option: `-machine-output`
com.fortify.sca.
SnippetContextLines	Sets the number of lines of code to display surrounding an issue. The two lines of code on each side of the line where the error occurs are always included. By default, five lines are displayed.
	Value Type: Number
	Default: 2
com.fortify.sca.
FVDLDisableDescriptions	If set to true, excludes Fortify security content descriptions from the analysis results file (FVDL).
	Value Type: Boolean
	Default: `false`
	Command-Line Option: `-fvdl-no-descriptions`
com.fortify.sca.
FVDLDisableEngineData	If set to true, excludes engine data from the analysis results file (FVDL).
	Value Type: Boolean
	Default: `false`
	Command-Line Option: `-fvdl-no-enginedata`
com.fortify.sca.
FVDLDisableLabelEvidence	If set to true, excludes label evidence from the analysis results file (FVDL).
	Value Type: Boolean
	Default: `false`
com.fortify.sca.
FVDLDisableProgramData	If set to true, excludes the `ProgramData` section from the analysis results file (FVDL).
	Value Type: Boolean
	Default: `false`
	Command-Line Option: `-fvdl-no-progdata`
com.fortify.sca.
FVDLDisableSnippets	If set to true, excludes code snippets from the analysis results file (FVDL).
	Value Type: Boolean
	Default: `false`
	Command-Line Option: `-fvdl-no-snippets`
com.fortify.sca.
FVDLStylesheet	Specifies location of the style sheet for the analysis results.
	Value Type: String (path)
	Default: `${com.fortify.Core}/resources/sca/fvdl2html.xsl`

Mobile Build Sessions (MBS) Properties

The properties described 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. **Value Type:** Boolean **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. **Note:** Fortify Static Code Analyzer does not extract the mobile build with this property. **Value Type:** Boolean **Default:** `false`

Logging Properties

The properties described in the following table apply to log files.

Property Name	Description
`com.fortify.sca.` `LogFile`	Specifies the default log file name and location. **Value Type:** String (path) **Default:** `${com.fortify.sca.ProjectRoot}/log/sca.log` and `${com.fortify.sca.ProjectRoot}/log/sca_FortifySupport.log` **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 174 and "Configuring Log Files" on page 174. **Value Type:** String **Default:** `INFO`
`com.fortify.sca.` `ClobberLogFile`	If set to true, Fortify Static Code Analyzer overwrites the log file for each run of sourceanalyzer. **Value Type:** Boolean **Default:** `false` **Command-Line Option:** `-clobber-log`
`com.fortify.sca.` `PrintPerformanceDataAfterScan`	If set to true, Fortify Static Code Analyzer writes performance-related data to the Fortify Support log file after the scan is complete. This value is automatically set to true when in debug mode. **Value Type:** Boolean **Default:** `false`

Debug Properties

The properties described in the following table apply to debug settings.

Property Name	Description
com.fortify.sca. Debug	Includes debug information in the Fortify Support log file, which is only useful for Micro Focus Fortify Customer Support to help troubleshoot. **Value Type:** Boolean **Default:** `false` **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. **Value Type:** Boolean **Default:** (not enabled) **Command-Line Option:** `-debug-verbose`
com.fortify.sca. Verbose	If set to true, includes verbose messages in the Fortify Support log file. **Value Type:** Boolean **Default:** `false` **Command-Line Option:** `-verbose`
com.fortify.sca. DebugTrackMem	If set to true, enables additional performance information to be written to the Fortify Support log. **Value Type:** Boolean **Default:** (not enabled) **Command-Line Option:** `-debug-mem`
com.fortify.sca. CollectPerformanceData	If set to true, enables additional timers to track performance. **Value Type:** Boolean **Default:** (not enabled)
com.fortify.sca. Quiet	If set to true, disables the command-line progress information. **Value Type:** Boolean **Default:** `false` **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. **Value Type:** Boolean **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. **Value Type:** Integer **Quick Scan Default:** `30000` **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`. **Value Type:** String **Quick Scan Default:** `controlflow:buffer` **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 189 to specify an issue template that contains the filter set to use. 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 *Micro Focus Fortify Audit Workbench User Guide*. **Value Type:** String **Quick Scan Default:** `Quick View` **Default:** (none)

Property Name	Description
com.fortify.sca.FPRDisableMetatable	Disables the creation of the metatable, which includes information for the Function view in Micro Focus 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. **Value Type:** Boolean **Quick Scan Default:** `true` **Default:** `false` **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`. **Value Type:** Boolean **Quick Scan Default:** `true` **Default:** `false` **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. **Value Type:** Integer **Quick Scan Default:**`10000` **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. **Value Type:** String **Quick Scan Default:** (none) **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. **Value Type:** Integer **Quick Scan Default:** `10000` **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

Property Name	Description
	analysis, which results in longer scan times.
	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()`.
	Value Type: Integer
	Quick Scan Default: 3
	Default: 5
`com.fortify.sca.` `limiters.MaxFunctionVisits`	Sets the number of times taint propagation analyzer visits functions.
	Value Type: Integer
	Quick Scan Default: 5
	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.
	Note: Fortify does not recommend setting this property to a value larger than 5 because it might increase the scan time.
	Value Type: Integer
	Quick Scan Default: 1
	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.
	Value Type: Integer
	Quick Scan Default: 250
	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.
	Value Type: Integer
	Quick Scan Default: 500
	Default: 4000

Appendix E: Fortify Java Annotations

Fortify provides two versions of the Java Fortify annotations library.

- Annotations with the retention policy set to CLASS (`FortifyAnnotations-CLASS.jar`). With this version of the library, Fortify 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 annotations are not propagated to the bytecode after the code that uses them is compiled.

If you use Fortify products to analyze bytecode of your applications (for example, with Fortify on Demand assessments), then use the version with the annotation retention policy set to CLASS. If you use Fortify products to analyze the source code of your applications, you can use either version of the library. However, Fortify strongly recommends that you use the library with retention policy set to SOURCE.

> **Important!** Leaving Fortify annotations in production code is a security risk because they can leak information about potential security problems in the code. Fortify recommends that you use annotations with the retention policy set to CLASS only for internal Fortify analysis, and never use them in your application production builds.

This section outlines the annotations available. A sample application is included with the Fortify SCA and Applications samples installation in the `<sca_install_dir>/Samples/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 the 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.

Fortify provides three main types of annotations:

- "Dataflow Annotations" on the next page
- "Field and Variable Annotations" on page 220
- "Other Annotations" on page 221

You also can write rules to support your own custom annotations. Contact Micro Focus Fortify 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, Fortify 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.

```java
@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.

```java
@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).

```java
@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 Micro Focus Fortify 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 22.1.0)

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@microfocus.com.

We appreciate your feedback!