Micro Focus
Fortify Static Code Analyzer

Software Version: 18.10

User Guide

Document Release Date: June 2018
Software Release Date: May 2018

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 - 2018 Micro Focus or one of its affiliates

Trademark Notices

Adobe™ is a trademark of Adobe Systems Incorporated.

Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation.

UNIX® is a registered trademark of The Open Group.

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

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-and-services/documentation

Contents

Preface 8
Contacting Micro Focus Fortify Customer Support 8
For More Information 8
About the Documentation Set 8

Change Log 9

Chapter 1: Introduction 11
Fortify Static Code Analyzer 11
Fortify CloudScan 11
Fortify Scan Wizard 12
Fortify Software Security Content 12
About the Analyzers 12
Related Documents 14
All Products 14
Micro Focus Fortify Software Security Center 15
Micro Focus Fortify Static Code Analyzer 15

Chapter 2: Analysis Process Overview 17
Analysis Process 17
Parallel Processing 18
Translation Phase 18
Mobile Build Sessions 19
Mobile Build Session Version Compatibility 19
Creating a Mobile Build Session 19
Importing a Mobile Build Session 19
Analysis Phase 20
Incremental Analysis 20
Translation and Analysis Phase Verification 21

Chapter 3: Translating Java Code 22
Java Command-Line Syntax 22
Java Command-Line Options 23
Java Command-Line Examples 25
Handling Resolution Warnings 25
Java Warnings 25
Using FindBugs 26

Translating Java EE Applications 27
Translating the Java Files 27
Translating JSP Projects, Configuration Files, and Deployment Descriptors 27
Java EE Translation Warnings 27
Translating Java Bytecode 28

Chapter 4: Translating .NET Code 29
About Translating .NET Code 29
.NET Command-Line Syntax 30
Manual .NET Command-Line Syntax 30
.NET Command-Line Options 31
Handling Translation Errors 35
.NET Translation Errors 35
ASP.NET Errors 35

Chapter 5: Translating C and C++ Code 36
C and C++ Code Translation Prerequisites 36
C and C++ Command-Line Syntax 36
Scanning Pre-processed C and C++ Code 37

Chapter 6: Translating JavaScript Code 38
Translating Pure JavaScript Projects 38
Skipping Translation of JavaScript Library Files 38
Translating JavaScript Projects with HTML Files 39
Including External JavaScript or HTML in the Translation 40
Translating AngularJS Code 40
Scanning JavaScript Code 41

Chapter 7: Translating Ruby Code 42
Ruby Command-Line Syntax 42
Ruby Command-Line Options 42
Adding Libraries 43
Adding Gem Paths 43

Chapter 8: Translating ABAP Code 44
About Scanning ABAP Code 44
INCLUDE Processing 45
Importing the Transport Request 45
Adding Fortify Static Code Analyzer to Your Favorites List 46

Running the Fortify ABAP Extractor 47

Chapter 9: Translating Code for Mobile Platforms 49
Translating Apple iOS Projects 49
Prerequisites 49
Xcodebuild Integration Command-Line Syntax 49
Translating Android Projects 50

Chapter 10: Translating Apex and Visualforce Code 51
Apex Translation Prerequisites 51
Apex and Visualforce Command-Line Syntax 51
Apex and Visualforce Command-Line Options 52
Downloading Customized Salesforce Database Structure Information 52

Chapter 11: Translating Flex and ActionScript 54
ActionScript Command-Line Syntax 54
Flex and ActionScript Command-Line Options 54
ActionScript Command-Line Examples 55
Handling Resolution Warnings 56
ActionScript Warnings 56

Chapter 12: Translating COBOL Code 57
Preparing COBOL Source Files for Translation 57
COBOL Command-Line Syntax 58
COBOL Command-Line Options 58

Chapter 13: Translating Other Languages 60
Translating Python Code 60
Python Command-Line Options 61
Python Command-Line Examples 62
Translating PHP Code 62
PHP Command-Line Options 62
Translating ColdFusion Code 63
ColdFusion Command-Line Syntax 63
ColdFusion Command-Line Options 63
Translating SQL 64
PL/SQL Command-Line Example 64
T-SQL Command-Line Example 64
Translating Scala Code 65
Translating ASP/VBScript Virtual Roots 65

Classic ASP Command-Line Example 67
VBScript Command-Line Example 67

Chapter 14: Integrating into a Build 68
Build Integration 68
Make Example 69
Devenv Example 69
Modifying a Build Script to Invoke Fortify Static Code Analyzer 69
Touchless Build Integration 70
Ant Integration 70
Gradle Integration 71
Maven Integration 71
Installing and Updating the Fortify Maven Plugin 71
Testing the Fortify Maven Plugin Installation 72
Using the Fortify Maven Plugin 73
Excluding Files from the Scan 74
MSBuild Integration 74
Setting Windows Environment Variables for Touchless MSBuild Integration 75
Using the Touchless MSBuild Integration 75
Adding Custom Tasks to your MSBuild Project 76

Chapter 15: Command-Line Interface 84
Output Options 84
Translation Options 86
Analysis Options 87
Other Options 90
Directives 91
Specifying Files 92

Chapter 16: Command-Line Utilities 93
Fortify Static Code Analyzer Utilities 93
Other Command-Line Utilities 94
Checking the Fortify Static Code Analyzer Scan Status 94
SCAState Utility Command-Line Options 95
Working with FPR Files from the Command Line 96
Merging FPR Files 97
Displaying Analysis Results Information from an FPR File 98
Extracting a Source Archive from an FPR File 101
Allocating More Memory for FPRUtility 102
Generating Reports from the Command Line 103
Generating a BIRT Report 103

Generating a Legacy Report 105
About Updating Security Content 106
Updating Security Content 106

Chapter 17: Troubleshooting 108
Exit Codes 108
Using the Log File to Debug Problems 109
Translation Failed Message 109
Issue Non-Determinism 109
JSP Translation Problems 110
C/C++ Precompiled Header Files 110
Reporting Issues and Requesting Enhancements 111

Appendix A: Filtering the Analysis 112
Filter Files 112
Filter File Example 112

Appendix B: Scan Wizard 115
Preparing to use the Scan Wizard 115
Starting the Scan Wizard 116
Starting Scan Wizard on a System with Fortify SCA and Applications Installed 116
Starting Scan Wizard as a Stand-Alone Utility 117

Appendix C: Sample Files 118
Basic Samples 118
Advanced Samples 120

Appendix D: Configuration Options 122
Fortify Static Code Analyzer Properties Files 122
Properties File Format 122
Precedence of Setting Properties 123
fortify-sca.properties 123
fortify-sca-quickscan.properties 151

Send Documentation Feedback 155

Preface

Contacting Micro Focus Fortify Customer Support

If you have questions or comments about using this product, contact Micro Focus Fortify Customer Support using one of the following options.

To Manage Your Support Cases, Acquire Licenses, and Manage Your Account

https://softwaresupport.softwaregrp.com

To Call Support

1.844.260.7219

For More Information

For more information about Fortify software products:
https://software.microfocus.com/solutions/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-and-services/documentation

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
18.10 Revision 1: June 1, 2018	Updated: • "Translating Python Code" on page 60 - Provided more information for the -python-path option and added examples
18.10	Added: • "Translating AngularJS Code" on page 40 - Merged the AngularJS Technical Preview information into this guide • "Downloading Customized Salesforce Database Structure Information" on page 52 for Apex translation Updated: • ".NET Command-Line Options" on page 31 and "Fortify.TranslateTask" on page 77 - New options for Shared Projects and Xamarin projects • "Python Command-Line Options" on page 61 - New option for Python version and other minor edits • "Maven Integration" on page 71 - Rebranded Fortify Maven Plugin group ID • "Fortify.TranslateTask" on page 77 - Added Xamarin options for the custom MSBuild translate task • "fortify-sca.properties" on page 123 - New properties for .NET and Python • "About the Analyzers" on page 12 and "fortify-sca.properties" on page 123 - Merged the High Order Analyzer Technical Preview information into this guide • Minor edits to incorporate branding changes
17.20	Added: • " Fortify Software Security Content" on page 12 • "About Translating .NET Code" on page 29 - Add descriptions for the different ways you can translate .NET project • "Translating Scala Code" on page 65 - New feature in this release • "Adding Custom Tasks to your MSBuild Project" on page 76 - Feature updated for the redesigned .NET implementation introduced in version 16.20 • "Allocating More Memory for FPRUtility" on page 102

Software Release / Document Version	Changes
	• "Issue Non-Determinism" on page 109 - Provide instructions for how to disable parallel analysis mode if scan results include issue non-determinism (parallel analysis mode is now enabled by default) Updated: • "Java Command-Line Options" on page 23 - Options -source and -jdk updated to support Java 9 • ".NET Command-Line Options" on page 31 - Added new options for manual translation of ASP.NET, .NET Core, ASP.NET Core, and .NET Standard projects • "Apex Translation Prerequisites" on page 51 - Added more details about prerequisites for translating Apex and Visualforce code • "Translating PHP Code" on page 62 - Added a new option for translating PHP code • "Displaying Analysis Results Information from an FPR File" on page 98 and "Extracting a Source Archive from an FPR File" on page 101 - Added new options for the FPRUtility • "About Updating Security Content" on page 106 - Added new options available for the fortifyupdate utility Removed: • "Migrating Audit Data from Previous FPR Versions" - The migrate feature for SCA 4.x to 5.x was removed from FPRUtility command.
17.10	Added: • "Skipping Translation of JavaScript Library Files" on page 38 - New property file settings to skip translation of library files • "Translating Apex and Visualforce Code" on page 51 - New supported language • "COBOL Command-Line Options" on page 58 - New section to describe the COBOL specific command-line options • "Issue Non-Determinism" on page 109 - New implementation of parallel analysis mode Updated: • "Java Command-Line Options" on page 23- Describes a change to the `-sourcepath` option • "Using the Touchless MSBuild Integration" on page 75 - Added a note about running MSBuild projects in parallel mode Removed: • The Appendix "Issue Tuning" - Issues addressed in this appendix are no longer valid

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 11
About the Analyzers 12
Related Documents 14

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, as well as make security code reviews more efficient, consistent, and complete. Its design enables you to quickly incorporate new third-party and customer-specific security rules.

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 (FPR file) in Fortify Audit Workbench or uploading them to Fortify Software Security Center for analysis, or directly with the results displayed on screen

Note: For information about how to transfer results to Audit Workbench, see the *Micro Focus Fortify Audit Workbench User Guide*.

Fortify CloudScan

You can use Micro Focus Fortify CloudScan to manage your resources by offloading the processor-intensive scanning phase of the Fortify Static Code Analyzer analysis from build machines to a cloud of machines provisioned for this purpose.

After the translation phase is completed on the build machine, Fortify CloudScan generates a mobile build session and moves it to an available machine for scanning. In addition to freeing up the build machines, this process makes it easy to expand the system by adding more resources to the cloud as

needed, without having to interrupt the build process. In addition, users of Fortify Software Security Center can direct Fortify CloudScan to output the FPR file directly to the server.

For more information about Fortify CloudScan, see the *Micro Focus Fortify CloudScan User Guide*.

Fortify Scan Wizard

Fortify Scan Wizard (Scan Wizard) is a utility that enables you to quickly and easily prepare and scan project code using Fortify Static Code Analyzer. With the Scan Wizard, you can run your scans locally, or, if you are using Micro Focus Fortify CloudScan, in a cloud of computers provisioned to manage the processor-intensive scan phase of the analysis.

For more information, see "Scan Wizard" on page 115.

Fortify Software Security Content

Fortify Static Code Analyzer uses a knowledge base of rules to enforce secure coding standards applicable to the codebase for static analysis. Fortify Software Security 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 DSS)

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*.

Security content is required for both translation and analysis. You can download and install security content when you install Fortify Static Code Analyzer. 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 106.

About the Analyzers

Fortify Static Code Analyzer comprises eight vulnerability analyzers: Buffer, Configuration, Content, Control Flow, Dataflow, Higher Order, 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 or not 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.
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.
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.
Higher Order	The Higher Order Analyzer 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, Python, Ruby, and Swift. By default, the Higher Order Analyzer runs when you scan Python, Ruby, and Swift code. You can also run this analyzer when you scan JavaScript. If your website uses complex scripts, you might want to enable the Higher Order Analyzer for JavaScript. For instructions, see "Scanning JavaScript Code" on page 41.
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,

Analyzer	Description
	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.

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-and-services/documentation.

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_Doc_<*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 Software System Requirements Fortify_Sys_Reqs_<*version*>.pdf Fortify_Sys_Reqs_Help_<*version*>	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*>.txt	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	This document describes the new features in Fortify

Document / File Name	Description
Software <version> Fortify_Whats_New_*<version>*.pdf Fortify_Whats_New_Help_*<version>*	Software products.
Micro Focus Fortify Open Source and Third-Party License Agreements Fortify_OpenSrc_*<version>*.pdf	This document provides open source and third-party software license agreements for software components used in Fortify Software.

Micro Focus Fortify Software Security Center

The following documents provide information about Fortify Software Security Center. Unless otherwise noted, these documents are available on the Micro Focus Product Documentation website.

Document / File Name	Description
Micro Focus Fortify Software Security Center User Guide SSC_Guide_<version>.pdf SSC_Help_<version>	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.

Document / File Name	Description
Micro Focus Fortify Static Code Analyzer Installation Guide SCA_Install_*<version>*.pdf SCA_Install_Help_*<version>*	This document contains installation instructions for Fortify Static Code Analyzer and Applications.

Document / File Name	Description
Micro Focus Fortify Static Code Analyzer User Guide SCA_Guide_<*version*>.pdf SCA_Help_<*version*>	This document describes how to 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.
Micro Focus Fortify Static Code Analyzer Performance Guide SCA_Perf_Guide_<*version*>.pdf SCA_Perf_Help_<*version*>	This document provides guidelines for selecting hardware to scan different types of codebases and offers tips for optimizing memory usage and performance.
Micro Focus Fortify Static Code Analyzer Custom Rules Guide SCA_Cust_Rules_Guide_<*version*>.zip SCA_Cust_Rules_Help_<*version*>	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.

Chapter 2: Analysis Process Overview

This section contains the following topics:

Analysis Process 17
Translation Phase 18
Mobile Build Sessions 19
Analysis Phase 20
Incremental Analysis 20
Translation and Analysis Phase Verification 21

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 68.
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 20.
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 21.

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 <results>.fpr
```

The three commands in the previous example illustrates 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.
3. Analyze the project code and produce the 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 substantial 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.

After translation, you can use the `-show-build-warnings` directive to list all warnings and errors that were encountered during the translation phase:

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

To view all of the files associated with a particular 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 22
- "Translating .NET Code" on page 29
- "Translating C and C++ Code" on page 36
- "Translating JavaScript Code" on page 38
- "Translating Ruby Code" on page 42
- "Translating ABAP Code" on page 44
- "Translating Code for Mobile Platforms" on page 49
- "Translating Apex and Visualforce Code" on page 51

- "Translating Flex and ActionScript" on page 54
- "Translating COBOL Code" on page 57
- "Translating Other Languages" on page 60

Mobile Build Sessions

With a Fortify Static Code Analyzer mobile build session, you can translate a project on one machine and analyze it on another. A mobile build session (MBS file) includes all the files needed for the analysis phase. You can then move the MBS file to a different machine for analysis.

You must have the security content (Rulepacks) installed on 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.buildnumber (for example, 18.10.0140). The major and minor portions of the Fortify Static Code Analyzer version numbers on both the translation and the analysis machines must match. For example, 17.10 and 17.1x are compatible.

Note: Before version 16.10, the major portion of the Fortify Static Code Analyzer version number was not the same as the Fortify Software Security Center version number.

To determine the Fortify Static Code Analyzer version number, type `sourceanalyzer -version` 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 MBS file to the machine where you want to run the analysis, you need to import the mobile build session.

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

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

where *<file.mbs>* is the Fortify Static Code Analyzer mobile build session.

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.

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 and any required analysis or output options (see "Analysis Options" on page 87 and "Output Options" on page 84).

The basic command-line syntax for the analysis phase is:

```
sourceanalyzer -b <build_id> -scan -f results.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 <build_id1> -b <build_id2> -b <build_id3> -scan -f
results.fpr
```

Incremental Analysis

With incremental analysis, you can run a full analysis on a project, and then run subsequent incremental scans to analyze only the code that changed since the initial full scan. This reduces the scan time for subsequent incremental scans on the project.

Incremental analysis supports the Configuration and the Semantic analyzers. You can run incremental analysis on projects written in the following languages: Java, C/C++, C#, and Visual Basic.

When you use incremental analysis, consider the following:

- You must use the same build ID that you used in the initial complete analysis in all subsequent incremental scans.
- When you specify the same FPR file name for the initial complete scan and the subsequent scans, all issues are automatically merged with the previous scan.

When Fortify Static Code Analyzer merges the issue results, issues fixed in prior incremental scans are shown as removed, existing issues are shown as updated, and any new issues are shown as new. Otherwise all the issues found in the subsequent scan are shown as new and there is no record of previously fixed issues or existing issues. For more information about viewing results by these groupings in Audit Workbench, see the *Micro Focus Fortify Audit Workbench User Guide*.

To use incremental analysis, translate the code, and then run the initial full scan with the `-incremental-base` option. For example:

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

After you modify the project source code, translate the entire project, and then run any subsequent scans with the `-incremental` option. Specify the same *<build_id>* that you specified in the initial full scan. For example:

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

Translation and Analysis Phase Verification

Audit Workbench result certification indicates whether the code analysis during a scan is complete and valid. The project summary in 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

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

Chapter 3: Translating Java Code

This section contains the following topics:

Java Command-Line Syntax 22
Handling Resolution Warnings 25
Using FindBugs 26
Translating Java EE Applications 27
Translating Java Bytecode 28

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.

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 more convenient for command-line scans

For information about integrating Fortify Static Code Analyzer with Ant, see "Ant Integration" on page 70.

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.

  A class path is the path that the Java runtime environment searches for classes and other resource files. Include all JAR dependencies normally used to build the project. The format is the same as what javac expects (colon- or semicolon-separated list of paths).

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 a 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 <path>`	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 contains the `JspBatchCompiler` script. **Equivalent property name:** `com.fortify.sca.AppServerHome`
`-appserver-version <version>`	Specifies the version of the application server. See the *Micro Focus Fortify Software System Requirements* document for supported versions. **Equivalent property name:** `com.fortify.sca.AppServerVersion`

Java/Java EE Option	Description

`-cp <paths>`|
`-classpath <paths>`
	Specifies the class path to use for analyzing Java source code. The format is same as javac: a colon- or semicolon-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" on page 92.
	Equivalent property name:
	`com.fortify.sca.JavaClasspath`

`-extdirs <dirs>`	Similar to the javac `extdirs` option, accepts a colon- or semicolon-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. You must specify this for FindBugs results as described in "Analysis Options" on page 87.
`-source <version>`| `-jdk <version>`	Indicates the JDK version for which the Java code is written. The valid values for *<version>* are `1.5`, `1.6`, `1.7`, `1.8`, and `1.9`. The default is `1.8`. **Equivalent property name:** `com.fortify.sca.JdkVersion`
`-sourcepath <paths>`	Specifies a colon- or semicolon-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 of the required files that your application uses.

Using FindBugs

FindBugs (http://findbugs.sourceforge.net) is a static analysis tool that detects quality issues in Java code. You can run FindBugs with Fortify Static Code Analyzer and the results are integrated into the analysis results file. Unlike Fortify Static Code Analyzer, which runs on Java source files, FindBugs runs on Java bytecode. Therefore, before you run an analysis on your project, first compile the project and produce the class files.

To see an example of how to run FindBugs automatically with Fortify Static Code Analyzer, compile the sample code `Warning.java` as follows:

1. Go to the following directory:

   ```
   <sca_install_dir>/Samples/advanced/findbugs
   ```

2. Type the following commands to compile the sample:

   ```
   mkdir build

   javac -d build Warning.java
   ```

3. Scan the sample with FindBugs and Fortify Static Code Analyzer as follows:

   ```
   sourceanalyzer -b findbugs_sample -java-build-dir build Warning.java

   sourceanalyzer -b findbugs_sample -scan -findbugs -f findbugs_
   sample.fpr
   ```

4. Examine the analysis results in Audit Workbench:

   ```
   auditworkbench findbugs_sample.fpr
   ```

The output contains the following issue categories:

- Bad casts of Object References (1)
- Dead local store (2)
- Equal objects must have equal hashcodes (1)
- Object model violation (1)
- Unwritten field (2)
- Useless self-assignment (2)

If you group by analyzer, you can see that the Structural Analyzer produced one issue and FindBugs produced eight. The `Object model violation` issue Fortify Static Code Analyzer detected on line 25 is similar to the `Equal objects must have equal hash codes` issue that FindBugs detected. In addition, FindBugs produces two sets of issues (`Useless self-assignment` and `Dead local store`) about the same vulnerabilities on lines 6 and 7. To avoid overlapping results, use the `-filter` option during the scan to apply the `filter.txt` filter file. Note that the filtering is not complete

because each tool filters at a different level of granularity. To see how to avoid overlapping results, scan the sample code using `filter.txt` as follows:

```
sourceanalyzer -b findbugs_sample -scan -findbugs -filter filter.txt
-f findbugs_sample.fpr
```

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 the Java Files

To translate Java EE applications, use the same procedure used to translate Java files. For examples, see "Java Command-Line Examples" on page 25.

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 layout, 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 <build_id> /**/*.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 of the `.jar` and `.class` files required for your application. Also verify that you have all of the `TLD` files for all of your tags and the corresponding JAR files with their tag implementations.

Translating Java Bytecode

In addition to translating source code, you can translate the bytecode in your project. You must specify two configuration properties and include the bytecode files in the Fortify Static Code Analyzer translation phase.

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

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.

To include bytecode in the Fortify Static Code Analyzer translation:

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

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

   This specifies how Fortify Static Code Analyzer processes `.class` and `.jar` files.
2. 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"
   ```

Chapter 4: Translating .NET Code

This chapter describes how to use Fortify Static Code Analyzer to translate Visual Studio .NET and ASP.NET applications built with Visual Studio. Fortify Static Code Analyzer analyzes code written in C#, VB.NET, and ASP.NET (including `.cshtml`, `.vbhtml`, and `.xaml` files). See the *Micro Focus Fortify Software System Requirements* document for supported source code languages and versions of Visual Studio.

This section contains the following topics:

About Translating .NET Code 29
.NET Command-Line Syntax 30
Manual .NET Command-Line Syntax 30
Handling Translation Errors 35

About Translating .NET Code

There are three different methods of translating .NET applications:

- Use the Micro Focus Fortify Package for Visual Studio to translate and analyze your source code project or solution. This is the easiest way to analyze a .NET application because it automatically gathers all the required project information needed to perform the analysis. You can either:
  - Translate and analyze your project from the command line using the Developer Command Prompt for Visual Studio. This is described in ".NET Command-Line Syntax" on the next page.
  - Translate and analyze your project directly from the IDE using the interface provided in the Micro Focus Fortify Package for Visual Studio. See *Micro Focus Fortify Package for Visual Studio User Guide* for more information.
- Include Fortify Static Code Analyzer as part of your MSBuild project. With this option, you can translate your .NET source code on systems where Visual Studio is not installed. You can either:
  - Use the touchless MSBuild integration to launch a Fortify Static Code Analyzer analysis as part of your MSBuild project. For more information, see "Using the Touchless MSBuild Integration" on page 75.
  - Add custom tasks to your MSBuild project to invoke Fortify Static Code Analyzer. For information, see "Adding Custom Tasks to your MSBuild Project" on page 76.
- Use the manual Fortify Static Code Analyzer command line to translate source code and binaries (see "Manual .NET Command-Line Syntax" on the next page. With this method, there is no automatic detection of project information. Therefore, you must provide all the needed information manually with the .NET options (see ".NET Command-Line Options" on page 31).

Note: Fortify recommends that you use either the Micro Focus Fortify Package for Visual Studio (from the IDE or with the Developer Command Prompt for Visual Studio) or the MSBuild touchless integration because all required project information is automatically determined for you.

Fortify recommends that you translate complete .NET projects. To prepare your application for analysis, you need:

- All the C#, VB.NET, and ASP.NET source files

  The supported types of ASP.NET files are: ASPX, ASCX, ASAX, ASHX, ASMX, Master, XAML, BAML, CSHTML, VBHTML, and AXML.
- All third-party DLLs

.NET Command-Line Syntax

Fortify recommends that you use the Developer Command Prompt for Visual Studio to run these commands. If you perform command-line builds with Visual Studio .NET, you can wrap the build command line with an invocation of Fortify Static Code Analyzer to integrate static analysis. You must have the Micro Focus Fortify Package for Visual Studio installed for your version of Visual Studio.

The following example demonstrates the command-line syntax for Visual Studio .NET:

```
sourceanalyzer -b <build_id> devenv Sample1.sln /REBUILD debug
```

This performs the translation phase on all files built with Visual Studio. Make sure that you clean or rebuild the project to include all files.

Note: When you translate .NET code this way, you do not need to specify any of the .NET command-line options described in ".NET Command-Line Options" on the next page. Micro Focus Fortify Package for Visual Studio automatically gathers all information needed for translation and provides it to Fortify Static Code Analyzer.

You can then perform the analysis phase, as shown in the following example:

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

If you are scanning a .NET solution that builds more than one executable, Fortify recommends that you scan each executable separately after the solution is translated. To do this, use the `-binary-name` option and specify the executable (file name or assembly name) as the parameter (see "Analysis Options" on page 87).

Manual .NET Command-Line Syntax

You can use the manual Fortify Static Code Analyzer command-line interface to translate .NET applications.

To translate binaries instead of source files, completely rebuild your project with the Debug configuration option enabled. Make sure PDB files are available for the binaries you intend to translate.

The first time you translate a .NET Core or an ASP.NET Core project, run the `dotnet restore` command first to ensure that all the necessary reference libraries are installed locally.

To manually translate simple .NET projects, which do not contain multiple sub-projects, run Fortify Static Code Analyzer from the command line as follows:

```
sourceanalyzer -b <build_id> -dotnet-version <version> -libdirs <libs>
<project_src_files>
```

Note: You can improve translation by providing information using the command-line options described in ".NET Command-Line Options" below. Unless you specify the `-libdirs-only` option, Fortify Static Code Analyzer uses the .NET framework DLLs that correspond to the target .NET framework version your project uses (specified with the `-dotnet-version`, `-dotnet-core-version`, or `-dotnet-std-version` options) so you do not need to include them with the `-libdirs` option on the command line.

If your project is large and includes sub-projects, perform the translation phase separately for each sub-project and use the same build ID, as follows:

```
sourceanalyzer -dotnet-version <version> -b <build_id>
-libdirs <paths> <project_1_src_files>
...
sourceanalyzer -dotnet-version <version> -b <build_id>
-libdirs <paths> <project_n_src_files>
```

where *<project_1_src_files>* represents the source files in the first sub-project inside your main project or solution, and so on.

.NET Command-Line Options

The following table describes the .NET command-line options.

Note: These options are ***not*** required if you translate the code with the Developer Command Prompt for Visual Studio and you have Micro Focus Fortify Package for Visual Studio installed or if you are using Touchless MSBuild integration (see "MSBuild Integration" on page 74).

.NET Option	Description
`-dotnet-version <version>`	Specifies the .NET framework version. See the *Micro Focus Fortify Software System Requirements* for a list of supported versions. If you specify a value outside the supported version range, Fortify Static Code Analyzer uses the latest supported version. This adds the location of the .NET framework libraries (DLLs) for the specified .NET framework version to the list of DLLs/paths specified by the `-libdirs` option, unless the `–libdirs-only` option is specified. **Equivalent property name:** `com.fortify.sca.DotnetVersion`
`-dotnet-core-version <version>`	Specifies the .NET Core version. This adds the location of .NET framework libraries (DLLs) for the specified .NET framework version to the list of

.NET Option	Description
	DLLs/paths specified by the `-libdirs` option, unless the `-libdirs-only` option is specified. **Equivalent property name:** `com.fortify.sca.DotnetCoreVersion`
`-dotnet-std-version <version>`	Specifies the .NET Standard version. This adds the location of .NET framework libraries (DLLs) for the specified .NET framework version to the list of DLLs/paths specified by the `-libdirs` option, unless the `-libdirs-only` option is specified. **Equivalent property name:** `com.fortify.sca.DotnetStdVersion`
`-libdirs <dlls>` | `<paths>`	Specifies a semicolon-separated list of directories where referenced system or third-party DLLs are located. You can also specify paths to specific DLLs with this option. **Equivalent property name:** `com.fortify.sca.DotnetLibdirs`
`-libdirs-only`	Sets the list of directories or paths to only those specified by the `-libdirs` option. Otherwise, Fortify Static Code Analyzer includes the location of the .NET framework libraries (DLLs) that correspond to the .NET framework version specified with the `–dotnet-version` option. **Equivalent property name:** `com.fortify.sca.DotnetLibdirsOnly`
`-dotnet-shared-files <files>`	Specifies a semicolon-separated list of the source files for all Shared Projects included in the project you are translating. Use this option if your project includes one or more Shared Projects. **Note:** Do not include these shared source files with the primary source files provided to Fortify Static Code Analyzer for translation. **Equivalent property name:** `com.fortify.sca.DotnetSharedFiles`
`-dotnet-output-dir <path>`	Specifies the output directory where the binary (EXE or DLL) built from the project is placed. **Equivalent property name:** `com.fortify.sca.DotnetOutputDir`
`-nuget-cache-dir <path>`	**(.NET Core and .NET Standard projects only)** Overrides the default path to the NuGet cache directory. You might need this option when you translate .NET Core or .NET Standard projects and a custom location for the NuGet cache is specified in project settings. The default path is `%HOMEDIR%.nuget\packages`.

.NET Option	Description
	Equivalent property name: `com.fortify.sca.NugetCacheDir`
`-dotnetwebroot <root_dir>`	**(.NET Web projects only)** Specifies the home directory of an ASP.NET project. **Equivalent property name:** `com.fortify.sca.DotnetWebRoot`
`-dotnet-website`	**(.NET Web projects only)** Indicates that the project is an ASP.NET website. **Note:** Do not specify this option for ASP.NET Web Applications or any other kind of ASP.NET project except Website. **Equivalent property name:** `com.fortify.sca.WebSiteProject`
`-dotnet-applibs <dlls>`	**(.NET Web projects only)** Specifies a semicolon-separated list of additional reference DLLs needed to translate ASP.NET pages. Typically, these are the application assemblies built from the source code of the same sub-project to which target ASP.NET pages belong. **Equivalent property name:** `com.fortify.sca.DotnetWebAppLibs`
`-dotnet-codebehind <files>`	**(.NET Web projects only)** Specifies a semicolon-separated list of source files that are bound to ASP.NET pages (referred to as *code-behind files*). Do not include the files specified as the `-dotnet-codebehind` parameter in the list of other source files targeted for translation on the command line. **Equivalent property name:** `com.fortify.sca.DotnetCodeBehind`
`-aspnetcore`	**(.NET Web projects only)** Indicates a web project (ASP.NET or ASP.NET Core) that targets the .NET Core or .NET Standard framework. **Equivalent property name:** `com.fortify.sca.AspNetCore`
`-dotnet-assembly-name <assembly_name>`	Specifies the name of the target .NET assembly as specified in Visual Studio project settings. **Equivalent property name:** `com.fortify.sca.DotnetAssemblyName`
`-dotnet-preproc-symbols <symbols>`	Specifies a semicolon-separated list of preprocessor symbols used in the source code. For example: `-dotnet-preproc-symbols "DEBUG;TRACE"`

.NET Option	Description
	Equivalent property name: `com.fortify.sca.DotnetPreprocessorSymbols`
`-cs-extern-alias <aliases_path_pairs>`	**(C# projects only)** Specifies a list of external aliases for a specified DLL file in the following format: `alias1,alias2,..=<path_to_dll>`. If multiple DLLs are assigned external aliases, specify multiple `-cs-extern-alias` options on the command line. **Equivalent property name:** `com.fortify.sca.DotnetAlias`

`-vb-compile-options <compile_options>`	**(VB.NET projects only)** Specifies any special compilation options required for the correct translation of the source code, such as `OptionStrict`, `OptionInfer`, and `OptionExplicit`.
	The format for *<compile_options>* is a comma-separated list of: *<option>*=`On` | `Off`. For example:
	`-vb-compile-options "OptionStrict=On,OptionExplicit=Off"`
	Equivalent property name:
	`com.fortify.sca.VBCompileOptions`

`-vb-imports <namespaces>`	**(VB.NET projects only)** Specifies a semicolon-separated list of namespaces imported for all source files in the project. **Equivalent property name:** `com.fortify.sca.VBGlobalImports`
`-vb-mytype <symbol>`	**(VB.NET projects only)** Specifies the value for the `_MYTYPE` preprocessor symbol that is specified in the `<MyType>` tag in the project settings. This is required if the source code to be translated uses My namespace. **Equivalent property name:** `com.fortify.sca.VBMyType`
`-vb-root <namespace>`	**(VB.NET projects only)** Specifies the root namespace for the project as specified in Visual Studio project settings. **Equivalent property name:** `com.fortify.sca.VBRootNamespace`
`-xamarin-android-version <version>`	**(Required for Xamarin.Android projects)** Specifies the target Android SDK version for Xamarin Android projects. If you specify an Android SDK version that is not available, Fortify Static Code Analyzer uses the latest installed version. This adds Xamarin.Android reference libraries (DLLs) for the specified Android SDK version to the list of DLLs/paths specified by the `-libdirs` option, unless the `-libdirs-only` option is specified.

.NET Option	Description
	Equivalent property name: `com.fortify.sca.XamarinAndroidVersion`
`-xamarin-ios-version <version>`	**(Required for Xamarin.iOS projects)** Specifies the target iOS SDK version for Xamarin iOS projects. If you specify an iOS SDK version that is not available, Fortify Static Code Analyzer uses the latest installed version. This adds Xamarin.iOS reference libraries (DLLs) for the specified iOS SDK version to the list of DLLs/paths specified by the `-libdirs` option, unless the `-libdirs-only` option is specified. **Equivalent property name:** `com.fortify.sca.XamariniOSVersion`

Handling Translation Errors

To see all warnings that Fortify Static Code Analyzer generated during translation, type the following command before you start the scan phase:

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

.NET Translation Errors

You might see the following error for .NET:

```
Translator execution failed. Please consult the Troubleshooting section of
the User Manual. Translator returned status <large_negative_number>
```

This error indicates that Fortify Static Code Analyzer could not successfully translate of all the source files in your project. Report this issue to Micro Focus Fortify Customer Support for investigation.

ASP.NET Errors

Any error reported for ASP.NET translation is prefixed with `ASP.Net Translation:` and is followed by detailed information about the error. This error indicates that Fortify Static Code Analyzer could not successfully translate all the ASP.NET pages in your project. Report this issue to Micro Focus Fortify Customer Support for investigation.

Chapter 5: Translating C and C++ Code

This section contains the following topics:

C and C++ Code Translation Prerequisites 36
C and C++ Command-Line Syntax 36
Scanning Pre-processed C and C++ Code 37

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.

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:

- *<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.
- *<sca_options>* are options passed to Fortify Static Code Analyzer.
- *<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 <build_id> 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.

Chapter 6: Translating JavaScript Code

You can analyze JavaScript projects that can contain either pure JavaScript source files or a combination of JavaScript and HTML files.

This section contains the following topics:

Translating Pure JavaScript Projects 38
Skipping Translation of JavaScript Library Files 38
Translating JavaScript Projects with HTML Files 39
Including External JavaScript or HTML in the Translation 40
Translating AngularJS Code 40
Scanning JavaScript Code 41

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>`.

Skipping Translation of JavaScript Library Files

You can avoid translating specific JavaScript library files 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.AngularJS`
- `com.fortify.sca.skip.libraries.ES6`
- `com.fortify.sca.skip.libraries.jQuery`
- `com.fortify.sca.skip.libraries.javascript`

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 JavaScript library file reference:

```
<script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js">
</script>
```

By default, the `com.fortify.sca.skip.libraries.AngularJS` property in the `fortify-sca.properties` file includes `angular.min.js`, and therefore Fortify Static Code Analyzer does not translate the file shown in the previous example. Also, any local copy of the `angular.min.js` file is not translated.

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 86.

Translating JavaScript Projects with HTML Files

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

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

When you set the `com.fortify.sca.EnableDOMModeling` property to true, Fortify Static Code Analyzer generates JavaScript code to model the DOM tree structure in the HTML files.

Note: If you enable this option, the duration of the analysis phase might increase (because there is more translated code to analyze). However, if you leave the default value of `false`, you might get reports of false negatives of DOM-related attacks, such as DOM-based cross-site scripting.

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 using the `com.fortify.sca.DOMModeling.tags` property. By default, Fortify Static Code Analyzer includes a limited set of DOM element tags such as `html`, `head`, `input`, `button`, and `iframe`.

For example, to include the HTML tags `div` and `p` in the DOM model, use the following command:

```
sourceanalyzer –b <build_id> <js_file_or_dir>
-Dcom.fortify.sca.DOMModeling.tags=”div,p”
```

Including External JavaScript or HTML in the Translation

To include external JavaScript or HTML files that are specified with the `src` attribute, you can whitelist specific domains to have Fortify Static Code Analyzer download and include them in translation. 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'/>
```

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 whitelist property value. For example, if the src tag in the original HTML file specifies to download files from `www.google.com`, you can whitelist just the `google.com` domain.

To whitelist 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.

Translating AngularJS Code

You can analyze AngularJS 1.x source code. Fortify Static Code Analyzer supports the following application architectures:

- MVC
- Partials
- UI Router

Fortify Static Code Analyzer can detect AngularJS specific configuration-related issues. It can also find dataflow issues, such as Cross-Site Scripting: DOM, in small AngularJS projects.

To enable AngularJS scanning, set the following two properties in the `fortify-sca.properties` file to `true` (or include these properties settings on the command line using the `-D` option):

```
com.fortify.sca.EnableDOMModeling=true
com.fortify.sca.hoa.Enable=true
```

Scanning JavaScript Code

You can configure the Higher Order Analyzer for JavaScript analysis, which improves the ability to track dataflow through higher-order code. However, adding analysis of JavaScript might result in long scan times. To configure the Higher Order Analyzer for JavaScript, set the `com.fortify.sca.Phase0HigherOrder.Languages` property to include `javascript` in the `fortify-sca.properties` file or specify the property directly on the command line using the `-D` option as follows:

```
-Dcom.fortify.sca.Phase0HigherOrder.Languages=javascript
```

Three property settings control the effort the Higher Order Analyzer expends to find call targets, and therefore the amount of time it takes. The analyzer goes through three major passes: static inits, main, and fixpoint. The main and fixpoint passes are further divided into a variable number of subpasses. To see log messages about these passes, turn on debug logging and search the log file for `Pushdown HOA`. Allow the static inits and the main passes to run to completion, so that most call sites are resolved. For a description of these properties, see "fortify-sca.properties" on page 123 and look for property names that include *Phase0HigherOrder*.

Chapter 7: Translating Ruby Code

This section contains the following topics:

Ruby Command-Line Syntax 42
Adding Libraries 43
Adding Gem Paths 43

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 (*) wild card to select all Ruby files in a specified directory. For example, to find all of the Ruby files in a directory called `src`, use the following `sourceanalyzer` command:

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

Ruby Command-Line Options

The following table describes the Ruby translation options.

Ruby Option	Description
`-ruby-path`	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`
`-rubygem-path`	Specifies the path(s) to a RubyGems location (see "Adding Gem Paths" on the next page) **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
```

To use multiple libraries, use a delimited list. On Windows, separate the paths with a semicolon; and on all other platforms use a colon, as in the following non-Windows example:

```
-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, add them by specifying a delimited list of paths such as:

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

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

Chapter 8: 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 chapter assumes you have Fortify Static Code Analyzer installed and have a basic understanding of Fortify Static Code Analyzer, SAP, and ABAP.

This section contains the following topics:

About Scanning ABAP Code 44
Importing the Transport Request 45
Adding Fortify Static Code Analyzer to Your Favorites List 46
Running the Fortify ABAP Extractor 47

About Scanning ABAP Code

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 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 `LIKE programname` 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 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

ABAP scanning is available as a premium component of Fortify Static Code Analyzer. If you purchased a license that includes this capability, you need to import the Fortify ABAP Extractor transport request on your SAP Server.

The Fortify transport request is located in the `SAP_Extractor.zip` package. The package is located in the `Tools` directory:

```
<sca_install_dir>/Tools/SAP_Extractor.zip
```

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

- `K9000XX.NSP` (where the "XX" is the release number)
- `R9000XX.NSP` (where the "XX" 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. Your SAP administrator or an individual authorized to install transport requests on the system should import the transport request.

The NSP 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 was created on a system running SAP release 7.02, SP level 0006. If you are running a different SAP release, you might get the transport request import error: `Install release does not match the current version`.

This causes the installation to fail. To resolve this issue:

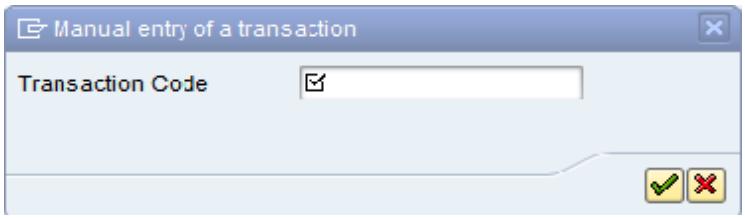
1. Run the transport request import again.

   The Import Transport Request dialog box opens.
2. Click the **Options** tab.
3. Select the **Ignore Invalid Component Version** check box.
4. Complete the import procedure.

Adding Fortify Static Code Analyzer to Your Favorites List

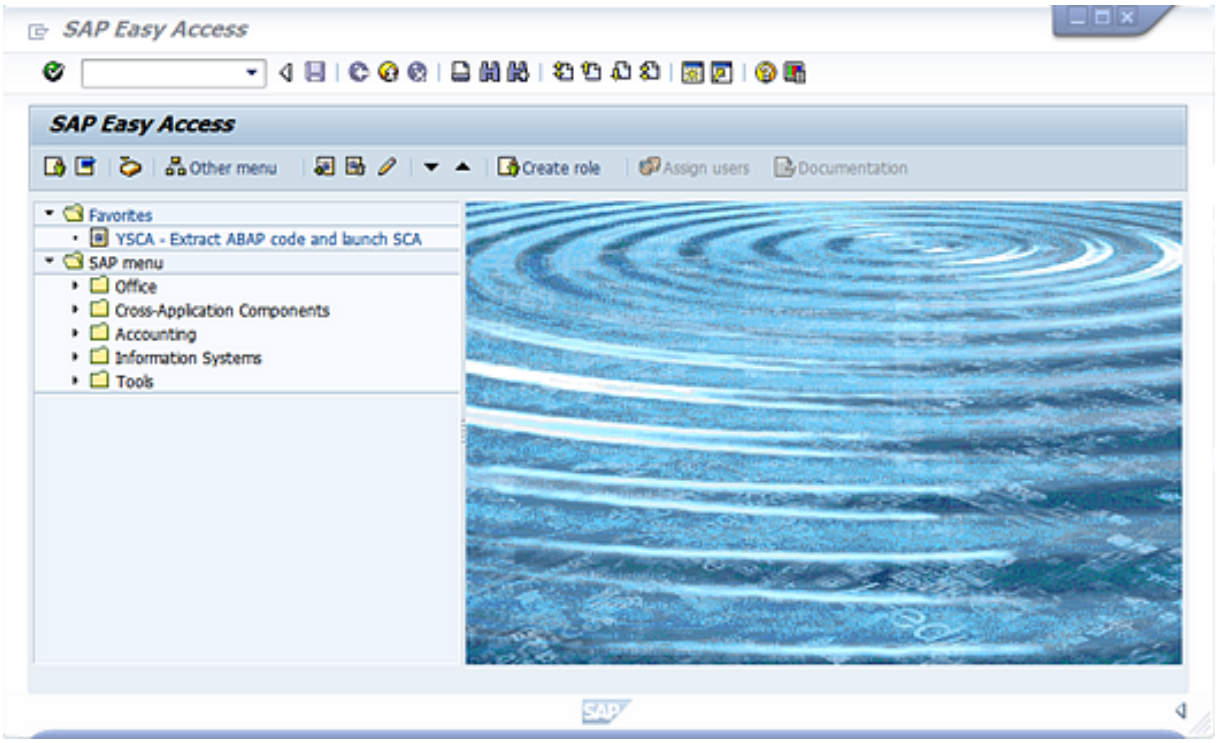
Adding Fortify Static Code Analyzer to your Favorites list is optional, but doing so can make it quicker to access and launch Fortify Static Code Analyzer scans. The following steps assume that you use the user menu in your day-to-day work. If your work is done from a different menu, add the Favorites 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.

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 YHP_FORTIFY_SCA object.

2. Provide the requested information.

Section	Data
Objects	Type the name of the software component, package, program, or BSP application you want to scan.
Sourceanalyzer parameters	**FPR File Path**: Type the directory where you want to store your FPR file. Include the name for the FPR file in the path name. **Working Directory**: Type the directory where you want the extracted source code copied. **Build-ID**: Type the build ID for the scan. **Translation Parameters**: Type any optional Fortify Static Code Analyzer comand-line translation options. **Scan Parameters**: Type any optional Fortify Static Code Analyzer command-line scan options. **ZIP File Name**: 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.
Actions	**Download**: Select this check box to have Fortify Static Code Analyzer download the 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. **Scan**: Select this check box to request a scan. **Launch AWB**: Select this check box to start Audit Workbench and load the FPR. **Create ZIP**: Select this check box to compress the output. **Process in Background**: Select this check box to have processing occur in the background.

3. Click **Execute**.

Chapter 9: 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

This section contains the following topics:

Translating Apple iOS Projects 49
Translating Android Projects 50

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 Apple Xcode command-line tool to identify the project source files.

Prerequisites

- Install Xcode command-line tools on the path.
- 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.
- Ensure that you have any dependencies required to build the project available.
- To translate Swift code, make sure that all third party modules, including cocoapods are available. Bridging headers must also be available. However, Xcode usually generates them automatically during the build.
- To translate Objective-C projects, ensure that the headers for third-party libraries are available. Fortify Static Code Analyzer translation does not require object files and static/dynamic library files.
- To translate WatchKit applications, make sure that you translate both the iPhone application target and the WatchKit extension target.

Xcodebuild Integration Command-Line Syntax

The command-line syntax to translate a single iOS Xcode project is:

```
sourceanalyzer -b <build_id> -clean
```

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

where *<compiler_options>* are options passed to Xcode.

The following examples illustrate usage patterns for the supported compilers. Run the following command samples from the directory that contains the project files. Note that the first command is optional (`-clean`) and removes any previous build artifacts.

```
xcodebuild [<options>] clean

sourceanalyzer -b <build_id> -clean

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

To scan the application artifact files, type:

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

Note: Fortify Static Code Analyzer compiles the source code when you run these commands.

Translating Android Projects

Translating Android projects is similar to translating Java code. See "Translating Java Code" on page 22 for more information.

Chapter 10: Translating Apex and Visualforce Code

This section contains the following topics:

Apex Translation Prerequisites 51
Apex and Visualforce Command-Line Syntax 51
Apex and Visualforce Command-Line Options 52
Downloading Customized Salesforce Database Structure Information 52

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" on the next page.

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.fileextension.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.
`-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.

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 11: Translating Flex and ActionScript

This section contains the following topics:

ActionScript Command-Line Syntax 54
Handling Resolution Warnings 56

ActionScript Command-Line Syntax

The basic command-line syntax for translating ActionScript is:

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

where:

<libs> is a semicolon-separated list (Windows) or a colon-separated list (non-Windows systems) 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`	The location of the root of a valid Flex SDK. This folder should contain a frameworks folder that contains a `flex-config.xml` file. It should also contain a `bin` folder that contains an MXMLC executable. **Equivalent property name:** `com.fortify.sca.FlexSdkRoot`
`-flex-libraries`	A colon- or semicolon-separated list (colon on most platforms, semicolon on Windows) of library names that you want to "link" to. 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 and ActionScript Option	Description
`-flex-source-roots`	A colon- or semicolon-separated list of root directories in which MXML sources are located. Normally, these contain a subfolder named `com`. For instance, if a Flex source root is given that is pointing to `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 resolving all ActionScript.

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. As a consequence, 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 also identifies 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 of the required libraries. You might need to specify additional SWC or SWF Flex libraries (`-flex-libraries` option, or `com.fortify.sca.FlexLibraries` property) so that Fortify Static Code Analyzer can complete the analysis.

Chapter 12: Translating COBOL Code

This section contains the following topics:

Preparing COBOL Source Files for Translation 57
COBOL Command-Line Syntax 58
COBOL Command-Line Options 58

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 specialized Fortify license specific for COBOL scanning capabilities. Contact Micro Focus Fortify Customer Support for more information about scanning COBOL and the required license.

Preparing COBOL Source Files for Translation

Fortify Static Code Analyzer runs only on the supported systems listed in the *Micro Focus Fortify Software System Requirements* document, not on mainframe computers. Before you can scan a COBOL program, you must copy the following program components to the system where you run Fortify Static Code Analyzer:

- COBOL source code
- All copybook files that the COBOL source code uses
- All SQL INCLUDE files that the COBOL source code references

Fortify Static Code Analyzer processes only top-level COBOL sources. Do not include copybook or SQL INCLUDE files in the directory or the subdirectory where the COBOL sources reside. Fortify recommends that you place your COBOL source code in a folder called `sources/` and your copybooks in a folder called `copybooks/`. Create these folders at the same level. Emulate the following structure with the translation command:

```
sourceanalyzer -b <build_id> -noextension-type COBOL -copydirs copybooks/
sources/
```

If your COBOL source code contains:

```
COPY FOO
```

where `FOO` is a copybook file or a SQL INCLUDE file, then the corresponding file in the `copybooks` folder or the `SQL INCLUDE` folder, as specified with the `-copydirs` option, should be `FOO`. The `COPY` command can also take a directory-file-path structure rather than a just a file name. Follow the same translation command structure, using a directory-file-path structure instead of just the file name.

Preparing COBOL Source Code Files

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 <path>
```

Specify the directory and folder with all COBOL files as the parameter to the sourceanalzyer command, and Fortify Static Code Analyzer translates all the files in that directory and folder without any need for COBOL file extensions.

Preparing COBOL Copybook Files

Fortify Static Code Analyzer does not identify copybooks by the file extension. All copybook files must therefore retain the names used in the COBOL source code `COPY` statements. Do not place copybook files in the same folder as the main COBOL source files, but instead, put them in a directory named `copybooks/` at the same level as the folder that contains your COBOL source files.

If the copybooks have file extensions, use the `-copy-extensions` option to specify the copybook file extensions. For more information, see "COBOL Command-Line Options" below.

COBOL Command-Line Syntax

Free-format COBOL is the default translation and scan mode for Fortify Static Code Analyzer. Fortify Static Code Analyzer supports the analysis of fixed-format COBOL. When you analyze fixed-format COBOL, you must include the `-fixed-format` command-line option for both the translation and scan commands. For more information, see "COBOL Command-Line Options" below.

The basic syntax to translate a single free-format COBOL source code file is:

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

The basic syntax to scan a translated free-format COBOL program is:

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

COBOL Command-Line Options

The following table describes the COBOL command-line options.

COBOL Option	Description
`-copy-extensions <ext>`	Specifies one or more colon-separated copybook file extensions.

COBOL Option	Description
`-copydirs <path>`	Directs Fortify Static Code Analyzer to search a list of colon-separated paths for copybooks and `SQL INCLUDE` files.
`-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. Use this option for both the translation and the scan commands. If your COBOL code is IBM Enterprise COBOL, then it is most likely 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 a lot of parsing errors in the COBOL translation **Equivalent property name:** `com.fortify.sca.CobolFixedFormat`

Chapter 13: Translating Other Languages

This section contains the following topics:

Translating Python Code 60
Translating PHP Code 62
Translating ColdFusion Code 63
Translating SQL 64
Translating Scala Code 65
Translating ASP/VBScript Virtual Roots 65
Classic ASP Command-Line Example 67
VBScript Command-Line Example 67

Translating Python Code

Fortify Static Code Analyzer translates Python applications, and processes files with the `.py` extension as Python source code.

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

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

Note: If you translate your files with multiple commands that have the same *<build_id>*, always specify the same Python version. Do not translate different versions of Python code with the same build ID.

To translate Python applications and prepare for a scan, Fortify Static Code Analyzer searches any import files for the application. Fortify Static Code Analyzer does not respect the `PYTHONPATH` environment variable, which the Python runtime system uses to find imported files. Therefore, you can provide this information to Fortify Static Code Analyzer by specifying all paths used to import packages or modules 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. Fortify recommends that you add the path to the standard Python library that is shipped with your Python release to the `-python-path` list.

Note: Fortify Static Code Analyzer translates all import files located in the directory path defined by the `-python-path` option. Subsequently, translation might take a long time to complete.

Using the Django Framework with Python

Fortify Static Code Analyzer supports the Django Framework with Python 2 builds.

To scan 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
```

For the translation phase, include the following option in the Fortify Static Code Analyzer command:

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

Python Command-Line Options

The following table describes the Python options.

Python Option	Description
-python-path *<paths>*	Specifies a colon- or semicolon-separated list of additional import directories. You can use the -python-path option to specify all paths used to import packages or modules. **Equivalent property name:** com.fortify.sca.PythonPath
-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
-django-template-dirs *<template_path>*	Specifies to scan code created using the Django framework where *<template_path>* is the location of the Django template directories. **Equivalent property name:** com.fortify.sca.DjangoTemplateDirs

Python Command-Line Examples

To translate Python 2 code, type:

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

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
```

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" below.

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 7.0. For a list of valid versions, see the *Micro Focus Fortify Software System Requirements*. **Equivalent property name:** `com.fortify.sca.PHPVersion`

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" on page 92.

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 provided that you specify an appropriate parameter for the `-source-base-dir` option.

ColdFusion Command-Line Options

The following table describes the ColdFusion 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

By default, files with the `.sql` extension are assumed to be T-SQL rather than PL/SQL on Windows platforms. If you are using Windows and have PL/SQL files with the `.sql` extension, configure Fortify Static Code Analyzer to treat them as PL/SQL. To change the default behavior, set the `com.fortify.sca.fileextensions.sql` property in `fortify-sca.properties` to `TSQL` or `PLSQL`.

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>
```

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

To translate Scala code, you must have a standard Lightbend Enterprise Suite license. Download the Scala translation plugin from Lightbend. For instructions on how to download 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 Lightbend's Scala translation plugin, and then translate the other source code with sourceanalyzer using the same build ID before you run the scan phase.

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 14: Integrating into a Build

You can integrate the analysis into supported build tools.

This section contains the following topics:

Build Integration 68
Modifying a Build Script to Invoke Fortify Static Code Analyzer 69
Touchless Build Integration 70
Ant Integration 70
Gradle Integration 71
Maven Integration 71
MSBuild Integration 74

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. For information about integrating with Xcodebuild, see "Xcodebuild Integration Command-Line Syntax" on page 49.

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

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

where *<build_id>* is the name of your build tool, such as make, gmake, 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.

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" on the next page.

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 commands:

```
make clean
sourceanalyzer -b <build_id> make
make install
```

Devenv Example

If you build a Visual Studio .NET project with the following build command:

```
devenv MyProject.sln /REBUILD
```

then you can translate and compile the project with the command:

```
sourceanalyzer -b <build_id> devenv MyProject.sln /REBUILD
```

Note: Integration with Visual Studio .NET requires that you install the Micro Focus Fortify Package for Visual Studio for your specific version of Visual Studio.

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 "Build Integration" on page 68) 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.

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 buildxyz gradle clean build
sourceanalyzer -b buildxyz gradle --info assemble
```

Note: If you use the Fortify Static Code Analyzer `-verbose` option, then you must also include the `-gradle` option. For example:

```
sourceanalyzer -b buildxyz -gradle -verbose gradle assemble
```

Maven Integration

Micro Focus Fortify Static Code Analyzer includes a Maven plugin that provides a way to add Fortify Static Code Analyzer clean, translate, scan, Fortify CloudScan, and FPR upload capabilities to your Maven project builds. 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 version information is displayed 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 that is 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

You can use the Fortify Maven Plugin as a Maven plugin or in a Fortify Static Code Analyzer build integration.

Maven Plugin

To analyze your code as a plugin in Maven, see the documentation included in the Fortify Maven Plugin. The following table describes where to find the documentation after the Fortify Maven Plugin is properly installed.

Package Type	Documentation Location
Binary	`<root_dir>/docs/usage.html`
Source	`<root_dir>/sca-maven-plugin/target/site/usage.html`

Fortify Static Code Analyzer Build Integration

To analyze your files as part of a Fortify Static Code Analyzer build integration:

1. Install the target application in the local repository:

   ```
   mvn install
   ```

2. Clean out the previous build:

   ```
   sourceanalyzer -b <build_id> -clean
   ```

3. Translate the code with one of the options listed in the following table.

Translation Command Options
`sourceanalyzer -b <build_id> [<sca_options>] mvn`
`sourceanalyzer -b <build_id> [<sca_options>] mvn com.fortify.sca.plugins.maven:sca-maven-plugin:<ver>:translate`
`sourceanalyzer -b <build_id> [<sca_options>] mvn com.fortify.sca.plugins.maven:sca-maven-plugin:translate`
`sourceanalyzer -b <build_id> [<sca_options>] mvn sca:translate` **Note:** To use this version of the command, you must have placed a copy of the `settings.xml` file in the local repository.

4. Run the scan:

```
sourceanalyzer -b <build_id> [<sca_scan_options>] -scan -f <file>.fpr
```

Excluding Files from the Scan

If you do not want to include all of the files in your project or solution, you can direct Fortify Static Code Analyzer to exclude selected files from your scan. To specify the files you want to exclude, add the `-D` option with the `fortify.sca.exclude` property to the translate step as shown in the following example:

```
-Dfortify.sca.exclude="fileA;fileB;fileC;"
```

Note: On Windows, separate the file names with a semicolon; and on all other platforms use a colon. Wild cards are supported; use a single asterisk (*) to match part of a file name and use two asterisks (**) to recursively match directories.

For example, for a Java 1.8 project, issue the following command to translate the source code and exclude three source files:

```
mvn com.fortify.sca.plugins.maven:sca-maven-plugin:<ver>:translate
-Dfortify.sca.source.version=1.8 -Dfortify.sca.exclude="fileA;fileB;fileC;
```

MSBuild Integration

Fortify Static Code Analyzer enables you to translate your .NET source code as part of your MSBuild build process. With the Fortify Static Code Analyzer MSBuild integration, you can translate files on machines where Visual Studio is not installed. See the *Micro Focus Fortify Software System Requirements* document for the supported MSBuild versions. The MSBuild executable allows for translation of the following project/source code types:

- C/C++ Console Applications
- C/C++ Libraries
- Visual C# and Visual Basic Websites
- Visual C# and Visual Basic Libraries
- Visual C# and Visual Basic web Applications
- Visual C# and Visual Basic Console Applications

Note: MSBuild integration only works for the projects that you can build with MSBuild. If you cannot build your project with MSBuild, use the Micro Focus Fortify Package for Visual Studio to analyze it.

This section describes how to run a Fortify Static Code Analyzer analysis as part of your MSBuild project.

Setting Windows Environment Variables for Touchless MSBuild Integration

When you integrate Fortify Static Code Analyzer into your MSBuild process, there are several Windows environment variables that you can set. If you do not set these variables, Fortify Static Code Analyzer uses the variable defaults. After you set the Windows environment variables, successive builds use the same variables until you change them. The following table lists the environment variables that you can set.

Environment Variable	Description	Default Value
`FORTIFY_MSBUILD_BUILDID`	Specifies the Fortify Static Code Analyzer build ID	Build ID specified on the command line
`FORTIFY_MSBUILD_DEBUG`	Puts the logger and Fortify Static Code Analyzer into debug mode	False
`FORTIFY_MSBUILD_MEM`	Specifies the memory used to invoke Fortify Static Code Analyzer (for example, `-Xmx2000M`)	Automatic allocation based on physical memory available on the system
`FORTIFY_MSBUILD_LOG`	Specifies the location for the log file	`${win32.LocalAppdata}/Fortify/MSBuildPlugin`
`FORTIFY_MSBUILD_SCALOG`	Specifies the location (absolute path) for the Fortify Static Code Analyzer log file	Location specified by the `com.fortify.sca.LogFile` property in the `fortify-sca.properties` file
`FORTIFY_MSBUILD_LOGALL`	Specifies whether the plugin logs every message passed to it (this creates a large amount of information)	False

Using the Touchless MSBuild Integration

The touchless MSBuild integration requires the `FortifyMSBuildTouchless.dll` located in the *<sca_install_dir>*`/Core/lib` directory. Run the commands described in this section from the Developer Command Prompt for Visual Studio. You must provide a solution for the command.

Note: The touchless MSBuild integration only works with one MSBuild process. When you use the touchless MSBuild integration, do not build your projects in parallel mode by specifying the MSBuild `/m` (or `/maxcpucount`) option.

The following is an example of the command to run the build and a Fortify Static Code Analyzer analysis:

```
sourceanalyzer -b buildxyz msbuild <solution_file> <msbuild_options>
```

Alternatively, you can call MSBuild to run a build and a Fortify Static Code Analyzer analysis:

```
Msbuild <solution_file> /logger:"C:<sca_install_
dir>\Core\lib\FortifyMSBuildTouchless.dll" <msbuild_options>
```

Note: When you build solutions with Xamarin.iOS projects using touchless MSBuild integration, Fortify recommends that you explicitly specify `iPhone` or `iPhoneSimulator` as the solution's target platform with the MSBuild `/p` command-line option. This is because the default platform selected by MSBuild might result in the exclusion of Xamarin.iOS projects from the build, which leads to the loss of translation results for these projects. For example, the *<msbuild_options>* in either of the previous command-line examples should include one of the following: `/p:Platform=iPhone` or `/p:Platform=iPhoneSimulator`.

Adding Custom Tasks to your MSBuild Project

As an alternative to using the touchless MSBuild integration method, you can add custom tasks to your MSBuild project to invoke Fortify Static Code Analyzer.

Note: MSBuild custom tasks are only supported with .NET projects.

To add a custom task to an MSBuild project:

1. Create a task to identify and locate the `FortifyMSBuildTasks.dll`:

   ```
   <UsingTask TaskName="Fortify.TranslateTask" AssemblyFile="<sca_install_
   dir>\Core\lib\FortifyMSBuildTasks.dll" />
   ```

2. Create a new target or add a custom target to an existing target to invoke the custom task.

The following sections describe the parameters and provide examples for the different custom tasks that you can add to an MSBuild project:

- "Fortify.CleanTask" on the next page - Remove any existing temporary files for the specified build ID
- "Fortify.TranslateTask" on the next page - Translate the source code
- "Fortify.ScanTask" on page 80 - Scan the source files
- "Fortify.SSCTask" on page 81 - Upload an FPR to Fortify Software Security Center
- "Fortify.CloudScanTask" on page 82 - Upload an FPR to Fortify CloudScan or Fortify Software Security Center

Fortify.CleanTask

The following table describes the parameters for the `Fortify.CleanTask`.

Parameter	Description
Required Parameters	
BuildID	Build ID
Optional Parameters	
Debug	Whether to enable debug mode for the task and Fortify Static Code Analyzer

The following example shows the `Fortify.CleanTask` task in an MSBuild project:

```
<UsingTask TaskName="Fortify.CleanTask"
  AssemblyFile="<sca_install_dir>\Core\lib\FortifyMSBuildTasks.dll" />

<Target Name="FortifyBuild" AfterTargets="AfterBuild"
  Outputs="dummy.out">
  <CleanTask BuildID="MyProj" />
</Target>
```

Fortify.TranslateTask

You must customize these tasks to each sub-project of your MSBuild project so that each sub-project is translated separately. This is similar to manual translation described in "Manual .NET Command-Line Syntax" on page 30.

The following table describes the parameters for the `Fortify.TranslateTask`.

Parameter	Description
Required Parameters	
BuildID	The build ID for the translation
TargetsFolder	Directory where the source code or binary files to be translated reside
Include at least one of the following six parameters. **Note:** Do not combine the DotNetVersion, DotNetCoreVersion, DotNetStandardVersion, XamarinAndroidVersion, and XamariniOSVersion parameters within the same task.	
DotNetVersion	The .NET framework version
DotNetCoreVersion	The .NET Core version

Parameter	Description
DotNetStandardVersion	The .NET Standard version
XamarinAndroidVersion	The target Android SDK version for Xamarin Android projects
XamariniOSVersion	The target iOS SDK version for Xamarin iOS projects
References	Semicolon-separated list of directories where referenced system or third-party DLLs are located (you can also specify paths to specific DLLs with this option)
Optional Parameters	
DotNetSharedFiles	Semicolon-separated list of of the source files for all Shared Projects included in the project you are translating
DotNetOutputDir	Output directory where the binary (EXE or DLL) built from the project is placed
ReferencesOnly	Sets the list of directories or paths to only those specified by the References parameter
NugetCacheDir	**(.NET Core and .NET Standard projects only)** A custom location for the NuGet cache directory
WebRootFolder	**(.NET Web projects only)** Home directory of an ASP.NET project
AspNetPages	**(.NET Web projects only)** Semicolon-separated list of paths to an ASP.NET pages **Important!** Use this parameter only to specify ASP.NET pages that belong to the current sub-project but are located outside of the folder specified by the TargetsFolder parameter.
DotNetWebsite	**(.NET Web projects only)** Whether the project is an ASP.NET Website (Do not use this parameter for ASP.NET Web Applications or any other kind of ASP.NET project except Website.)
DotNetAppLibs	**(.NET Web projects only)** Semicolon-separated list of additional reference DLLs needed to translate ASP.NET pages (typically, these are the assemblies built from the source files that belong to the same sub-project as the target ASP.NET pages)
DotNetCodeBehind	**(.NET Web projects only)** Semicolon-separated list of source files that are bound to ASP.NET pages (referred to as *code-behind files*)
AspNetCore	**(.NET Web projects only)** Whether the web project (ASP.NET or ASP.NET Core) targets the .NET Core or .NET Standard framework.
AssemblyName	Name of the target .NET assembly as specified in Visual Studio project settings

Parameter	Description
PreprocessorSymbols	Semicolon-separated list of preprocessor symbols used in the source code
Exclude	Semicolon-separated list of folders and specific files to skip during the translation (You can use the asterisk (*) wild card character such as *.dll in this parameter's value.)
JVMSettings	Memory settings to pass to Fortify Static Code Analyzer
LogFile	Location of the Fortify Static Code Analyzer log file
Debug	Whether to enable debug mode for the task and Fortify Static Code Analyzer
ExternalAliases	**(C# projects only)** Semicolon-separated list of external aliases for a specified DLL file
VBCompileOptions	**(VB.NET projects only)** Any special compilation options required for the correct translation of the source code
VBImports	**(VB.NET projects only)** Semicolon-separated list of namespaces imported for all source files in the project
VBMyType	**(VB.NET projects only)** Value for the `_MYTYPE` preprocessor symbol specified in the `<MyType>` tag in the project settings
VBRootFolder	**(VB.NET projects only)** Root namespace for the project as specified in the Visual Studio project settings

The following example shows the `Fortify.TranslateTask` task in an MSBuild project:

```
<UsingTask TaskName="Fortify.TranslateTask"
  AssemblyFile="<sca_install_dir>\Core\lib\FortifyMSBuildTasks.dll" />

<Target Name="FortifyBuild" AfterTargets="AfterBuild" Outputs="dummy.out">

  <TranslateTask
    TargetsFolder="$(OutDir)"
    BuildID="MyProj"
    DotNetVersion="4.6"
    JVMSettings="-Xmx2000M"
    LogFile="trans_task.log"
    Debug="true" />
</Target>
```

The `FortifyBuild` target is invoked after the `AfterBuild` target is run. The `AfterBuild` target is one of several default targets defined in the MSBuild target file. You must specify all required parameters.

Fortify.ScanTask

The following table describes the parameters for the `Fortify.ScanTask`.

Parameter	Description
Required Parameters	
BuildID	Build ID for the scan
Output	Name of the FPR file to generate
Optional Parameters	
JVMSettings	Memory settings to pass to Fortify Static Code Analyzer
LogFile	Location of the Fortify Static Code Analyzer log file
Debug	Whether to enable debug mode for the task and Fortify Static Code Analyzer

The following example shows a `Fortify.TranslateTask` and a `Fortify.ScanTask` task in an MSBuild project:

```
<UsingTask TaskName="Fortify.TranslateTask"
  AssemblyFile="<sca_install_dir>\Core\lib\FortifyMSBuildTasks.dll" />

<UsingTask TaskName="Fortify.ScanTask"
  AssemblyFile="<sca_install_dir>\Core\lib\FortifyMSBuildTasks.dll" />

<Target Name="FortifyBuild" AfterTargets="AfterBuild" Outputs="dummy.out">
  <TranslateTask
    TargetsFolder="$(OutDir)"
    DotNetVersion="4.6"
    BuildID="MyProj"
    JVMSettings="-Xmx2000M"
    LogFile="trans_task.log"
    Debug="true" />

  <ScanTask
    BuildID="MyProj"
    JVMSettings="-Xmx2000M"
    LogFile="scan_task.log"
    Debug="true"
    Output="MyProj.fpr" />
</Target>
```

Fortify.SSCTask

The following table describes the parameters for the `Fortify.ScanTask`.

Parameter	Description
Required Parameters	
AuthToken	Authentication token (if not included, you must specify the username and password parameters)
Project	Fortify Software Security Center application name (if not provided, you must include the ProjectID and ProjectVersionID parameters)
ProjectVersion	Fortify Software Security Center application version (if not provided, you must include the ProjectID and ProjectVersionID parameters)
FPRFile	Name of the file to upload to Fortify Software Security Center
SSCURL	URL for Fortify Software Security Center
Optional Parameters	
Debug	Whether to enable debug mode for the task and Fortify Static Code Analyzer
Username	Include a username if the AuthToken parameter is not specified
Password	Include a password if AuthToken parameter is not specified
ProjectID	Include with ProjectVersionID if the Project and ProjectVersion parameters are not specified
ProjectVersionID	Include with ProjectID if Project and ProjectVersion parameters are not specified
Proxy	Include if a proxy is required

The following example shows the `Fortify.TranslateTask`, `Fortify.ScanTask`, and `Fortify.SSCTask` tasks in an MSBuild project:

```
<UsingTask TaskName="Fortify.TranslateTask"
  AssemblyFile="<sca_install_dir>\Core\lib\FortifyMSBuildTasks.dll" />

<UsingTask TaskName="Fortify.ScanTask"
  AssemblyFile="<sca_install_dir>\Core\lib\FortifyMSBuildTasks.dll" />

<UsingTask TaskName="Fortify.SSCTask"
  AssemblyFile=<sca_install_dir>\Core\lib\FortifyMSBuildTasks.dll" />

<Target Name="FortifyBuild" AfterTargets="AfterBuild" Outputs="dummy.out">
```

```
  <TranslateTask
    TargetsFolder="$(OutDir)"
    DotNetVersion="4.6"
    BuildID="MyProj"
    JVMSettings="-Xmx1000M"
    LogFile="trans_task.log"
    Debug="true" />

  <ScanTask BuildID="TestTask"
    JVMSettings="-Xmx1000M"
    LogFile="scan_task.log"
    Debug="true"
    Output="MyProj.fpr" />

  <SSCTask
    Username="admin"
    Password="admin"
    Project="Test Project"
    ProjectVersion="Test Version 1"
    FPRFile="MyProjSSC.fpr"
    SSCURL="http://localhost:8180/ssc" />
</Target>
```

Fortify.CloudScanTask

If you are using Micro Focus Fortify CloudScan to process your scans, you can send the translated output to your Fortify CloudScan environment. You can also use this task to upload to Fortify Software Security Center.

The following table describes the parameters for the `Fortify.CloudScanTask`.

Parameter	Description
Required Parameters	
BuildID	Build ID for the translation
SSCUpload	Whether to upload output to Fortify Software Security Center
CloudURL	Cloud URL (include either this parameter or the SSCURL parameter)
SSCURL	URL for Fortify Software Security Center (include either this parameter or the CloudURL parameter)
Optional Parameters	

Debug	Whether to enable debug mode for the task and Fortify Static Code Analyzer
SSCToken	**(Use only when SSCUpload is true)** Fortify Software Security Center token
Project	**(Use only when SSCUpload is true)** Target application on Fortify Software Security Center
VersionName	**(Use only when SSCUpload is true)** Target application version on Fortify Software Security Center
FPRName	**(Use only when SSCUpload is false)** Name for the FPR file

The following example shows the `Fortify.CloudScanTask` task in an MSBuild project:

```
<UsingTask TaskName="Fortify.CloudScanTask"
  AssemblyFile="<sca_install_dir>\Core\lib\FortifyMSBuildTasks.dll"/>

<Target Name="FortifyBuild" AfterTargets="AfterBuild" Outputs="dummy.out">
  <CloudScanTask
    BuildID="MyProj"
    SSCUpload="false"
    FPRName="MyProj.fpr"
    CloudURL="http://localhost:8080/cloud-ctrl" />
</Target>
```

Chapter 15: 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:

Output Options 84
Translation Options 86
Analysis Options 87
Other Options 90
Directives 91
Specifying Files 92

Output Options

The following table describes the output options.

Output Option	Description
`-f <file>`	Specifies the file to which results are written. If you do not specify an output file, the output is written to the terminal. **Equivalent property name:** `com.fortify.sca.ResultsFile`
`-format <format>`	Controls the output format. Valid options are `fpr`, `fvdl`, `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. **Note:** If you use result certification, you must specify the `fpr` format. See the *Micro Focus Fortify Audit Workbench User Guide* for information on result certification. **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.

Output Option	Description
	The engine data, which includes Rulepack 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 not possible to analyze an entire application at once. **Equivalent property name:** `com.fortify.sca.OutputAppend`
`-disable-source-bundling`	Excludes source files from the FPR file. **Equivalent property name:** `com.fortify.sca.FPRDisableSourceBundling`
`-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`

Translation Options

The following table describes the translation options.

Translation Option	Description
`-b <build_id>`	Specifies the build ID. Fortify Static Code Analyzer uses the build ID to track which files are compiled and combined as part of a build, and later, to scan those files. **Equivalent property name:** `com.fortify.sca.BuildID`

`-exclude <file_specifiers>`	Removes files from the list of files to translate. See "Specifying Files" on page 92 for more information on how to use file specifiers.
	For example:
	`sourceanalyzer -cp "**/*.jar" "**/*" -exclude "**/Test.java"`
	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 `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`. **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.

Translation Option	Description
`-noextension-type <file_type>`	Specifies the file type for source files that have no file extension. The possible values are: ABAP, ACTIONSCRIPT, APEX, APEX_TRIGGER, ARCHIVE, ASPNET, ASP, ASPX, BITCODE, BYTECODE, CFML, COBOL, CSHARP, HTML, JAVA, JAVA_PROPERTIES, JAVASCRIPT, JSP, JSPX, MSIL, MXML, PHP, PLSQL, PYTHON, RUBY, RUBY_ERB, SCALA, SWIFT, TLD, SQL, TSQL, VB, VB6, VBSCRIPT, VISUAL_FORCE, and XML.

Analysis Options

The following table describes the analysis options.

Analysis Option	Description
`-scan`	Causes Fortify Static Code Analyzer to perform analysis for the specified build ID.
`-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`, `findbugs`, `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`
`-b <build_id>`	Specifies the build ID. **Equivalent property name:** `com.fortify.sca.BuildID`
`-quick`	Scans the project in Quick Scan mode, using the `fortify-sca-quickscan.properties` file. By default, this scan searches for high-confidence, high-severity issues that Fortify Static Code Analyzer can discover quickly. **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:** `com.fortify.sca.BinaryName`

Analysis Option	Description
`-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:Controlflow`, `Characterization:Issue`, and `Characterization:Generic`. The *<type>* parameter is case-insensitive.
`-exit-code-level`	Extends the default exit code options. See "Exit Codes" on page 108 for a description of the exit codes. The valid values are: The valid values are: • `nothing`—Returns exit codes 0, 1, 2, or 3. This is the default setting. • `warnings`—Returns exit codes 0, 1, 2, 3, 4, or 5. • `errors`—Returns exit codes 0, 1, 2, 3, or 5. • `no_output_file`—Returns exit codes 0, 1, 2, 3, or 6. **Equivalent property name:** `com.fortify.sca.ExitCodeLevel`
`-filter <file>`	Specifies a results filter file. **Equivalent property name:** `com.fortify.sca.FilterFile`
`-findbugs`	Enables FindBugs analysis for Java code. You must specify the Java class directories with the `-java-build-dir` option, which is described in "Java Command-Line Options" on page 23. **Equivalent property name:** `com.fortify.sca.EnableFindbugs`
`-incremental-base`	Specifies that this is the initial full scan of a project for which you plan to run subsequent incremental scans. Use this option for the first scan when you plan to run subsequent scans on the same project with the `-incremental` option. See "Incremental Analysis" on page 20 for more information about performing incremental analysis. **Equivalent property name:** `com.fortify.sca.IncrementalBaseScan`
`-incremental`	Specifies that this is a subsequent scan of a project for which

Analysis Option	Description
	you have already run a full base scan with the `-incremental-base` option. See "Incremental Analysis" on page 20 for more information about performing incremental analysis. **Equivalent property name:** `com.fortify.sca.IncrementalScan`
`-no-default-issue-rules`	Disables rules in default Rulepacks that lead directly to issues. 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`
`-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 Fortify Software Security Center server, 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, includes all of the files in the directory with the `.bin` and `.xml` extensions.

Analysis Option	Description
	Equivalent property name: `com.fortify.sca.RulesFile`

Other Options

The following table describes other options.

Other Option	Description
`@<filename>`	Reads command-line options from the specified file.
`-h | -? | -help`	Prints a summary of command-line options.
`-debug`	Includes debug information in the log file, which is useful for troubleshooting. **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`
`-logfile <file>`	Specifies the log file that Fortify Static Code Analyzer creates. **Equivalent property name:** `com.fortify.sca.LogFile`
`-clobber-log`	Directs Fortify Static Code Analyzer to overwrite the log file for each new scan. **Equivalent property name:** `com.fortify.sca.ClobberLogFile`
`-quiet`	Disables the command-line progress information. **Equivalent property name:** `com.fortify.sca.Quiet`
`-verbose`	Sends verbose status messages to the console and to the log file. **Equivalent property name:** `com.fortify.sca.Verbose`
`-version`	Displays the Fortify Static Code Analyzer version number.

Other Option	Description
`-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. 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, assuming no other memory intensive processes are running, do not allocate more than 2/3 of the available memory.

Directives

Use the following directives to list information about previous translation commands. Use only one directive at a time and do not use any directive in conjunction with normal translation or analysis commands.

Directive	Description
`-clean`	Deletes all Fortify Static Code Analyzer intermediate files and build records. If a build ID is specified, only files and build records relating to that build ID are deleted.
`-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 of 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:** Audit Workbench also displays these errors and warnings in the results certification panel.
`-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.

Specifying Files

File specifiers are expressions that allow you to pass a long list of files to Fortify Static Code Analyzer using wild card characters. Fortify Static Code Analyzer recognizes two types of wild card 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 specifiers>

The following table describes the different file specifier formats.

Note: In the following table, the `.java` extension is only used as an example to show the different file specifier options.

File Specifier	Description
<dir>	All files found in the named directory or any sub-directories.
<dir>`/**/Example.java`	Any file named `Example.java` found in the named directory or any sub-directories.
<dir>`/*.java`	Any file with the extension `.java` found in the named directory.
<dir>`/**/*.java`	Any file with the extension `.java` found in the named directory or any sub-directories.
<dir>`/**/*`	All files found in the named directory or any sub-directories (same as *<dirname>*).

Note: Windows and many Unix 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 (/).

File specifiers do not apply to C, C++, or Objective-C++ languages.

Chapter 16: Command-Line Utilities

This section contains the following topics:

Fortify Static Code Analyzer Utilities 93
Checking the Fortify Static Code Analyzer Scan Status 94
Working with FPR Files from the Command Line 96
Generating Reports from the Command Line 103
About Updating Security Content 106

Fortify Static Code Analyzer Utilities

Fortify Static Code Analyzer command-line utilities enable you to manage Rulepacks and FPR files, run reports, and perform other system-level processes. 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.

Utility	Description	For More Information
SCAState	Provides state analysis information on the JVM during the scan phase	"Checking the Fortify Static Code Analyzer Scan Status" on the next page
FPRUtility	With this utility you can: • Merge audited projects • Verify FPR signatures • Display mappings for a migrated project • Display any errors associated with an FPR • Display the number of issues in an FPR • Display filtered lists of issues in different formats • Display table of analyzed functions • Combine or split source code files and audit projects into FPR files	"Working with FPR Files from the Command Line" on page 96
BIRTReportGenerator and ReportGenerator	Generates BIRT reports and legacy reports from FPR files	"Generating Reports from the Command Line" on page 103

Utility	Description	For More Information
`fortifyupdate`	Compares installed security content to the most current version and makes any required updates	"About Updating Security Content" on page 106

Other Command-Line Utilities

In addition to the Fortify Static Code Analyzer command-line utilities described in this guide, Fortify provides the command-line utilities described in the following table.

Utility	Description	For More Information
`fortifyclient`	Use this utility to create Fortify Software Security Center authentication tokens and securely transfer objects to and from Fortify Software Security Center.	See the *Micro Focus Fortify Software Security Center User Guide*
`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.	See the *Micro Focus Fortify Static Code Analyzer Installation Guide*

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.

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: 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 or Unix 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).

Option	Description
`-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\<userID>\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.

Working with FPR Files from the Command Line

Use the FPRUtility that is 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 98
- "Extracting a Source Archive from an FPR File" on page 101
- "Allocating More Memory for FPRUtility" on page 102

Merging FPR Files

The FPRUtility `-merge` option combines the analysis information from two FPR files into a single FPR file using the values of the primary project to resolve conflicts.

To merge FPR files:

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

To merge FPR files and set instance ID migrator options:

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

FPRUtility Data Merge Options

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

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 <output>.fpr`	Specifies the name of the merged output file. This file is 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 the engine and the Rulepack versions of the two projects are the same.
`-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.
`-useSourceIssueTemplate`	Specifies to use the filter sets and folders from the issue template in the secondary FPR. By default, Fortify Static Code Analyzer uses the filter sets and folders from the issue template in the primary FPR.

Option	Description
`-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 output.fpr -iidmigratorOptions "-u scheme_file"`

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.

To display project signature information:

```
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] \
[-includeSupressed] [-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 `-includeSupressed` 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
```

FPRUtility Information Options

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

Option	Description
`-information`	Displays information for the project.

One of:
`-signature`
`-mappings`
`-errors`
`-versions`
`-functionsMeta`
`-categoryIssueCounts`
`-analyzerIssueCounts`
`-search -query <search_expression>`
`-search -queryAll`
	The `-signature` option displays the signature.

	The `-mappings` option displays the migration mappings report.
	The `-errors` option displays a full error report for the FPR.
	The `-versions` option displays the engine version and the Rulepack version used in the static scan.
	The `-functionsMeta` option displays all functions that the static analyzer encountered in CSV format. To filter which functions are displayed, include `-excludeCoveredByRules`, and `-excludeFunctionsWithoutSource`.

Option	Description
	The `-categoryIssueCounts` option displays the number of issues for each vulnerability category.
	The `-analyzerIssueCounts` option displays the number of issues for each analyzer.
	The `-search -query` option displays 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.
	The `-search -queryAll` searches all the issues in the FPR, including suppressed and removed issues.
`-project <project>.fpr`	Specifies the FPR from which to extract the results information.
`-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`.
`-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 `-analyzerIssueCounts`, then the output includes (`<category>`). If you specify the `-outputFormat CSV`, then each issue is displayed as a line in the format: `"<instanceid>", "<category>", "<sink_filename>:<line_num>", "<analyzer>"`

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 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
<outputArchive>.fsa
```

To archive data to a folder:

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

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.

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).
	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 folder 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_folder>`	Specifies the output file. You can generate an FPR, a folder, or an FSA file.

Allocating More Memory for FPRUtility

Performing tasks with large and complex FPR files could 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. BIRT is an open source reporting system.
- ReportGenerator—Generates legacy reports from FPR files from the command line. You can specify a report template, otherwise the 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_
project>.fpr -format PDF|DOC|HTML|XLS -output <report_file>
```

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

```
BIRTReportGenerator -template "OWASP Top 10" -source auditedProject.fpr
-format PDF -showSuppressed --Version "OWASP Top 10 2017"
--UseFortifyPriorityOrder -output MyOWASP_Top10_Report.pdf
```

BIRTReportGenerator Command-Line Options

The following table lists the BIRTReportGenerator options.

Option	Description
`-template <template_name>`	Specifies the report template name. The valid values are: `"Developer Workbook"`, `"DISA STIG"`, `"CWE/SANS Top 25"`, `"FISMA Compliance"`, `"OWASP Mobile Top 10"`, `"OWASP Top 10"`, and `"PCI DSS Compliance"`.
`-source <audited_project>.fpr`	Specifies the audited project on which to base the report.
`-format <format>`	Specifies the generated report format. The valid values for *<format>* are: `PDF`, `DOC`, `HTML`, and `XLS`.
`-output <report_file.***>`	Specifies the file to which the report is written.
`-searchQuery <query>`	Specifies a search query to filter issues before generating the report.
`-showSuppressed`	Include issues that are marked as suppressed.

Option	Description
`-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 templates versions are listed below.
	Notes:
	• The first version listed for each template is the default.
	• Some templates only have one version available and therefore do not require inclusion of this option.
	For the "CWE/SANS Top 25" template:
	`"2011 CWE/SANS Top 25"`, `"2010 CWE/SANS Top 25"`, and `"2009 CWE/SANS Top 25"`
	For the "DISA STIG" template:
	`"DISA STIG 4.4"`, `"DISA STIG 4.3"`, `"DISA STIG 4.2"`, `"DISA STIG 4.1"`, `"DISA STIG 3.10"`, `"DISA STIG 3.9"`, `"DISA STIG 3.7"`, `"DISA STIG 3.5"`, and `"DISA STIG 3.4"`
	For the "OWASP Top 10" template:
	`"OWASP Top 10 2017"`, `"OWASP Top 10 2013"`, `"OWASP Top 10 2010"`, `"OWASP Top 10 2007"`, and `"OWASP Top 10 2004"`
	For the "PCI DSS Compliance" template:
	`"3.2 Compliance"`, `"3.1 Compliance"`, `"3.0 Compliance"` and `"2.0 Compliance"`

`--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.

Option	Description
`--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.

Generating a Legacy Report

To generate a PDF report, type the following command:

```
ReportGenerator -format pdf -f <results_file>.pdf -source <audited_
project>.fpr
```

To generate an XML report, type the following command:

```
ReportGenerator -format XML -f <results_file>.xml -source <audited_
project>.fpr
```

ReportGenerator Command-Line Options

The following table lists the ReportGenerator options.

Option	Description
`-format <format>`	Specifies the generated report format. The valid values for `<format>` are: `PDF`, `RTF`, and `XML`.
`-f <resultsfile.***>`	Specifies the file to which the report is written.
`-source <audited_project>.fpr`	Specifies the audited project on which to base the report.
`-template <template_name>`	Specifies the issue template used to define the report. If not specified, ReportGenerator uses the default template.
`-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.

About Updating Security Content

You can use the `fortifyupdate` utility to download the latest Fortify Secure Coding Rulepacks and metadata from the Fortify Customer Portal for your installation.

The `fortifyupdate` utility gathers information about the existing security content in your Fortify installation and contacts the 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 Customer Portal, type the following command:

```
fortifyupdate [<options>]
```

fortifyupdate Command-Line Options

The following table lists the `fortifyupdate` options.

Option	Description
`-acceptKey`	Accept the public key. When this is specified, you are not prompted to provide a public key.
`-acceptSSLCertificate`	Use the SSL certificate provided by the server.
`-coreDir <dir>`	Specifies the core directory where the update is stored.
`-import <file>.zip`	Imports the ZIP file that contains archived security content.
`-includeMetadata`	Specifies to only update external metadata.
`-includeRules`	Specifies to only update Rulepacks.
`-locale <locale>`	Specifies a locale. The default is the value set for the locale property in the `fortify.properties` configuration file. For more information about the `fortify.properties` configuration file, see the *Micro Focus Fortify Static Code Analyzer Tools Properties Reference Guide*.
`-proxyhost <host>`	Specifies a proxy server network name or IP address.

Option	Description
`-proxyport <port>`	Specifies a proxy server port number.
`-proxyUsername <username>`	If the proxy server requires authentication, specifies the user name.
`-proxyPassword <password>`	If the proxy server requires authentication, specifies the password.
`-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*.

Chapter 17: Troubleshooting

This section contains the following topics:

Exit Codes 108
Using the Log File to Debug Problems 109
Translation Failed Message 109
Issue Non-Determinism 109
JSP Translation Problems 110
C/C++ Precompiled Header Files 110
Reporting Issues and Requesting Enhancements 111

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

By default, Fortify Static Code Analyzer only returns exit codes 0, 1, 2, or 3.

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.

Note: The equivalent command-line option is `-exit-code-level`.

The valid values are:

- `nothing`—Returns exit codes 0, 1, 2, or 3. This is the default setting.
- `warnings`—Returns exit codes 0, 1, 2, 3, 4, or 5.
- `errors`—Returns exit codes 0, 1, 2, 3, or 5.
- `no_output_file`—Returns exit codes 0, 1, 2, 3, or 6.

Using the Log File to Debug Problems

If you encounter warnings or errors when you run Fortify Static Code Analyzer, re-run Fortify Static Code Analyzer using the `-debug` option. This generates a file named `sca.log` in the following directory:

- On Windows: `C:\Users\<user>\AppData\Local\Fortify\sca<version>\log`
- On other platforms: `$HOME/.fortify/sca<version>/log`

where *<version>* is the version of Fortify Static Code Analyzer that you are using.

Email the `sca.log` file as a ZIP file to Micro Focus Fortify Customer Support.

Translation Failed Message

If your C or C++ application builds successfully but you see one or more "translation failed" messages during the Fortify Static Code Analyzer translation, edit the *<sca_install_dir>*`/Core/config/fortify-sca.properties` file to change the following line:

```
com.fortify.sca.cpfe.options= --remove_unneeded_entities --suppress_vtbl
```

to:

```
com.fortify.sca.cpfe.options= -w --remove_unneeded_entities --suppress_
vtbl
```

Re-run the translation to print the errors that the translator encountered. If the output indicates an incompatibility between your compiler and the Fortify Static Code Analyzer translator, send your output to Micro Focus Fortify Customer Support for further investigation.

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 should provide 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
```

JSP Translation Problems

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
- You are missing some JAR files or classes required for the application
- You are missing some tag libraries or their definitions (TLD) for the application

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 strings `Jsp parser stdout:` and `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 27.

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.

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://softwaresupport.softwaregrp.com.

Include the following information when you contact customer support:

- Product: Fortify Static Code Analyzer
- Version number: To determine the version number, 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 112
Filter File Example 112

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 this filter files for standard audits, because auditors should 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. The file functions as a blacklist, where only the filter items you *do not* want are specified. 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 scans have taken place, whereas an instance ID filter is applied after the analysis phase.

Filter File Example

As an example, the following output resulted 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 seven detected issues:

```
[F7A138CDE5235351F6A4405BA4AD7C53 : low : Unchecked Return Value :
semantic ]
EightBall.java(12) : Reader.read()
```

```
[63C4F599F304C400E4BB77AB3EF062F6 : 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)

[EFE997D3683DC384056FA40F6C7BD0E8 : critical : Path Manipulation :
dataflow ]
EightBall.java(12) :  ->new FileReader(0)
  EightBall.java(6) : <=> (filename)
  EightBall.java(4) :  ->EightBall.main(0)

[60AC727CCEEDE041DE984E7CE6836177 : high : Unreleased Resource : Streams :
controlflow ]
  EightBall.java(12) : start -> loaded : new FileReader(...)
  EightBall.java(12) : loaded -> loaded : <inline expression> refers to an
allocated resource
  EightBall.java(12) : java.io.IOException thrown
  EightBall.java(12) : loaded -> loaded : throw
  EightBall.java(12) : loaded -> loaded : <inline expression> no longer
refers to an allocated resource
  EightBall.java(12) : loaded -> end_of_scope : end scope : Resource
leaked : java.io.IOException thrown
  EightBall.java(12) : start -> loaded : new FileReader(...)
  EightBall.java(12) : loaded -> loaded : <inline expression> refers to an
allocated resource
  EightBall.java(14) : loaded -> loaded : <inline expression> no longer
refers to an allocated resource
  EightBall.java(14) : loaded -> end_of_scope : end scope : Resource
leaked

[BB9F74FFA0FF75C9921D0093A0665BEB : low : J2EE Bad Practices : Leftover
Debug Code : structural ]
  EightBall.java(4)

[FF0D787110C7AD2F3ACFA5BEB6E951C3 : low : Poor Logging Practice : Use of a
System Output Stream : structural ]
  EightBall.java(10)

[FF0D787110C7AD2F3ACFA5BEB6E951C4 : low : Poor Logging Practice : Use of a
System Output Stream : structural ]
  EightBall.java(13)
```

The following is example filter file content that will perform the following:

- Remove all results related to the Poor Logging Practice category
- Remove the Unreleased Resource based on its instance ID
- Remove any dataflow issues that were generated from a specific rule ID

```
#This is a category that will be filtered from scan output
Poor Logging Practice

#This is an instance ID of a specific issue to be filtered from scan
output
60AC727CCEEDE041DE984E7CE6836177

#This is a specific Rule ID that leads to the reporting of a specific
issue in
#the scan output: in this case the data flow 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:

```
[F7A138CDE5235351F6A4405BA4AD7C53 : low : Unchecked Return Value :
semantic]
  EightBall.java(12) : Reader.read()

[63C4F599F304C400E4BB77AB3EF062F6 : 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)

[BB9F74FFA0FF75C9921D0093A0665BEB : low : J2EE Bad Practices : Leftover
Debug Code : structural]
EightBall.java(4)
```

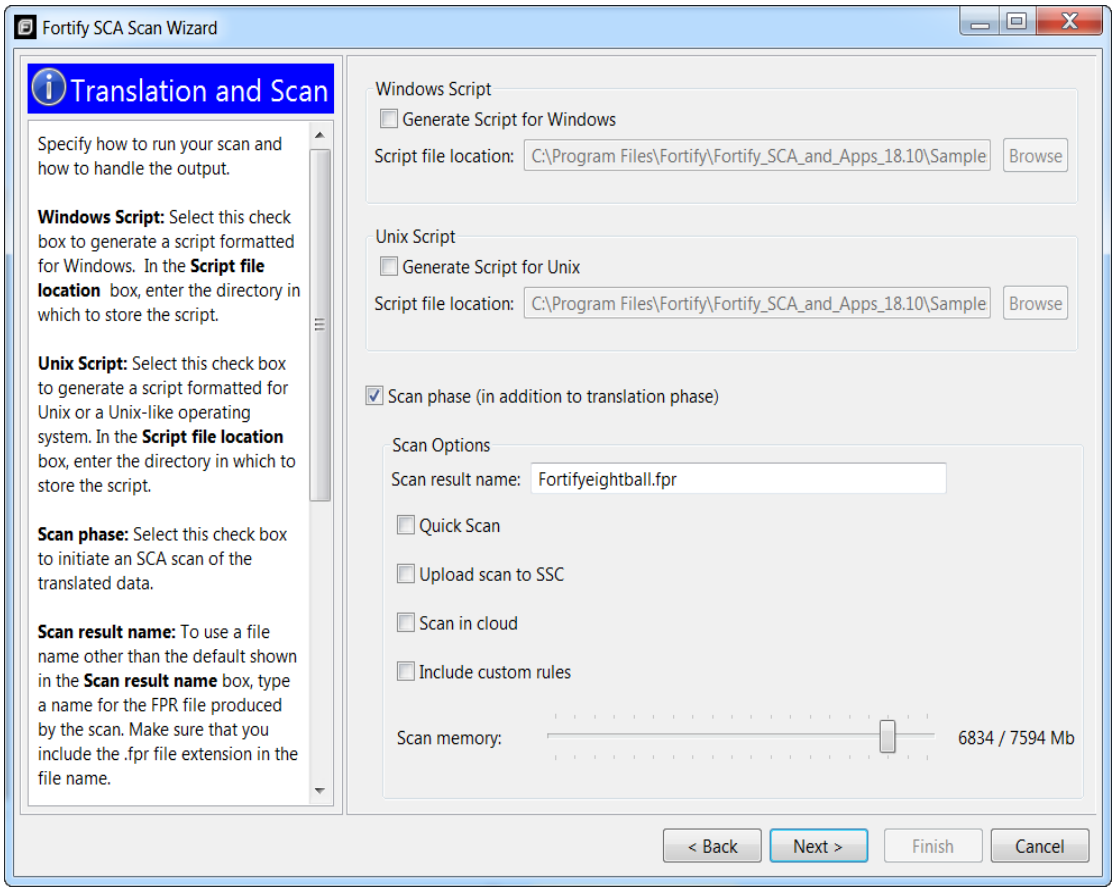
Appendix B: Scan Wizard

This section contains the following topics:

Preparing to use the Scan Wizard 115
Starting the Scan Wizard 116

Preparing to use the Scan Wizard

Scan Wizard uses the information you provide to create a script with the commands for Fortify Static Code Analyzer to translate and scan project code and optionally upload the results directly to Fortify Software Security Center. You can use Scan Wizard to run your scans locally or upload them to a Fortify CloudScan server.



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

To use the Scan Wizard, you need the following:

- Location of the build directory or directories of the project to be scanned
- Access to the build directory or directories of the project to be scanned
- To scan Java code, the version of the Java JDK used to develop the code
- To use Fortify CloudScan to scan your code, the URL of the CloudScan Controller
- (Optional) Location of custom rule files

To upload your scan results to Fortify Software Security Center, you also need:

- Your Fortify Software Security Center logon credentials
- The Fortify Software Security Center server URL
- An upload authentication token

> **Note:** If you do not have an upload token, you can use the Scan Wizard to generate one. To do this, you must have Fortify Software Security Center logon credentials.

If you do not have Fortify Software Security Center logon credentials, you must have the following:

- Application name
- Application version name

> **Note:** Scan Wizard uses a default scan memory setting of 90% of the total available memory if it is greater than 4 GB, otherwise the default memory setting is 2/3 the total available memory. Adjust the scan memory as necessary in the **Translation and Scan** step.

Starting the Scan Wizard

How you start the Scan Wizard depends on whether you installed Fortify SCA and Applications locally or if you are using the Scan Wizard as a stand-alone utility on your system.

Starting Scan Wizard on a System with Fortify SCA and Applications Installed

To start the 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**.
- On Linux, navigate to the *<sca_install_dir>*`/bin` directory, and then run the `ScanWizard` file from the command line.
- On macOS, navigate to the *<sca_install_dir>*`/bin` directory, and then double-click `ScanWizard`.

Starting Scan Wizard as a Stand-Alone Utility

To start the Scan Wizard as a stand-alone utility:

1. Download and uncompress the Scan Wizard package (`Fortify_Scan_Wizard_<version>_<OS_platform>.zip`) for your operating system.
2. Navigate to the `Fortify-<version>-Scan-Wizard/bin` directory.
3. Do one of the following:
   - On Windows, double-click the `ScanWizard.cmd` file.
   - On Linux, run the `ScanWizard` file.
   - On macOS, double-click the `ScanWizard` file.

Appendix C: Sample Files

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

<sca_install_dir>/Samples

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

The basic sub-directory 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 Audit Workbench, see *Micro Focus Fortify Audit Workbench User Guide*.

This section contains the following topics:

Basic Samples 118
Advanced Samples 120

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. It 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 J2EE Bad Practices: Leftover Debug Code
formatstring	The `formatstring.c` file. It requires a gcc or cl compiler.	Format String
javascript	The `sample.js` JavaScript file.	Cross Site Scripting (XSS) Open Redirect
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 the WebGoat project located in the `Samples/advanced/webgoat` directory.	Example input for Audit Workbench.
stackbuffer	The `stackbuffer.c` file. It requires a gcc or cl compiler.	Buffer Overflow
toctou	The `toctou.c` file.	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 sample files in the *<sca_install_dir>*/Samples/advanced directory. Many of the samples include a README.txt file that provides further details and instructions on its use.

Folder Name	Description
BugTrackerPlugin *<bugtracker>*	Includes source code for the supported bug tracker plugin.
c++	A sample solution for different 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 and the package 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, struts). The output contains several issues of different types, including two 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 different supported versions of Visual Studio. Scanning this sample reveals the SQL Injection vulnerabilities and an Unreleased Resource vulnerability. Other categories might also be present, depending on the Rulepacks 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.
findbugs	A sample that demonstrates how to run the FindBugs static analysis tool together with Fortify Static Code Analyzer and filter out results that overlap.
java1.5	A sample Java file: ResourceInjection.java. The result file should include a Path Manipulation, a J2EE Bad Practices, and a Poor Style vulnerability.

Folder Name	Description
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 example illustrates how the use of Fortify Annotations can result in increased accuracy in the reported vulnerabilities. The accompanying `Java Annotations Sample.txt` file describes the potential problems and solutions associated with vulnerability results.
JavaDoc	JavaDoc directory for the `public-api` and `WSClient`.
riches.java	A Java EE 1.4 sample web application with various known security vulnerabilities including Cross-Site Scripting, SQL Injection, and Command Injection.
riches.net	A .NET 4.0 sample web application with various known security vulnerabilities including Cross-Site Scripting, SQL Injection, and Command Injection.
webgoat	The WebGoat test Java EE web application provided by the Open Web Application Security Project (https://www.owasp.org). This directory contains the WebGoat 5.0 source code.

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 122
fortify-sca.properties 123
fortify-sca-quickscan.properties 151

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.SqlLanguage = TSQL
```

As shown above, the property sets the SQL language variant. The property name is `com.fortify.sca.SqlLanguage` and the value is set to `TSQL`.

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	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. If Quick Scan is not invoked, this file is ignored.
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 "fortify-sca-quickscan.properties" on page 151 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.

Property Name	Description
`com.fortify.sca.BuildID`	Specifies the build ID of the build. **Value type:** String **Default:** (none) **Command-line Option:** `-b`

Property Name	Description
com.fortify.sca. ProjectRoot	Specifies the folder to store intermediate files generated in the translation and scan 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\<user>\AppData\Local\`
com.fortify.sca. DisableDeadCode Elimination	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. 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. fileextensions.java com.fortify.sca. fileextensions.cs com.fortify.sca. fileextensions.js	Specifies how to translate specific file extensions for languages that do not require build integration. The valid types are: ABAP, ACTIONSCRIPT, APEX, APEX_TRIGGER, ARCHIVE, ASPNET, ASP, ASPX, BITCODE, BYTECODE, CFML, COBOL, CSHARP, HTML, JAVA, JAVA_PROPERTIES, JAVASCRIPT, JSP, JSPX, MSIL, MXML, PHP, PLSQL, PYTHON, RUBY, RUBY_ERB, SCALA, SWIFT, TLD, SQL, TSQL, VB, VB6, VBSCRIPT,

Property Name	Description

com.fortify.sca.
fileextensions.py
 	
com.fortify.sca.
fileextensions.rb
 	
com.fortify.sca.
fileextensions.aspx
 	
com.fortify.sca.
fileextensions.php
 	
Note: This is a partial list. For the complete list, see the properties file.
	VISUAL_FORCE, and XML.
	
	Value type: String (valid language type)
	
	Default: See the `fortify-sca.properties` file for the complete list.
	
	Examples:
	
	`com.fortify.sca.fileextensions.java=JAVA`
	`com.fortify.sca.fileextensions.cs=CSHARP`
	`com.fortify.sca.fileextensions.js=JAVASCRIPT`
	`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. JavacCompiler com.fortify.sca. compilers.c++= com.fortify.sca. util.compilers. GppCompiler	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:**

Property Name	Description

`com.fortify.sca.compilers.make=com.fortify.sca.util.compilers.TouchlessCompiler`
`com.fortify.sca.compilers.mvn=com.fortify.sca.util.compilers.MavenAdapter`
Note: This is a partial list. For the complete list, see the properties file.
	• Compiler names can begin or end with an asterisk (*), which matches zero or more characters.
	• Execution of Apple LLVM 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`
`com.fortify.sca.exclude`	Specifies a file or a list of files to exclude from translation. Separate the file list with semicolons (Windows) or a colons (non-Windows systems). **Note:** Fortify Static Code Analyzer only uses this property during translation without build integration. When you integrate with a compiler or build tool, Fortify Static Code Analyzer translates all source files that the build tool processes even if they are specified with this property. **Value type:** String (list of file names) **Default:** Not enabled **Command-line Option:** `-exclude` **Example:** `com.fortify.sca.exclude=file1.x;file2.x`

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. 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 `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. 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`, `findbugs`, `nullptr`, `semantic`, and `structural`. **Value type:** String **Default:** This property is commented out and all analysis types are used in scans. **Command-line Option:** `-analyzers`
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`, `findbugs`, `nullptr`, `semantic`, and `structural`. **Value type:** String **Default:** (none)
com.fortify.sca.	Extends the default exit code options. See "Exit Codes" on

Property Name	Description
`ExitCodeLevel`	page 108 for a description of the exit codes. The valid values are: The valid values are: • `nothing`—Returns exit codes 0, 1, 2, or 3. This is the default setting. • `warnings`—Returns exit codes 0, 1, 2, 3, 4, or 5. • `errors`—Returns exit codes 0, 1, 2, 3, or 5. • `no_output_file`—Returns exit codes 0, 1, 2, 3, or 6. **Command-line Option:** `-exit-code-level`
`com.fortify.sca.IncrementalBaseScan`	If set to true, requests a full scan of a project for which you can run subsequent incremental scans. **Value type:** Boolean **Default:** `false`
`com.fortify.sca.IncrementalScan`	If set to true, requests an incremental rescan of a previously scanned project. **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.hoa.Enable`	If set to true, higher-order analysis is enabled. **Value type:** Boolean **Default:** `true`
`com.fortify.sca.Phase0HigherOrder.Languages`	The languages for which to run phase zero higher order analysis. Valid values are: `python`, `swift`, `ruby`, and `javascript`. **Value type:** String (comma separated list of languages) **Default:** `python,ruby,swift`
`com.fortify.sca.Phase0HigherOrder.Limiter.FixpointPasses`	Specifies the number of fixpoint subpasses to trade off the thoroughness of the analysis with the amount of time it takes.

Property Name	Description
	Note: Running scans with different numbers of fixpoint subpasses often produces slightly different results. **Value type:** Number **Default:** 6
com.fortify.sca.Phase0HigherOrder.Timeout.Soft	Specifies the total time (in seconds) for the High Order Analyzer. When the analyzer reaches the soft timeout limit, it exits at the beginning of the next fixpoint subpass. **Note:** If the analyzer reaches the soft timeout limit during the static inits or main pass, it is ignored. **Value type:** Number **Default:** 1800
com.fortify.sca.Phase0HigherOrder.Timeout.Hard	Specifies the total time (in seconds) for the High Order Analyzer. 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.TypeInferenceLanguages	Comma- or colon-separated list of languages using type inference. **Value type:** String **Default:** javascript,python,ruby
com.fortify.sca.TypeInferencePhase0Timeout	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

Property Name	Description
	Default: `60`
`com.fortify.sca.DisableFunctionPointers`	If set to true, disables function pointers during the scan. **Value type:** Boolean **Default:** `false`
`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.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.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 creating custom

Property Name	Description
	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`
`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.EnableFindbugs`	If set to true, FindBugs is enabled as part of the scan. **Value type:** Boolean **Default:** `true` **Command-line Option:** `-findbugs`
`com.fortify.sca.findbugs.maxheap`	Sets the maximum heap size for findbugs. **Value type:** String **Default:** Maximum heap size for Fortify Static Code Analyzer **Example:** `com.fortify.sca.findbugs.maxheap=500m`
`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.`	Specifies the cutoff level for severity suppression. Fortify

Property Name	Description
`LowSeverityCutoff`	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.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.FilterFile`	Specifies the path to a filter file for the scan. See "Filter Files" on page 112 for more information. **Value type:** String (path) **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)
`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`

Property Name	Description
`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 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.alias.Enable`	If set to true, enables alias analysis. **Value type:** Boolean **Default:** `true`
`com.fortify.sca.UniversalBlacklist`	Specifies a list of functions to blacklist from all analyzers. **Value type:** String (colon-separated list) **Default:** `.*yyparse.*`
`com.fortify.sca.MultithreadedAnalysis`	Specifies whether or not 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)

Property Name	Description
com.fortify.sca. DISabledLanguages	Add a colon-separated list of languages to exclude from the translation phase. The valid language values are `abap, actionscript, apex, cfml, cpp, cobol, configuration, dotnet, java, javascript, jsp, objc, php, plsql, python, ruby, scala, sql, swift, tsql, vb` **Value type:** String **Default:** (none)
com.fortify.sca. JdkVersion	Specifies the Java source code version to the Java translator. **Value type:** String **Default:** `1.8` **Command-line Option:** `-jdk`
com.fortify.sca. JavaClasspath	Specifies the class path used to analyze Java source code. Specify the paths as a semicolon-separated list (Windows) or a colon-separated list (non-Windows systems). **Value type:** String (paths) **Default:** (none) **Command-line Option:** `-cp` or `-classpath`
com.fortify.sca. Appserver	Specifies the application server to process JSP files. The valid values are `weblogic` or `websphere`. **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 application server. **Value type:** String **Default:** (none)

Property Name	Description
	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.JavaSourcepath`	Specifies a colon- or semicolon-separated list of source file directories that are not included in the scan but are used for name resolution. The source path is similar to classpath, except it uses source files rather than class files for resolution. **Value type:** String (paths) **Default:** (none) **Command-line Option:** `-sourcepath`
`com.fortify.sca.JavaSourcepathSearch`	If set to true, Fortify Static Code Analyzer only translates 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. The 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 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. **Value type:** String (encoding) **Default:** `ISO-8859-1`
`com.fortify.sca DotnetVersion`	Specifies the .NET framework version. See the *Micro Focus Fortify Software System Requirements* for a list of

Property Name	Description
	supported versions. **Value type:** String **Default:** (the latest supported version) **Command-line Option:** `-dotnet-version`
`com.fortify.sca DotnetCoreVersion`	Specifies the .NET Core version. **Value type:** String **Default:** (none) **Command-line Option:** `-dotnet-core-version`
`com.fortify.sca DotnetStdVersion`	Specifies the .NET Standard version. **Value type:** String **Default:** (none) **Command-line Option:** `-dotnet-std-version`
`com.fortify.sca. DotnetLibdirs`	Specifies the semicolon-delimited list of directories where third-party DLL files are located. Used for default .NET library files. **Value type:** String (path) **Default:** (none) **Command-line Option:** `-libdirs`
`com.fortify.sca. DotnetLibdirsOnly`	If set to true, disables use of runtime .NET libraries corresponding to target .Net framework version and only uses the libraries referenced by the `com.fortify.sca. DotnetLibdirs` property or the `-libdirs` command-line option. **Value type:** Boolean **Default:** `false` **Command-line Option:** `-libdirs-only`
`com.fortify.sca. NugetCacheDir`	Overrides the default path to the NuGet cache directory. **Value type:** String (path) **Default:** The `.nuget/packages` folder in the current user's home directory (Windows environment variable: USERPROFILE) **Command-line Option:** `-nuget-cache-dir`

Property Name	Description
com.fortify.sca. DotnetSharedFiles	Specifies a semicolon-separated list of the source files for all Shared Projects included in the project you are translating. **Value type:** String (files) **Default:** (none) **Command-line Option:** `-dotnet-shared-files`
com.fortify.sca. DotnetOutputDir	Specifies the output directory where the binary (EXE or DLL) built from the project is placed. **Value type:** String (path) **Default:** (none) **Command-line Option:** `-dotnet-output-dir`
com.fortify.sca. DotnetWebRoot	Specifies the root (home) directory of an ASP.NET project. **Value type:** String (path) **Default:** (none) **Command-line Option:** `-dotnetwebroot`
com.fortify.sca. WebSiteProject	If set to true, the project is of type WebSite. **Value type:** Boolean **Default:** `false` **Command-line Option:** `-dotnet-website`
com.fortify.sca. DotnetWebAppLibs	Specifies a semicolon-separated list of additional reference DLLs needed to translate ASP.NET pages. **Value type:** String (path) **Default:** (none) **Command-line Option:** `-dotnet-applibs`
com.fortify.sca. DotnetCodeBehind	Specifies a semicolon-separated list of source files that are bound to ASP.NET pages (referred to as *code-behind files*). **Value type:** String (path) **Default:** (none) **Command-line Option:** `-dotnet-codebehind`
com.fortify.sca. AspNetCore	If set to true, indicates a web project (ASP.NET or ASP.NET Core) that targets the .NET Core or .NET

Property Name	Description
	Standard framework. **Value type:** Boolean **Default:** `false` **Command-line Option:** `-aspnetcore`
`com.fortify.sca.DotnetAssemblyName`	Specifies the name of the target .NET assembly as specified in Visual Studio project settings. **Value type:** String **Default:** (none) **Command-line Option:** `-dotnet-assembly-name`
`com.fortify.sca.DotnetAlias`	Specifies a list of external aliases for a specified DLL file in the following format: *<alias1>,<alias2>,..=<path_to_dll>*. You can include multiple aliases as a semicolon-separated list of pairs. **Value type:** String **Default:** (none) **Command-line Option:** `-cs-extern-alias`
`com.fortify.sca.DotnetPreprocessorSymbols`	Specifies a semicolon-separated list of preprocessor symbols used in the source code. **Value type:** String **Default:** (none) **Command-line Option:** `-dotnet-preproc-symbols`
`com.fortify.sca.VBCompileOptions`	Specifies any special compilation options required for the correct translation of the source code, such as `OptionStrict`, `OptionInfer`, and `OptionExplicit`. **Value type:** String **Default:** (none) **Command-line Option:** `-vb-compile-options`
`com.fortify.sca.VBGlobalImports`	Specifies a semicolon-separated list of namespaces imported for all source files in the project. **Value type:** String **Default:** (none) **Command-line Option:** `-vb-imports`

Property Name	Description
com.fortify.sca. VBMyType	Specifies the value for the _MYTYPE preprocessor symbol that is specified in the <MyType> tag in the project settings. This is required if the source code to be translated uses My namespace. **Value type:** String **Default:** (none) **Command-line Option:** -vb-mytype
com.fortify.sca. VBRootNamespace	Specifies the root namespace for the project as specified in Visual Studio project settings. **Value type:** String **Default:** (none) **Command-line Option:** -vb-root
com.fortify.sca. XamarinAndroidVersion	Specifies the target Android SDK version for Xamarin Android projects. **Value type:** String **Default:** (the latest installed version) **Command-line Option:** -xamarin-android-version
com.fortify.sca. XamariniOSVersion	Specifies the target iOS SDK version for Xamarin iOS projects. **Value type:** String **Default:** (the latest installed version) **Command-line Option:** -xamarin-ios-version
WinForms. TransformDataBindings WinForms. TransformMessageLoops WinForms. TransformChange NotificationPattern WinForms. CollectionMutation Monitor.Label WinForms. ExtractEventHandlers	Set 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

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 scanning includes JavaScript-based code or any HTML files that include JavaScript. Note that enabling this property can increase the translation and scan 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 HTML tags for Fortify Static Code Analyzer to include in the DOM modeling. **Value type:** String (comma-separated HTML tag names) **Default:** A limited set of DOM element tags such as `html`, `head`, `input`, `button`, and `iframe`. **Example:** `com.fortify.sca.DOMModeling.tags=div,p`

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. DisableJavascript Extraction	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.AngularJS com.fortify.sca. skip.libraries.ES6 com.fortify.sca. skip.libraries.jQuery	Specifies a list of comma- or colon-separated JavaScript library files that are not translated. You can use regular expressions in the file names. Note that the regular expression `'(-\d\.\d\.\d)?'` is automatically inserted before `.min.js` or `.js` for each file name included in the `com.fortify.sca.skip.libraries.jQuery` property value.

Property Name	Description
com.fortify.sca. skip.libraries.javascript	**Value type:** String **Defaults:** • AngularJS: angular.js,angular.min.js, angular-animate.js,angular-aria.js, angular_1_router.js,angular-cookies.js, angular-message-format.js,angular-messages.js, angular-mocks.js,angular-parse-ext.js, angular-resource.js,angular-route.js, angular-sanitize.js,angular-touch.js • 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
com.fortify.sca. PHPVersion	Specifies the PHP version. For a list of valid versions, see the *Micro Focus Fortify Software System Requirements*. **Value type:** String **Default:** 7.0 **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
com.fortify.sca. PythonPath	Specifies a colon- or semicolon-separated 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.

Property Name	Description
	Value type: String (path) **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.DjangoTemplateDirs`	Specifies path to 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.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 the path(s) to a RubyGems location. Set this value if the project has associated gems to scan. **Value type:** String (path) **Default:** (none) **Command-line Option:** `-rubygem-path`
`com.fortify.sca.FlexLibraries`	Specifies a semicolon-separated list (Windows) or a colon-separated list (non-Windows systems) of libraries to "link" to. This list should 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) **Default:** (none) **Command-line Option:** `-flex-libraries`

Property Name	Description
com.fortify.sca. FlexSdkRoot	Specifies the root location of a valid Flex SDK. The folder should contain a frameworks folder that contains a `flex-config.xml` file. It should 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 the list of directories with semicolons (Windows) or colons (non-Windows systems). **Value type:** String (path) **Default:** (none) **Command-line Option:** `-flex-source-root`
com.fortify.sca. AbapDebug	If set to true, Fortify Static Code Analyzer adds ABAP statements to debug messages. **Value type:** String (statement) **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)
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. **Value type:** Boolean **Default:** `false` **Command-line Option:** `-fixed-format`
com.fortify.sca. SqlLanguage	Sets the SQL language variant. The valid values are `PLSQL` (for Oracle PL/SQL) and `TSQL` (for Microsoft T-SQL). **Value type:** String (SQL language type) **Default:** `TSQL` **Command-line Option:** `-sql-language`
com.fortify.sca.	If set to true, Fortify Static Code Analyzer treats

Property Name	Description
`CfmlUndefinedVariablesAreTainted`	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`
`com.fortify.sca.FVDLDisableDescriptions`	If set to true, excludes descriptions from the analysis results file (FVDL). **Value type:** Boolean **Default:** `false` **Command-line Option:** `-fvdl-no-description`
`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.FVDLDisableEngineData`	If set to true, excludes the engine data from the analysis results file (FVDL). **Value type:** Boolean **Default:** `false` **Command-line Option:** `-fvdl-no-enginedata`
`com.fortify.sca.`	If set to true, excludes code snippets from the analysis

Property Name	Description
FVDLDisableSnippets	results file (FVDL). **Value type:** Boolean **Default:** false **Command-line Option:** -fvdl-no-snippets
com.fortify.sca. FVDLDisableLabelEvidence	If set to true, excludes the label evidence from the analysis results file (FVDL). **Value type:** Boolean **Default:** false
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
com.fortify.sca. ResultsFile	The file to which results are written. **Value type:** String **Default:** (none) **Command-line Option:** -f **Example:** com.fortify.sca.ResultsFile=results.fpr
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. 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.	If set to true, Fortify Static Code Analyzer prints results as

Property Name	Description
`ResultsAsAvailable`	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.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.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.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 **Default:** (not enabled)
`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

Property Name	Description
	Default: 2
com.fortify.sca. MobileBuildSession	If set to true, Fortify Static Code Analyzer copies source files into the build session. **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
com.fortify.sca. LogFile	Specifies the default log file location. **Value type:** String (path) **Default:** ${com.fortify.sca.ProjectRoot}/log/sca.log **Command-line Option:** -logfile
com.fortify.sca. LogMaxSize	When this property is set, it enables log rotation for the Fortify Static Code Analyzer log. The value is the number bytes that Fortify Static Code Analyzer can write to the log file before it is rotated. You must use this property with com.fortify.sca.LogMaxFiles. **Value type:** Number **Default:** (none)
com.fortify.sca. LogMaxFiles	The number of log files to include in the log file rotation set. When all files are filled, the first file in the rotation is overwritten. The value must be at least 1. You must use this property with com.fortify.sca.LogMaxSize. **Value type:** Number **Default:** (none)
com.fortify.sca. ClobberLogFile	If set to true, Fortify Static Code Analyzer overwrites the log file for each new scan. **Value type:** Boolean **Default:** false

Property Name	Description
	Command-line Option: `-clobber-log`
`com.fortify.sca.SuppressLogPrefix`	If set to true, no prefix is added to the log file name. **Value type:** Boolean **Default:**`true`
`com.fortify.sca.PrintPerformanceDataAfterScan`	If set to true, Fortify Static Code Analyzer writes performance-related data to the log file after the scan is complete. This value is automatically set to true when in debug mode. **Value type:** Boolean **Default:** `false`
`com.fortify.sca.Debug`	If set to true, Fortify Static Code Analyzer includes debug information in the log file. **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.DebugTrackMem`	If set to true, enables additional debugging for performance information. **Value type:** Boolean **Default:** (not enabled)
`com.fortify.sca.CollectPerformanceData`	If set to true, enables additional timers to track performance. **Value type:** Boolean **Default:** (not enabled)
`com.fortify.sca.Verbose`	If set to true, includes verbose messages in the log file. **Value type:** Boolean **Default:** (not enabled)

Property Name	Description
	Command-line Option: `-verbose`
`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`
`com.fortify.sca.cpfe.command`	Sets the location of the CPFE binary to use in the translation phase. **Value type:** String (path) **Default:** `${com.fortify.Core}/private-bin/sca/cpfe48`
`com.fortify.sca.cpfe.441`	If set to true, Fortify Static Code Analyzer uses CPFE version 4.4.1. **Value type:** Boolean **Default:** `false`
`com.fortify.sca.cpfe.441.command`	Sets the location of the CPFE binary (version 4.4.1) to use in the translation phase. **Value type:** String (path) **Default:** `${com.fortify.Core}/private-bin/sca/cpfe441.rfct`
`com.fortify.sca.cpfe.options`	Adds options to the CPFE command line to use when translating C/C++ code. **Value type:** String **Default:** `--remove_unneeded_entities --suppress_vtbl -tused`
`com.fortify.sca.cpfe.file.option`	Sets the name of CPFE option that specifies the output (for example NST) file name.

Property Name	Description
	Value type: String **Default:** `--gen_c_file_name` **Example:** `com.fortify.sca.cpfe.file.option= --gen_c_file_name`
`com.fortify.sca. cpfe.multibyte`	If set to true, CPFE handles multibyte characters in the source code. This enables Fortify Static Code Analyzer to handle code with multibyte encoding, such as SJIS (Japanese). **Value type:** Boolean **Default:** `false`
`com.fortify.sca. cpfe.CaptureWarnings`	If set to true, any CPFE warnings are included in the Fortify Static Code Analyzer log. **Value type:** Boolean **Default:** `false`
`com.fortify.sca. cpfe.FailOnError`	If set to true, CPFE fails if there is an error. **Value type:** Boolean **Default:** `false`
`com.fortify.sca. cpfe.IgnoreFileOpen Failures`	If set to true, any failure to open a source file (including headers) is considered a warning instead of an error. **Value type:** Boolean **Default:** `false`
`com.fortify.sca. ASPVirtualRoots. <virtual_path>`	Specifies a semicolon delimited 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 analysis. **Value type:** Boolean **Default:** `false`

fortify-sca-quickscan.properties

Fortify Static Code Analyzer offers a less-intensive 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 searches for high-confidence, high-severity issues only. For more information about Quick Scan mode, see the *Micro Focus Fortify Audit Workbench User Guide*.

Note: Properties in this file are only used if you specify the `-quick` option on the command line for your scan.

The 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 values for this property are: `buffer`, `content`, `configuration`, `controlflow`, `dataflow`, `findbugs`, `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 "fortify-sca.properties" on page 123 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*

Property Name	Description
	Guide. **Value type:** String **Quick scan default:** `Quick View` **Default:** (none)
`com.fortify.sca.FPRDisableMetatable`	Disables the creation of the metatable, which includes creation of the information for the Function view in Audit Workbench and enables right-click on a variable in the source window to show the declaration. If C/C++ scans take an extremely long time, you can set this property to potentially reduce the scan time by hours. **Value type:** Boolean **Quick scan default:** `true` **Default:** `false`
`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`
`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)

Property Name	Description
	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 analysis, and 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`

Property Name	Description
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

Send Documentation Feedback

If you have comments about this document, you can contact the documentation team by email. If an email client is configured on this computer, click the link above and an email window opens with the following information in the subject line:

Feedback on User Guide (Fortify Static Code Analyzer 18.10)

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!