
Micro Focus Fortify Static Code Analyzer

Software Version: 18.20

User Guide

Document Release Date: November 2018
Software Release Date: November 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	9
Contacting Micro Focus Fortify Customer Support	9
For More Information	9
About the Documentation Set	9
Change Log	10
Chapter 1: Introduction	13
Fortify Static Code Analyzer	13
Fortify CloudScan	13
Fortify Scan Wizard	14
Fortify Software Security Content	14
About the Analyzers	14
Related Documents	16
All Products	16
Micro Focus Fortify Software Security Center	17
Micro Focus Fortify Static Code Analyzer	17
Chapter 2: Analysis Process Overview	19
Analysis Process	19
Parallel Processing	20
Translation Phase	20
Mobile Build Sessions	21
Mobile Build Session Version Compatibility	21
Creating a Mobile Build Session	21
Importing a Mobile Build Session	21
Analysis Phase	22
Incremental Analysis	22
Translation and Analysis Phase Verification	23
Chapter 3: Translating Java Code	24
Java Command-Line Syntax	24
Java Command-Line Options	25
Java Command-Line Examples	27
Handling Resolution Warnings	27
Java Warnings	27
Using FindBugs	28

Translating Java EE Applications	29
Translating the Java Files	29
Translating JSP Projects, Configuration Files, and Deployment Descriptors	29
Java EE Translation Warnings	29
Translating Java Bytecode	30
Troubleshooting JSP Translation Issues	30
Chapter 4: Translating .NET Code	32
About Translating .NET Code	32
.NET Command-Line Syntax	33
Translating .NET Binaries	34
Binary .NET Translation Command-Line Options	35
Handling Translation Errors	38
.NET Translation Errors	38
ASP.NET Errors	38
Chapter 5: Translating C and C++ Code	39
C and C++ Code Translation Prerequisites	39
C and C++ Command-Line Syntax	39
Options for Code in Visual Studio Solution or MSBuild Project	40
Scanning Pre-processed C and C++ Code	40
Chapter 6: Translating JavaScript Technologies	41
Translating Pure JavaScript Projects	41
Skipping Translation of JavaScript Library Files	41
Translating JavaScript Projects with HTML Files	42
Including External JavaScript or HTML in the Translation	43
Translating AngularJS Code	43
Scanning JavaScript Technologies	44
Chapter 7: Translating Python Code	45
Python Translation Command-Line Syntax	45
Including Import Files	45
Including Namespace Packages	46
Using the Django Framework with Python	46
Python Command-Line Options	46
Python Command-Line Examples	47
Chapter 8: Translating Code for Mobile Platforms	48

Translating Apple iOS Projects	48
iOS Project Translation Prerequisites	48
iOS Code Analysis Command-Line Syntax	49
Translating Android Projects	49
Android Project Translation Prerequisites	49
Android Code Analysis Command-Line Syntax	50
Filtering Issues Detected in Android Layout Files	50
Chapter 9: Translating Ruby Code	51
Ruby Command-Line Syntax	51
Ruby Command-Line Options	51
Adding Libraries	52
Adding Gem Paths	52
Chapter 10: Translating Apex and Visualforce Code	53
Apex Translation Prerequisites	53
Apex and Visualforce Command-Line Syntax	53
Apex and Visualforce Command-Line Options	54
Downloading Customized Salesforce Database Structure Information	54
Chapter 11: Translating COBOL Code	56
Preparing COBOL Source Files for Translation	56
COBOL Command-Line Syntax	57
COBOL Command-Line Options	57
Chapter 12: Translating Other Languages	59
Translating PHP Code	59
PHP Command-Line Options	59
Translating ABAP Code	60
INCLUDE Processing	60
Importing the Transport Request	61
Adding Fortify Static Code Analyzer to Your Favorites List	61
Running the Fortify ABAP Extractor	62
Uninstalling the Fortify ABAP Extractor	63
Translating Flex and ActionScript	64
Flex and ActionScript Command-Line Options	64
ActionScript Command-Line Examples	65
Handling Resolution Warnings	66
ActionScript Warnings	66
Translating ColdFusion Code	66
ColdFusion Command-Line Syntax	66

ColdFusion Command-Line Options	67
Translating SQL	67
PL/SQL Command-Line Example	68
T-SQL Command-Line Example	68
Translating Scala Code	68
Translating ASP/VBScript Virtual Roots	68
Classic ASP Command-Line Example	70
VBScript Command-Line Example	71
Chapter 13: Integrating into a Build	72
Build Integration	72
Make Example	73
Devenv Example	73
Modifying a Build Script to Invoke Fortify Static Code Analyzer	73
Touchless Build Integration	74
Ant Integration	74
Gradle Integration	75
Maven Integration	75
Installing and Updating the Fortify Maven Plugin	75
Testing the Fortify Maven Plugin Installation	76
Using the Fortify Maven Plugin	77
Excluding Files from the Scan	77
MSBuild Integration	78
Using MSBuild Integration	79
Using the Touchless MSBuild Integration	79
Adding Custom Tasks to your MSBuild Project	80
Chapter 14: Command-Line Interface	88
Output Options	88
Translation Options	90
Analysis Options	91
Other Options	94
Directives	95
Specifying Files	96
Chapter 15: Command-Line Utilities	98
Fortify Static Code Analyzer Utilities	98
Other Command-Line Utilities	99
Checking the Fortify Static Code Analyzer Scan Status	99
SCAState Utility Command-Line Options	100

Working with FPR Files from the Command Line	101
Merging FPR Files	102
Displaying Analysis Results Information from an FPR File	103
Extracting a Source Archive from an FPR File	106
Allocating More Memory for FPRUtility	107
Generating Reports from the Command Line	108
Generating a BIRT Report	108
Generating a Legacy Report	110
About Updating Security Content	111
Updating Security Content	111
 Chapter 16: Troubleshooting	 113
Exit Codes	113
Translation Failed Message	114
Issue Non-Determinism	114
C/C++ Precompiled Header Files	114
Accessing Log Files	115
Configuring Log Files	115
Understanding Log Levels	116
Reporting Issues and Requesting Enhancements	116
 Appendix A: Filtering the Analysis	 117
Filter Files	117
Filter File Example	117
 Appendix B: Scan Wizard	 120
Preparing to use the Scan Wizard	120
Starting the Scan Wizard	121
Starting Scan Wizard on a System with Fortify SCA and Applications Installed	121
Starting Scan Wizard as a Stand-Alone Utility	122
 Appendix C: Sample Files	 123
Basic Samples	123
Advanced Samples	125
 Appendix D: Configuration Options	 127
Fortify Static Code Analyzer Properties Files	127
Properties File Format	127
Precedence of Setting Properties	128
fortify-sca.properties	128

fortify-sca-quickscan.properties	156
Send Documentation Feedback	161

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
18.20	<p>Added:</p> <ul style="list-style-type: none">• "Uninstalling the Fortify ABAP Extractor" on page 63 <p>Updated:</p> <ul style="list-style-type: none">• "Translating .NET Code" on page 32 - Added Fortify Static Code Analyzer msbuild integration and changed manual command-line option descriptions to specify that they to be used only for translating binaries• "Translating JavaScript Technologies" on page 41 - Includes new information about how to translate and scan TypeScript code• "Translating Python Code" on page 45 - Described new support for namespace packages and other changes• "Translating Code for Mobile Platforms" on page 48 - Expanded the information for both iOS and Android projects• "Fortify Static Code Analyzer Build Integration" on page 77 - Updated the command syntax to translate code with the Fortify Maven Plugin• "MSBuild Integration" on page 78 - Clarified the three different methods of integrating Fortify Static Code Analyzer with MSBuild• "Translation Options" on page 90 - Added TYPESCRIPT as a valid value for the <code>-noextension-type</code> option• "Generating a BIRT Report" on page 108 - Options added to support new DISA STIG versions (4.5, 4.6, and 4.7), and added a new report template DISA CCI 2.• "Accessing Log Files" on page 115 - Logging updates• "fortify-sca.properties" on page 128 - Added new properties: <code>com.fortify.sca.LogLevel</code> and <code>com.fortify.sca.skip.libraries.typescript</code>, added <code>typescript</code> as a valid value for several properties
18.10 Revision 1: June 1, 2018	<p>Updated:</p> <ul style="list-style-type: none">• "Translating Python Code" on page 45 - Provided more information for the <code>-python-path</code> option and added examples

Software Release / Document Version	Changes
18.10	<p>Added:</p> <ul style="list-style-type: none">• "Translating AngularJS Code" on page 43 - Merged the AngularJS Technical Preview information into this guide• "Downloading Customized Salesforce Database Structure Information" on page 54 for Apex translation <p>Updated:</p> <ul style="list-style-type: none">• "Binary .NET Translation Command-Line Options" on page 35 and "Fortify.TranslateTask" on page 81 - New options for Shared Projects and Xamarin projects• "Python Command-Line Options" on page 46 - New option for Python version and other minor edits• "Maven Integration" on page 75 - Branding changes for the Fortify Maven Plugin group ID• "Fortify.TranslateTask" on page 81 - Added Xamarin options for the custom MSBuild translate task• "fortify-sca.properties" on page 128 - New properties for .NET and Python• "About the Analyzers" on page 14 and "fortify-sca.properties" on page 128 - Merged the High Order Analyzer Technical Preview information into this guide• Minor edits to incorporate branding changes
17.20	<p>Added:</p> <ul style="list-style-type: none">• "Fortify Software Security Content" on page 14• "About Translating .NET Code" on page 32 - Add descriptions for the different ways you can translate .NET project• "Translating Scala Code" on page 68 - New feature in this release• "Adding Custom Tasks to your MSBuild Project" on page 80 - Feature updated for the redesigned .NET implementation introduced in version 16.20• "Allocating More Memory for FPRUtility" on page 107• "Issue Non-Determinism" on page 114 - Provide instructions for how to disable parallel analysis mode if scan results include issue non-determinism (parallel analysis mode is now enabled by default) <p>Updated:</p> <ul style="list-style-type: none">• "Java Command-Line Options" on page 25 - Options -source and -jdk updated to support Java 9• "Binary .NET Translation Command-Line Options" on page 35 - Added new options for manual translation of ASP.NET, .NET Core, ASP.NET Core, and .NET Standard projects

Software Release / Document Version	Changes
	<ul style="list-style-type: none">• "Apex Translation Prerequisites" on page 53 - Added more details about prerequisites for translating Apex and Visualforce code• "Translating PHP Code" on page 59 - Added a new option for translating PHP code• "Displaying Analysis Results Information from an FPR File" on page 103 and "Extracting a Source Archive from an FPR File" on page 106 - Added new options for the FPRUtility• "About Updating Security Content" on page 111 - Added new options available for the fortifyupdate utility <p>Removed:</p> <ul style="list-style-type: none">• "Migrating Audit Data from Previous FPR Versions" - The migrate feature for SCA 4.x to 5.x was removed from FPRUtility command.

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](#) 13
- [About the Analyzers](#) 14
- [Related Documents](#) 16

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 open and view results in 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 120](#).

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 111](#).

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	<p>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.</p> <p>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 and TypeScript. For a description of the Higher Order Analyzer properties, see "fortify-sca.properties" on page 128 and search for "higher-order analysis."</p>
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

Analyzer	Description
	string, and execution path issues, but is not limited to these categories. For example, the Semantic Analyzer detects deprecated functions in Java and unsafe functions in C/C++, such as <code>gets()</code> .
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
<i>About Micro Focus Fortify Product Software Documentation</i> About_Fortify_Docs_<version>.pdf	This paper provides information about how to access Micro Focus Fortify product documentation. Note: This document is included only with the product download.
<i>Micro Focus Fortify Software System Requirements</i> 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.
<i>Micro Focus Fortify Software Release Notes</i> 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.

Document / File Name	Description
<i>What's New in Micro Focus Fortify Software <version></i> Fortify_Whats_New_<version>.pdf Fortify_Whats_New_Help_<version>	This document describes the new features in Fortify Software products.
<i>Micro Focus Fortify Open Source and Third-Party License Agreements</i> Fortify_OpenSrc_<version>.pdf Fortify_OpenSrc_<version>	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
<i>Micro Focus Fortify Software Security Center User Guide</i> SSC_Guide_<version>.pdf SSC_Help_<version>	<p>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.</p> <p>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.</p>

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

Document / File Name	Description
<i>Micro Focus Fortify Static Code Analyzer User Guide</i> 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.
<i>Micro Focus Fortify Static Code Analyzer Performance Guide</i> 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.
<i>Micro Focus Fortify Static Code Analyzer Custom Rules Guide</i> 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	19
Translation Phase	20
Mobile Build Sessions	21
Analysis Phase	22
Incremental Analysis	22
Translation and Analysis Phase Verification	23

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 72](#).
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 22](#).
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 23](#).

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 24](#)
- ["Translating .NET Code" on page 32](#)
- ["Translating C and C++ Code" on page 39](#)
- ["Translating JavaScript Technologies" on page 41](#)
- ["Translating Python Code" on page 45](#)
- ["Translating Code for Mobile Platforms" on page 48](#)
- ["Translating Ruby Code" on page 51](#)
- ["Translating Apex and Visualforce Code" on page 53](#)

- ["Translating COBOL Code" on page 56](#)
- ["Translating Other Languages" on page 59](#)

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, 18.20.0140). 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 91](#) and ["Output Options" on page 88](#)).

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	24
Handling Resolution Warnings	27
Using FindBugs	28
Translating Java EE Applications	29
Translating Java Bytecode	30
Troubleshooting JSP Translation Issues	30

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 74](#).

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
<code>-appserver</code> <code>weblogic websphere</code>	Specifies the application server to process JSP files. Equivalent property name: <code>com.fortify.sca.AppServer</code>
<code>-appserver-home <path></code>	Specifies the application server's home. <ul style="list-style-type: none">• For WebLogic, this is the path to the directory that contains the <code>server/lib</code> directory.• For WebSphere, this is the path to the directory that contains the <code>JspBatchCompiler</code> script. Equivalent property name: <code>com.fortify.sca.AppServerHome</code>
<code>-appserver-version</code> <code><version></code>	Specifies the version of the application server. See the <i>Micro Focus Fortify Software System Requirements</i> document for supported versions. Equivalent property name: <code>com.fortify.sca.AppServerVersion</code>

Java/Java EE Option	Description
<code>-cp <paths> </code> <code>-classpath <paths></code>	<p>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:</p> <pre data-bbox="678 428 1403 487">-cp "build/classes:lib/*.jar"</pre> <p>For information about file specifiers, see "Specifying Files" on page 96.</p> <p>Equivalent property name: <code>com.fortify.sca.JavaClasspath</code></p>
<code>-extdirs <dirs></code>	<p>Similar to the javac <code>extdirs</code> option, accepts a colon- or semicolon-separated list of directories. Any JAR files found in these directories are included implicitly on the class path.</p> <p>Equivalent property name: <code>com.fortify.sca.JavaExtdirs</code></p>
<code>-java-build-dir <dirs></code>	<p>Specifies one or more directories that contain compiled Java sources. You must specify this for FindBugs results as described in "Analysis Options" on page 91.</p>
<code>-source <version> </code> <code>-jdk <version></code>	<p>Indicates the JDK version for which the Java code is written. The valid values for <code><version></code> are 1.5, 1.6, 1.7, 1.8, and 1.9. The default is 1.8.</p> <p>Equivalent property name: <code>com.fortify.sca.JdkVersion</code></p>
<code>-sourcepath <paths></code>	<p>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.</p> <p>Equivalent property name: <code>com.fortify.sca.JavaSourcePath</code></p>

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 27](#).

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"
```

Troubleshooting JSP Translation Issues

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 29](#).

Chapter 4: Translating .NET Code

This chapter describes how to use Fortify Static Code Analyzer to translate .NET and ASP.NET applications built with Visual Studio or MSBuild. Fortify Static Code Analyzer analyzes code written in C#, VB.NET, and ASP.NET (including .cshhtml, .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	32
.NET Command-Line Syntax	33
Translating .NET Binaries	34
Handling Translation Errors	38

About Translating .NET Code

There are four different methods of translating .NET applications.

- Use the Micro Focus Fortify Extension for Visual Studio to translate and analyze your project or solution directly from the Visual Studio IDE. See *Micro Focus Fortify Extension for Visual Studio User Guide* for more information.
- Use Fortify Static Code Analyzer build integrations to translate and analyze your project or solution from the command line (see "[.NET Command-Line Syntax](#)" on the next page).
- Include Fortify Static Code Analyzer as part of your MSBuild project. You can either:
 - Use touchless MSBuild integration to launch Fortify Static Code Analyzer as part of your MSBuild project. For more information, see "[Using the Touchless MSBuild Integration](#)" on page 79.
 - 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 80.
- Use the Fortify Static Code Analyzer command line to translate binaries (see "[Translating .NET Binaries](#)" on page 34. With this method, you must provide all the needed information with the .NET options (see "[Binary .NET Translation Command-Line Options](#)" on page 35).

Important! Fortify strongly recommends that you use either the Micro Focus Fortify Extension for Visual Studio from the IDE, the Fortify Static Code Analyzer build integrations, or MSBuild touchless integration because these are the only three methods where Fortify Static Code Analyzer automatically detects all the project information for the best translation of your project or solution.

Fortify Static Code Analyzer does not automatically detect all required project information if you use custom MSBuild tasks or translate binaries. Therefore, for these two methods you must provide all the necessary information by specifying appropriate command-line options (see "[Binary .NET Translation Command-Line Options](#)" on page 35) or custom MSBuild task parameters (see "[Fortify.TranslateTask](#)" on page 81).

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, AXML, BAML, CSHtml, Master, VBHTML, and XAML.
- All reference DLLs used by your project.
This includes those from the .NET framework, NuGet packages, and third-party DLLs.

Important! The first time you translate a .NET Core, an ASP.NET Core, a .NET Standard, or a .NET Portable project, run the `dotnet restore` command from the Developer Command Prompt for Visual Studio first to ensure that all the necessary reference libraries are installed locally.

Note: Fortify Static Code Analyzer does not support .NET Core 1.0 projects that use `project.json` and `<project_name>.xproj` configuration files. If your .NET Core 1.0 project contains these files, do one of the following before Fortify Static Code Analyzer translation:

- Convert `project.json` and `.xproj` files to the `.csproj` format
- Migrate your project to .NET Core 1.1 or later

.NET Command-Line Syntax

You can translate your .NET projects or solutions from the command line by wrapping the build command with an invocation of Fortify Static Code Analyzer.

The following examples demonstrate the command-line syntax to translate a solution called `Sample1.sln` with `devenv` and `MSBuild` commands:

```
sourceanalyzer -b <build_id> devenv Sample1.sln /rebuild  
sourceanalyzer -b <build_id> msbuild /t:rebuild Sample1.sln
```

Note: Fortify Static Code Analyzer converts `devenv` invocations and command-line options to `MSBuild` invocations, and therefore the previous two command lines examples are actually equivalent.

This performs the translation phase on all files built with Visual Studio or `MSBuild`.

You should run the `devenv` command-line example from the Developer Command Prompt for Visual Studio. You can run the `MSBuild` command-line example from the Developer Command Prompt for Visual Studio or from the Windows Command Prompt, except for Xamarin projects. When you run from the Windows Command Prompt, the path to the `MSBuild` executable must be included in your `PATH` environment variable.

Important! You must translate Xamarin projects from Developer Command Prompt for Visual Studio even if you use `msbuild` integration.

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 91).

Translating .NET Binaries

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.

To translate binaries from a simple .NET project or solution, which does 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_binaries> <project_aspnet_pages>
```

Note: You can improve translation by providing information using the command-line options described in "[Binary .NET Translation Command-Line Options](#)" on the next page.

If your project contains multiple sub-projects, you must translate built from each sub-project separately.

Important! Do not translate binaries from all or multiple sub-projects with a single Fortify Static Code Analyzer invocation.

To scan all sub-projects together after the translation, use the same build ID when you translate binaries from each sub-project, as follows:

```
sourceanalyzer -dotnet-version <version> -b <build_id>  
-libdirs <paths> <project_1_binaries> <project_1_aspnet_pages>  
...  
sourceanalyzer -dotnet-version <version> -b <build_id>  
-libdirs <paths> <project_n_binaries> <project_n_aspnet_pages>
```

where `<project_1_binaries>` represents binaries built from the first sub-project inside your main project or solution, `<project_1_aspnet_pages>` represents ASP.NET pages that belong to the first sub-project, and so on.

Binary .NET Translation Command-Line Options

The following table describes the .NET command-line options used only for translating binaries (see ["Translating .NET Binaries" on the previous page](#)).

.NET Option	Description
<code>-dotnet-version <version></code>	<p>Specifies the .NET framework version. See the <i>Micro Focus Fortify Software System Requirements</i> 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 <code>-libdirs</code> option, unless the <code>-libdirs-only</code> option is specified.</p> <p>Equivalent property name: <code>com.fortify.sca.DotnetVersion</code></p>
<code>-dotnet-core-version <version></code>	<p>Specifies the .NET Core 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 <code>-libdirs</code> option, unless the <code>-libdirs-only</code> option is specified.</p> <p>Equivalent property name: <code>com.fortify.sca.DotnetCoreVersion</code></p>
<code>-dotnet-std-version <version></code>	<p>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 <code>-libdirs</code> option, unless the <code>-libdirs-only</code> option is specified.</p> <p>Equivalent property name: <code>com.fortify.sca.DotnetStdVersion</code></p>
<code>-libdirs <dLLs> <paths></code>	<p>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.</p> <p>Equivalent property name: <code>com.fortify.sca.DotnetLibdirs</code></p>
<code>-libdirs-only</code>	<p>Sets the list of directories or paths to only those specified by the <code>-libdirs</code> 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 <code>-dotnet-version</code> option.</p> <p>Equivalent property name: <code>com.fortify.sca.DotnetLibdirsOnly</code></p>
<code>-dotnet-output-dir <path></code>	<p>Specifies the output directory where the binary (EXE or DLL) built from the project is placed.</p>

.NET Option	Description
	<p>Equivalent property name: com.fortify.sca.DotnetOutputDir</p>
<p>-nuget-cache-dir <path></p>	<p>(.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 the .nuget/packages folder in the current user's home directory (Windows environment variable: USERPROFILE).</p> <p>Equivalent property name: com.fortify.sca.NugetCacheDir</p>
<p>-dotnetwebroot <root_dir></p>	<p>(.NET Web projects only) Specifies the home directory of an ASP.NET project.</p> <p>Equivalent property name: com.fortify.sca.DotnetWebRoot</p>
<p>-dotnet-website</p>	<p>(.NET Web projects only) Indicates that the project is an ASP.NET website.</p> <p>Note: Do not specify this option for ASP.NET Web Applications or any other kind of ASP.NET project except Website.</p> <p>Equivalent property name: com.fortify.sca.WebSiteProject</p>
<p>-dotnet-applibs <dLLs></p>	<p>(.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.</p> <p>Equivalent property name: com.fortify.sca.DotnetWebAppLibs</p>
<p>-aspnetcore</p>	<p>(.NET Web projects only) Indicates a web project (ASP.NET or ASP.NET Core) that targets the .NET Core or .NET Standard framework.</p> <p>Equivalent property name: com.fortify.sca.AspNetCore</p>
<p>-dotnet-assembly-name <assembly_name></p>	<p>Specifies the name of the target .NET assembly as specified in Visual Studio project settings.</p> <p>Equivalent property name: com.fortify.sca.DotnetAssemblyName</p>

.NET Option	Description
<p><code>-dotnet-preproc-symbols <symbols></code></p>	<p>Specifies a semicolon-separated list of preprocessor symbols used in building the project binaries. For example:</p> <pre data-bbox="509 359 1398 422">-dotnet-preproc-symbols "DEBUG;TRACE"</pre> <p>Equivalent property name: com.fortify.sca.DotnetPreprocessorSymbols</p>
<p><code>-cs-extern-alias <aliases_path_pairs></code></p>	<p>(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 <code>-cs-extern-alias</code> options on the command line.</p> <p>Equivalent property name: com.fortify.sca.DotnetAlias</p>
<p><code>-vb-compile-options <compile_options></code></p>	<p>(VB.NET projects only) Specifies any special compilation options used in building the project binaries, such as <code>OptionStrict</code>, <code>OptionInfer</code>, and <code>OptionExplicit</code>.</p> <p>The format for <code><compile_options></code> is a comma-separated list of: <code><option>=On Off</code>. For example:</p> <pre data-bbox="509 1016 1398 1115">-vb-compile-options "OptionStrict=On,OptionExplicit=Off"</pre> <p>Equivalent property name: com.fortify.sca.VBCompileOptions</p>
<p><code>-vb-imports <namespaces></code></p>	<p>(VB.NET projects only) Specifies a semicolon-separated list of namespaces imported for the entire project.</p> <p>Equivalent property name: com.fortify.sca.VBGlobalImports</p>
<p><code>-vb-mytype <symbol></code></p>	<p>(VB.NET projects only) Specifies the value for the <code>_MYTYPE</code> preprocessor symbol that is specified in the <code><MyType></code> tag in the project settings. This is required if the project references <code>My</code> namespace.</p> <p>Equivalent property name: com.fortify.sca.VBMyType</p>
<p><code>-vb-root <namespace></code></p>	<p>(VB.NET and Silverlight projects only) Specifies the root namespace for the project as specified in Visual Studio project settings.</p> <p>Equivalent property name: com.fortify.sca.VBRootNamespace</p>
<p><code>-xamarin-android-version <version></code></p>	<p>(Required for Xamarin.Android projects) Specifies the target Android SDK version for Xamarin Android projects. If you specify an Android</p>

.NET Option	Description
	<p>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 <code>-libdirs</code> option, unless the <code>-libdirs-only</code> option is specified.</p> <p>Equivalent property name: <code>com.fortify.sca.XamarinAndroidVersion</code></p>
<code>-xamarin-ios-version <version></code>	<p>(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 <code>-libdirs</code> option, unless the <code>-libdirs-only</code> option is specified.</p> <p>Equivalent property name: <code>com.fortify.sca.XamariniOSVersion</code></p>

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 all the source files in your project. Rerun the translation with the `-debug-verbose` option and provide the Fortify Support log 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	39
C and C++ Command-Line Syntax	39
Scanning Pre-processed C and C++ Code	40

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.

Options for Code in Visual Studio Solution or MSBuild Project

If your C/C++ code is part of a Visual Studio solution or MSBuild project, you can translate it using one of the following three methods:

- Use Micro Focus Fortify Extension for Visual Studio to translate and analyze your solution or project directly from Visual Studio. See the *Micro Focus Fortify Extension for Visual Studio User Guide* for more information.
- Use Fortify Static Code Analyzer build integration to translate your project or solution from the Developer Command Prompt for Visual Studio. Wrap your build command with an invocation of Fortify Static Code Analyzer, as shown in the following command line examples:

```
sourceanalyzer -b <build_id> devenv <solution_file> /rebuild  
sourceanalyzer -b <build_id> msbuild /t:rebuild <solution_or_project_<br/>file>
```

- Use touchless MSBuild integration to run Fortify Static Code Analyzer as part of your MSBuild project. For more information, see "[Using the Touchless MSBuild Integration](#)" on page 79.

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 Technologies

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	41
Skipping Translation of JavaScript Library Files	41
Translating JavaScript Projects with HTML Files	42
Including External JavaScript or HTML in the Translation	43
Translating AngularJS Code	43
Scanning JavaScript Technologies	44

Translating Pure JavaScript Projects

The basic command-line syntax to translate JavaScript is:

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

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

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 90](#).

Translating JavaScript Projects with HTML Files

If the project contains HTML files in addition to JavaScript files, set the `com.fortify.sca.EnableDOMModeling` property to `true` in the `fortify-sca.properties` file or on the command line as 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`, this can decrease false negative reports of DOM-related attacks, such as DOM-related cross-site scripting issues.

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

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

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

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

Including External JavaScript or HTML in the Translation

To include external JavaScript or HTML files that are specified with the `src` attribute, you can 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

Fortify Static Code Analyzer supports the following application architectures in the analysis of AngularJS code:

- MVC
- Partial
- 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 translate AngularJS, set the following property in the `fortify-sca.properties` file to true (or include this property setting on the command line with the `-D` option):

```
com.fortify.sca.EnabledDOMModeling=true
```

To scan AngularJS, the property `com.fortify.sca.hoa.Enable` must be set to true.

Scanning JavaScript Technologies

You can configure the Higher Order Analyzer for JavaScript and TypeScript analysis, which improves the ability to track dataflow through higher-order code. However, adding analysis of JavaScript and TypeScript 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
```

To scan TypeScript code, include `typescript` for the value of the `com.fortify.sca.Phase0HigherOrder.Languages` property as follows:

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

Chapter 7: Translating Python Code

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

This section contains the following topics:

Python Translation Command-Line Syntax	45
Including Import Files	45
Including Namespace Packages	46
Using the Django Framework with Python	46
Python Command-Line Options	46
Python Command-Line Examples	47

Python Translation Command-Line Syntax

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

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

Including Import Files

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.

Including Namespace Packages

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

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

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

Using the Django Framework with Python

Fortify Static Code Analyzer supports the Django Framework.

To translate code created using the Django framework, add the following properties to the `<sc_a_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
<code>-python-version <version></code>	Specifies the Python source code version you want to scan. The valid values for <code><version></code> are 2 and 3. The default value is 2. Equivalent property name: <code>com.fortify.sca.PythonVersion</code>
<code>-python-path <paths></code>	Specifies a colon-separated (non-Windows) or semicolon-separated (Windows) list of additional import directories. You can use the <code>-python-path</code> option to specify all paths used to import packages or modules. Include all paths to namespace package directories with this option. Equivalent property name: <code>com.fortify.sca.PythonPath</code>

Python Option	Description
<code>-python-legacy</code>	<p>Specifies to translate python 2 code with the legacy Python translator. You should only use this option if the translation without the option fails and Micro Focus Fortify Customer Support has recommended that you use it.</p> <p>Important! Do not use this option to translate python 3 code (meaning do not include <code>-python-version 3</code> with this option).</p> <p>Equivalent property name: <code>com.fortify.sca.PythonLegacy</code></p>
<code>-django-template-dirs</code> <code><template_path></code>	<p>Specifies to translate code created with the Django framework where <code><template_path></code> is a colon-separated (non-Windows) or semicolon-separated (Windows) list of Django template directories.</p> <p>Equivalent property name: <code>com.fortify.sca.DjangoTemplateDirs</code></p>

Python Command-Line Examples

To translate Python 3 code, type:

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

To translate Python 2 code, type:

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

Chapter 8: 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](#) 48

[Translating Android Projects](#) 49

Translating Apple iOS Projects

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

iOS Project Translation Prerequisites

The following are the prerequisites for translating iOS projects:

- Objective-C++ projects must use the non-fragile Objective-C runtime (ABI version 2 or 3).
- Use Apple’s `xcode-select` command-line tool to set your Xcode path. Fortify Static Code Analyzer uses the system global Xcode configuration to find the Xcode toolchain and headers.
- Make sure that you have available any dependencies required to build the project.
- To translate Swift code, make sure that you have available all third-party modules, including CocoaPods. Bridging headers must also be available. However, Xcode usually generates them automatically during the build.
- To translate Objective-C projects, ensure that the headers for third-party libraries are available.
- To translate WatchKit applications, make sure that you translate both the iPhone application target and the WatchKit extension target.

iOS Code Analysis Command-Line Syntax

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

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

where *<compiler_options>* are the supported options that are passed to the Xcode compiler.

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

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

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

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

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

Translating Android Projects

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

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

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

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

Android Project Translation Prerequisites

The following are the prerequisites for translating Android projects:

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

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

Use the same version of Gradle that is provided with the version of Android Studio that you use to create your Android project

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

Android Code Analysis Command-Line Syntax

You use Gradlew to scan Android code, which is similar to using Gradle as described in ["Gradle Integration" on page 75](#) except that you use the Gradle Wrapper.

The command-line syntax to translate Android code is:

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

where *<options>* are the supported gradlew options such as `build` or `build clean`.

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

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

Filtering Issues Detected in Android Layout Files

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

Fortify recommends that Issues reported in any layout file be included in your standard audit so you can carefully determine if any of them are false positives. After you identify issues in layout files that you are not interested in, you can filter them out as described in ["Filtering the Analysis" on page 117](#). You can filter out the issues based on the Instance ID.

Chapter 9: Translating Ruby Code

This section contains the following topics:

- [Ruby Command-Line Syntax](#) 51
- [Adding Libraries](#) 52
- [Adding Gem Paths](#) 52

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 10: Translating Apex and Visualforce Code

This section contains the following topics:

- [Apex Translation Prerequisites](#) 53
- [Apex and Visualforce Command-Line Syntax](#) 53
- [Apex and Visualforce Command-Line Options](#) 54
- [Downloading Customized Salesforce Database Structure Information](#) 54

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	<p>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.</p> <p>Note: Alternatively, you can set the <code>com.fortify.sca.fileextension.cls</code> property to APEX either on the command line (include <code>-Dcom.fortify.sca.fileextensions.cls=APEX</code>) or in the <code><sca_install_dir>/Core/config/fortify-sca.properties</code> file.</p> <p>Equivalent property name: <code>com.fortify.sca.Apex</code></p>
-apex-subject-path <path>	<p>Specifies the location of the custom sObject JSON file <code>subjects.json</code>.</p> <p>For instructions on how to use the <code>sf_extractor</code> tool, see "Downloading Customized Salesforce Database Structure Information" below.</p> <p>Equivalent property name: <code>com.fortify.sca.ApexObjectPath</code></p>

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

Folder or File Name	Description
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 subjects.json file that you provide to Fortify Static Code Analyzer in the translation command using the -apex-object-path option.

Chapter 11: Translating COBOL Code

This section contains the following topics:

Preparing COBOL Source Files for Translation	56
COBOL Command-Line Syntax	57
COBOL Command-Line Options	57

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 F00
```

where F00 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 F00. 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 `sourceanalyzer` 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
<code>-copy-extensions <ext></code>	Specifies one or more colon-separated copybook file extensions.

COBOL Option	Description
<code>-copydirs <path></code>	Directs Fortify Static Code Analyzer to search a list of colon-separated paths for copybooks and SQL INCLUDE files.
<code>-fixed-format</code>	<p>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.</p> <p>If your COBOL code is IBM Enterprise COBOL, then it is most likely fixed-format. The following are indications that you might need the <code>-fixed-format</code> option:</p> <ul style="list-style-type: none">• The COBOL translation appears to hang indefinitely• Fortify Static Code Analyzer reports a lot of parsing errors in the COBOL translation <p>Equivalent property name: <code>com.fortify.sca.CobolFixedFormat</code></p>

Chapter 12: Translating Other Languages

This section contains the following topics:

- Translating PHP Code 59
- Translating ABAP Code 60
- Translating Flex and ActionScript 64
- Translating ColdFusion Code 66
- Translating SQL 67
- Translating Scala Code 68
- Translating ASP/VBScript Virtual Roots 68
- Classic ASP Command-Line Example 70
- VBScript Command-Line Example 71

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 <i>Micro Focus Fortify Software System Requirements</i> .

PHP Option	Description
	Equivalent property name: <code>com.fortify.sca.PHPVersion</code>

Translating ABAP Code

Translating ABAP code is similar to translating other operating language code. However, it requires additional steps to extract the code from the SAP database and prepare it for scanning. See "[Importing the Transport Request](#)" on the next page for more information. This section assumes you have a basic understanding of SAP and ABAP.

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

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

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

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

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

Each `READ REPORT` call produces a file in the temporary folder on the local system. This set of files is what Fortify Static Code Analyzer translates and scans, producing an FPR file that you can open with 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:

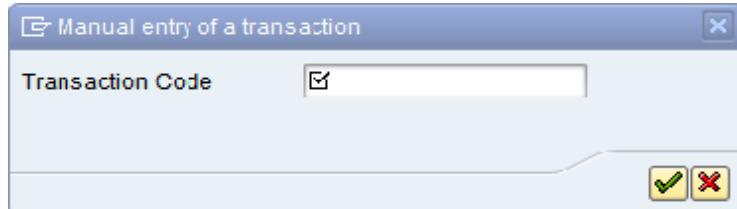
1. Run the transport request import again.
The Import Transport Request dialog box opens.
2. Click the **Options** tab.
3. Select the **Ignore Invalid Component Version** check box.
4. Complete the import procedure.


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.

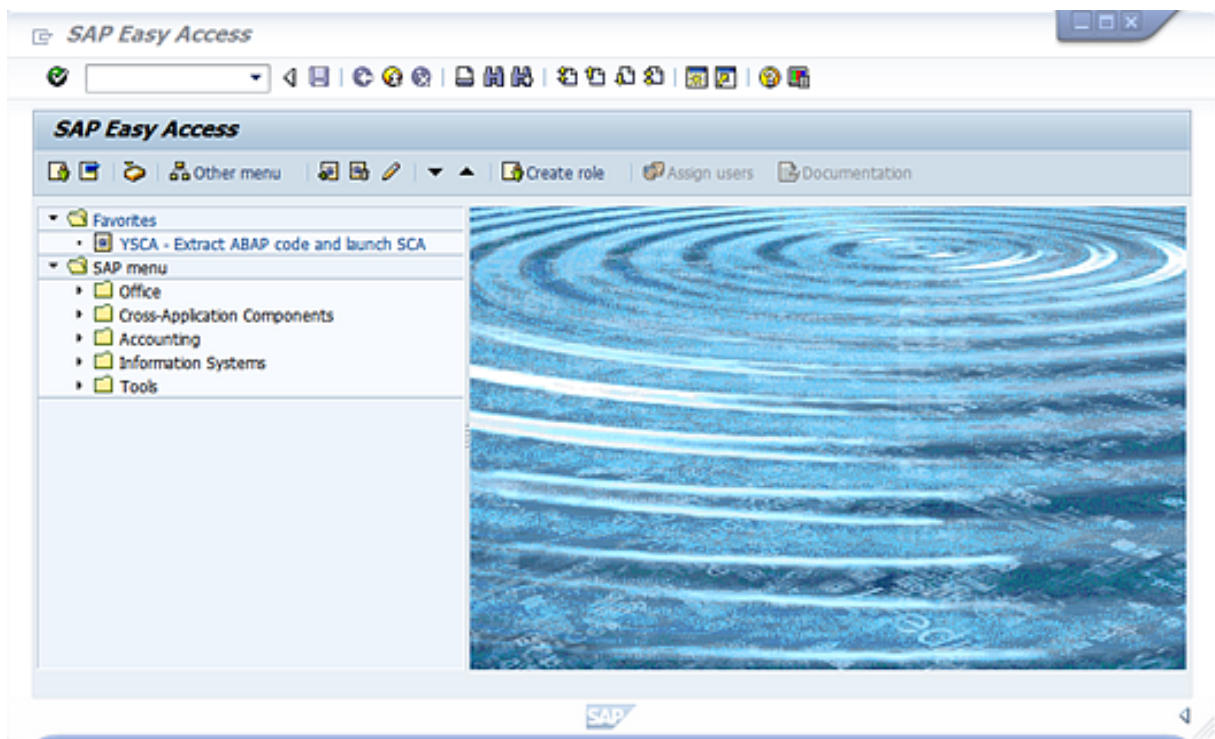
1. From the **SAP Easy Access** menu, type `S000` in the transaction box.
The **SAP Menu** opens.

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	<p>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.</p> <p>Working Directory: Type the directory where you want the extracted source code copied.</p> <p>Build-ID: Type the build ID for the scan.</p> <p>Translation Parameters: Type any optional Fortify Static Code Analyzer comand-line translation options.</p> <p>Scan Parameters: Type any optional Fortify Static Code Analyzer command-line scan options.</p> <p>ZIP File Name: Type a ZIP file name if you want your output in a compressed package.</p> <p>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.</p>
Actions	<p>Download: Select this check box to have Fortify Static Code Analyzer download the source code extracted from your SAP database.</p> <p>Build: Select this check box to have Fortify Static Code Analyzer translate all downloaded ABAP code and store it using the specified build ID.</p> <p>Scan: Select this check box to request a scan.</p> <p>Launch AWB: Select this check box to start Audit Workbench and load the FPR.</p> <p>Create ZIP: Select this check box to compress the output.</p> <p>Process in Background: Select this check box to have processing occur in the background.</p>

3. Click **Execute**.

Uninstalling the Fortify ABAP Extractor

To uninstall the ABAP extractor:

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

Translating Flex and ActionScript

The basic command-line syntax for translating ActionScript is:

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

where:

<libs> is a semicolon-separated 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	<p>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.</p> <p>Equivalent property name: com.fortify.sca.FlexSdkRoot</p>
-flex-libraries	<p>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).</p> <p>Note: You can specify SWC or SWF files as Flex libraries (SWZ is not currently supported).</p> <p>Equivalent property name: com.fortify.sca.FlexLibraries</p>
-flex-source-roots	<p>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).</p> <p>Equivalent property name: com.fortify.sca.FlexSourceRoots</p>

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.

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_<br>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 96](#).

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
<code>-source-base-dir <web_app_root_dir> <files> <file_specifiers></code>	The web application root directory. Equivalent property name: <code>com.fortify.sca.SourceBaseDir</code>

Translating SQL

On Windows platforms, Fortify Static Code Analyzer assumes that files with the `.sql` extension are T-SQL rather than PL/SQL. If you have PL/SQL files with the `.sql` extension on Windows, you must configure Fortify Static Code Analyzer to treat them as PL/SQL.

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

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

or

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

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

PL/SQL Command-Line Example

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

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

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

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

T-SQL Command-Line Example

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

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

The following example shows how to translate all T-SQL files in the sources directory:

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

Note: This example assumes the `com.fortify.sca.fileextensions.sql` property in `fortify-sca.properties` is set to `TSQL`.

Translating Scala Code

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 `<scd_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 13: Integrating into a Build

You can integrate the analysis into supported build tools.

This section contains the following topics:

Build Integration	72
Modifying a Build Script to Invoke Fortify Static Code Analyzer	73
Touchless Build Integration	74
Ant Integration	74
Gradle Integration	75
Maven Integration	75
MSBuild Integration	78

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 ["iOS Code Analysis 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_tool>` 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 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: Fortify Static Code Analyzer converts devenv invocations and command-line options to MSBuild invocations. For more information, see [".NET Command-Line Syntax" on page 33](#).

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 72](#)) 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	<code><root_dir>/docs/usage.html</code>
Source	<code><root_dir>/sca-maven-plugin/target/site/usage.html</code>

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 by prepending the maven command used to build your project with the sourceanalyzer command as follows:

```
sourceanalyzer -b <build_id> [<sca_options>] [<mvn_command_with_options>]
```

For example:

```
sourceanalyzer -b buildxyz mvn package
```

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 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 types:

- C/C++ console applications
- C/C++ libraries
- .NET, .NET Core, .NET Standard, and .NET Portable projects:
 - Libraries
 - Console applications
 - Websites
 - ASP.NET, ASP.NET Core web applications
- .NET Azure applications
- .NET Xamarin applications

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

Using MSBuild Integration

The simplest way to translate your MSBuild project is to append the MSBuild command that you normally run to build your project to the Fortify Static Code Analyzer command line. The following is an example of this command line.

```
sourceanalyzer -b <build_id> msbuild [<sca_options>] <msbuild_options>  
<solution_or_project_file>
```

You can provide a single project file or the entire solution for translation. For most project types, you can run this command line from either the Windows Command Prompt or the Developer Command Prompt for Visual Studio. The only exceptions are Xamarin and C/C++ projects and solutions, which you must translate from Developer Command Prompt for Visual Studio.

Using the Touchless MSBuild Integration

The touchless MSBuild integration is another method to integrate Fortify Static Code Analyzer into the MSBuild process. This method enables you to translate your MSBuild project without explicitly invoking Fortify Static Code Analyzer from the command line.

The touchless MSBuild integration requires the `FortifyMSBuildTouchless.dll` located in the `<sca_install_dir>/Core/lib` directory, which you provide to the MSBuild process with the `/logger` option. There are several Windows environment variables that you can set to configure the translation of your MSBuild projects. Most of these variables have default values. If the variables are not set, Fortify Static Code Analyzer uses the defaults.

Important! The `FORTIFY_MSBUILD_BUILDDID` variable does not have default value. Make sure that you set this variable before you use touchless MSBuild integration to translate your project.

The following table lists the Windows environment variables that you can set for touchless MSBuild integration.

Environment Variable	Description	Default Value
FORTIFY_MSBUILD_BUILDDID	Specifies the Fortify Static Code Analyzer build ID	None
FORTIFY_MSBUILD_DEBUG	Puts the logger and Fortify Static Code Analyzer into debug mode	False
FORTIFY_MSBUILD_DEBUG_VERBOSE	Puts Fortify Static Code Analyzer into verbose debug mode. In this mode, additional debug information is logged as compared to the standard debug mode set by the	False

Environment Variable	Description	Default Value
	FORTIFY_MSBUILD_DEBUG variable. Set this variable when you report translation failures or errors to Micro Focus Fortify Customer Support. <div style="background-color: #f0f0f0; padding: 5px; margin-top: 10px;"> Note: If both debug environment variables are set to true, this variable takes precedence over FORTIFY_MSBUILD_DEBUG. </div>	
FORTIFY_MSBUILD_MEM	Specifies the memory used to run 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 MSBuild log file	\${win32.LocalAppdata} /Fortify/MSBuildPlugin/Log/ MSBuildPlugin.log
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

For most project types, you can run the commands described in this section from either the Windows Command Prompt or the Developer Command Prompt for Visual Studio. You can provide a single project or the entire solution for the command.

Important! You must translate Xamarin and C/C++ projects from the Developer Command Prompt for Visual Studio.

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

```
msbuild <msbuild_options> /logger:"<sca_install_dir>\Core\lib\FortifyMSBuildTouchless.dll" <solution_or_project_file>
```

Adding Custom Tasks to your MSBuild Project

As an alternative to using MSBuild integration or touchless MSBuild integration (see ["Using MSBuild Integration" on the previous page](#) and ["Using the Touchless MSBuild Integration" on the previous page](#) respectively), 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" below](#) - Remove any existing temporary files for the specified build ID
- ["Fortify.TranslateTask" below](#) - Translate the source code
- ["Fortify.ScanTask" on page 84](#) - Scan the source files
- ["Fortify.SSCTask" on page 85](#) - Upload an FPR to Fortify Software Security Center
- ["Fortify.CloudScanTask" on page 87](#) - 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 ["Translating .NET Binaries" on](#)

page 34.

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 <code>DotNetVersion</code> , <code>DotNetCoreVersion</code> , <code>DotNetStandardVersion</code> , <code>XamarinAndroidVersion</code> , and <code>XamarinIOSVersion</code> parameters within the same task.	
DotNetVersion	The .NET framework version
DotNetCoreVersion	The .NET Core version
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 <code>References</code> 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 <code>TargetsFolder</code> parameter.

Parameter	Description
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 <i>code-behind files</i>)
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
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 <code>_MYTYPE</code> preprocessor symbol specified in the <code><MyType></code> tag in the project settings
VBRootFolder	(VB.NET and Silverlight 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.SSCTask.

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 14: 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 88
- Translation Options 90
- Analysis Options 91
- Other Options 94
- Directives 95
- Specifying Files 96

Output Options

The following table describes the output options.

Output Option	Description
-f <file> -output-file <file>	Specifies the file to which results are written. If you do not specify an output file, Fortify Static Code Analyzer writes the output to the terminal. Equivalent property name: com.fortify.sca.ResultsFile
-format <format>	Controls the output format. Valid options are fpr, fvd1, 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 <i>Micro Focus Fortify Audit Workbench User Guide</i> 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 fvd1. For information on the -format output option, see the description in this table.

Output Option	Description
	<p>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 <code>-append</code> option, Fortify does not certify results generated with <code>-append</code>.</p> <p>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.</p> <p>In general, only use the <code>-append</code> option when it is not possible to analyze an entire application at once.</p> <p>Equivalent property name: <code>com.fortify.sca.OutputAppend</code></p>
<code>-disable-source-bundling</code>	<p>Excludes source files from the FPR file.</p> <p>Equivalent property name: <code>com.fortify.sca.FPRDisableSourceBundling</code></p>
<code>-build-label <label></code>	<p>Specifies the label of the project being scanned. Fortify Static Code Analyzer does not use this label but includes it in the analysis results.</p> <p>Equivalent property name: <code>com.fortify.sca.BuildLabel</code></p>
<code>-build-project <project></code>	<p>Specifies the name of the project being scanned. Fortify Static Code Analyzer does not use the name but includes it in the analysis results.</p> <p>Equivalent property name: <code>com.fortify.sca.BuildProject</code></p>
<code>-build-version <version></code>	<p>Specifies the version of the project being scanned. Fortify Static Code Analyzer does not use the version but includes it in the analysis results.</p> <p>Equivalent property name: <code>com.fortify.sca.BuildVersion</code></p>

Translation Options

The following table describes the translation options.

Translation Option	Description
<code>-b <build_id></code>	<p>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.</p> <p>Equivalent property name: <code>com.fortify.sca.BuildID</code></p>
<code>-exclude <file_specifiers></code>	<p>Removes files from the list of files to translate. See "Specifying Files" on page 96 for more information on how to use file specifiers.</p> <p>For example:</p> <pre>sourceanalyzer -cp "**/*.jar" "**/*" -exclude "**/Test/*.java"</pre> <p>This example excludes all Java files in any Test sub-directory.</p> <p>Equivalent property name: <code>com.fortify.sca.exclude</code></p>
<code>-encoding <encoding_name></code>	<p>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 <code>-encoding</code> 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.</p> <p>Valid encoding names are from the <code>java.nio.charset.Charset</code>. Typically, if you do not specify the encoding type, Fortify Static Code Analyzer uses <code>file.encoding</code> from the <code>java.io.InputStreamReader</code> constructor with no encoding parameter. In a few cases (for example with the ActionScript parser), Fortify Static Code Analyzer defaults to UTF-8.</p> <p>Equivalent property name: <code>com.fortify.sca.InputFileEncoding</code></p>
<code>-nc</code>	<p>When specified before a compiler command line, Fortify Static Code Analyzer translates the source file but does not run the compiler.</p>

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, TYPESCRIPT, 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
<p><code>-disable-default-rule-type <type></code></p>	<p>Disables all rules of the specified type in the default Rulepacks. You can use this option multiple times to specify multiple rule types.</p> <p>The <code><type></code> parameter is the XML tag minus the suffix Rule. For example, use <code>DataflowSource</code> for <code>DataflowSourceRule</code> elements. You can also specify specific sections of characterization rules, such as <code>Characterization:ControlFlow</code>, <code>Characterization:Issue</code>, and <code>Characterization:Generic</code>.</p> <p>The <code><type></code> parameter is case-insensitive.</p>
<p><code>-exit-code-level</code></p>	<p>Extends the default exit code options. See "Exit Codes" on page 113 for a description of the exit codes. The valid values are:</p> <p>The valid values are:</p> <ul style="list-style-type: none"> • <code>nothing</code>—Returns exit codes 0, 1, 2, or 3. This is the default setting. • <code>warnings</code>—Returns exit codes 0, 1, 2, 3, 4, or 5. • <code>errors</code>—Returns exit codes 0, 1, 2, 3, or 5. • <code>no_output_file</code>—Returns exit codes 0, 1, 2, 3, or 6. <p>Equivalent property name: <code>com.fortify.sca.ExitCodeLevel</code></p>
<p><code>-filter <file></code></p>	<p>Specifies a results filter file. See "Filtering the Analysis" on page 117 for more information about this option.</p> <p>Equivalent property name: <code>com.fortify.sca.FilterFile</code></p>
<p><code>-findbugs</code></p>	<p>Enables FindBugs analysis for Java code. You must specify the Java class directories with the <code>-java-build-dir</code> option, which is described in "Java Command-Line Options" on page 25.</p> <p>Equivalent property name: <code>com.fortify.sca.EnableFindbugs</code></p>
<p><code>-incremental-base</code></p>	<p>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 <code>-incremental</code> option. See "Incremental Analysis" on page 22 for more information about performing incremental analysis.</p> <p>Equivalent property name: <code>com.fortify.sca.IncrementalBaseScan</code></p>

Analysis Option	Description
-incremental	<p>Specifies that this is a subsequent scan of a project for which you have already run a full base scan with the <code>-incremental-base</code> option. See "Incremental Analysis" on page 22 for more information about performing incremental analysis.</p> <p>Equivalent property name: <code>com.fortify.sca.IncrementalScan</code></p>
-no-default-issue-rules	<p>Disables rules in default Rulepacks that lead directly to issues. Still loads rules that characterize the behavior of functions.</p> <p>Note: This is equivalent to disabling the following rule types: DataflowSink, Semantic, Controlflow, Structural, Configuration, Content, Statistical, Internal, and Characterization:Issue.</p> <p>Equivalent property name: <code>com.fortify.sca.NoDefaultIssueRules</code></p>
-no-default-rules	<p>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.</p> <p>Equivalent property name: <code>com.fortify.sca.NoDefaultRules</code></p>
-no-default-source-rules	<p>Disables source rules in the default Rulepacks.</p> <p>Note: Characterization source rules are not disabled.</p> <p>Equivalent property name: <code>com.fortify.sca.NoDefaultSourceRules</code></p>
-no-default-sink-rules	<p>Disables sink rules in the default Rulepacks.</p> <p>Note: Characterization sink rules are not disabled.</p> <p>Equivalent property name: <code>com.fortify.sca.NoDefaultSinkRules</code></p>
-project-template	<p>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.</p> <p>Equivalent property name: <code>com.fortify.sca.ProjectTemplate</code></p>
-rules <file> <dir>	<p>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</p>

Analysis Option	Description
	the .bin and .xml extensions. 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. Note: The file must be ASCII or UTF-8 encoded.
-h -? -help	Prints a summary of command-line options.
-debug	Includes debug information in the Fortify Support log file, which is only useful for Micro Focus Fortify Customer Support to help troubleshoot. Equivalent property name: com.fortify.sca.Debug
-debug-verbose	This is the same as the -debug option, but it includes more details, specifically for parse errors. Equivalent property name: com.fortify.sca.DebugVerbose
-verbose	Sends verbose status messages to the console and to the Fortify Support log file. Equivalent property name: com.fortify.sca.Verbose
-logfile <file>	Specifies the log file that Fortify Static Code Analyzer creates. Equivalent property name: com.fortify.sca.LogFile
-clobber-log	Directs Fortify Static Code Analyzer to overwrite the log file for each run of sourceanalyzer. Without this option, Fortify Static Code Analyzer appends information to the log file. Equivalent property name: com.fortify.sca.ClobberLogFile

Other Option	Description
-quiet	Disables the command-line progress information. Equivalent property name: com.fortify.sca.Quiet
-version -v	Displays the Fortify Static Code Analyzer version number.
-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, and the assumption that no other memory intensive processes are running, do not allocate more than 2/3 of the available memory. Note: Specifying this option overrides the default memory allocation you would get with the -autoheap option.

Directives

Use 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.

Directive	Description
	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 different file specifier formats examples.

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 <dir>).

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 15: Command-Line Utilities

This section contains the following topics:

Fortify Static Code Analyzer Utilities	98
Checking the Fortify Static Code Analyzer Scan Status	99
Working with FPR Files from the Command Line	101
Generating Reports from the Command Line	108
About Updating Security Content	111

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: <ul style="list-style-type: none">• 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 101
BIRTReportGenerator and ReportGenerator	Generates BIRT reports and legacy reports from FPR files	"Generating Reports from the Command Line" on page 108

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 111

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 <i>Micro Focus Fortify Software Security Center User Guide</i>
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 <i>Micro Focus Fortify Static Code Analyzer Installation Guide</i>

Checking the Fortify Static Code Analyzer Scan Status

Use the SCASState 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:

```
SCASState [<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>	<p>Specifies the currently running Fortify Static Code Analyzer process ID. Use this option if there are multiple Fortify Static Code Analyzer processes running simultaneously.</p> <p>To obtain the process ID on Windows systems:</p> <ol style="list-style-type: none">1. Open a command window.2. Type <code>tasklist</code> at the command prompt. A list of processes is displayed.3. Find the <code>java.exe</code> process in the list and note its PID. <p>To find the process ID on Linux or Unix systems:</p> <ul style="list-style-type: none">• Type <code>ps aux grep sourceanalyzer</code> 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 SCASState 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, SCASState 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 SCASState 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 103](#)
- ["Extracting a Source Archive from an FPR File" on page 106](#)
- ["Allocating More Memory for FPRUtility" on page 107](#)

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
<code>-merge</code>	Merges the specified project and source FPR files.
<code>-project <primary>.fpr</code>	Specifies the primary FPR file to merge. Conflicts are resolved using the values in this file.
<code>-source <secondary>.fpr</code>	Specifies the secondary FPR file to merge. The primary project overrides values if conflicts exist.
<code>-f <output>.fpr</code>	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.
<code>-forceMigration</code>	Forces the migration, even if the engine and the Rulepack versions of the two projects are the same.
<code>-useMigrationFile <mapping_file></code>	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.
<code>-useSourceIssueTemplate</code>	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
<code>-iidmigratorOptions</code> <code><iidmigrator_options></code>	<p>Specifies instance ID migrator options. Separate included options with spaces and enclosed them in quotes. Some valid options are:</p> <ul style="list-style-type: none">• <code>-i</code> provides a case-sensitive file name comparison of the merged files• <code>-u <scheme_file></code> tells iidmigrator to read the matching scheme from <code><scheme_file></code> for instance ID migration <p>Note: Wrap <code>-iidmigrator</code> options in single quotes (<code>'-u <scheme_file>'</code>) when working from a Cygwin command prompt.</p> <p>Windows example:</p> <pre>FPRUtility -merge -project primary.fpr -source secondary.fpr -f output.fpr -iidmigratorOptions "-u scheme_file"</pre>

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] \
[-includeSuppressed] [-includeRemoved] \
-project <project>.fpr -f <output>.txt
```

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

To display information for issues in CSV format:

```
FPRUtility -information -listIssues \
-search [-queryAll | -query "search expression"] \
[-categoryIssueCounts] [-analyzerIssueCounts] \
[-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
<code>-information</code>	Displays information for the project.
One of:	The <code>-signature</code> option displays the signature.
<code>-signature</code>	The <code>-mappings</code> option displays the migration mappings report.
<code>-mappings</code>	The <code>-errors</code> option displays a full error report for the FPR.
<code>-errors</code>	The <code>-versions</code> option displays the engine version and the Rulepack version used in the static scan.
<code>-versions</code>	The <code>-functionsMeta</code> option displays all functions that the static analyzer encountered in CSV format. To filter which functions are displayed, include <code>-excludeCoveredByRules</code> , and <code>-excludeFunctionsWithoutSource</code> .
<code>-functionsMeta</code>	
<code>-categoryIssueCounts</code>	
<code>-analyzerIssueCounts</code>	
<code>-search -query <search_expression></code>	
<code>-search -queryAll</code>	

Option	Description
	<p>The <code>-categoryIssueCounts</code> option displays the number of issues for each vulnerability category.</p> <p>The <code>-analyzerIssueCounts</code> option displays the number of issues for each analyzer.</p> <p>The <code>-search -query</code> 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 <code>-categoryIssueCounts</code> and <code>-analyzerIssueCounts</code> options to the search option. Use the <code>-includeSuppressed</code> and <code>-includeRemoved</code> options to include suppressed or removed issues.</p> <p>The <code>-search -queryAll</code> searches all the issues in the FPR, including suppressed and removed issues.</p>
<code>-project <project>.fpr</code>	Specifies the FPR from which to extract the results information.
<code>-f <output></code>	Specifies the output file. The default is <code>System.out</code> .
<code>-outputformat <format></code>	Specifies the output format. The valid values are <code>TEXT</code> and <code>CSV</code> . The default value is <code>TEXT</code> .
<code>-listIssues</code>	<p>Displays the location for each issue in one of the following formats:</p> <pre data-bbox="760 1255 1401 1377"><sink_filename>:<line_num> or <sink_filename>:<line_num> (<category> <analyzer>)</pre> <p>You can also use the <code>-listIssues</code> option with <code>-search</code> and with both <code>issueCounts</code> grouping options. If you group by <code>-categoryIssueCounts</code>, then the output includes (<code><analyzer></code>) and if you group by <code>-analyzerIssueCounts</code>, then the output includes (<code><category></code>).</p> <p>If you specify the <code>-outputFormat CSV</code>, then each issue is displayed as a line in the format:</p> <pre data-bbox="760 1724 1401 1860"><instanceid>", "<category>", "<sink_filename>:<line_num>", "<analyzer>"</pre>

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 CCI 2", "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>	<p>Specifies the version for the template. The valid values for the templates versions are listed below.</p> <div data-bbox="699 583 1398 800" style="background-color: #f0f0f0; padding: 5px;"> <p>Notes:</p> <ul style="list-style-type: none"> • 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. </div> <p>For the "CWE/SANS Top 25" template:</p> <p>"2011 CWE/SANS Top 25", "2010 CWE/SANS Top 25", and "2009 CWE/SANS Top 25"</p> <p>For the "DISA STIG" template:</p> <p>"DISA STIG 4.7", "DISA STIG 4.6", "DISA STIG 4.5", "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"</p> <p>For the "OWASP Top 10" template:</p> <p>"OWASP Top 10 2017", "OWASP Top 10 2013", "OWASP Top 10 2010", "OWASP Top 10 2007", and "OWASP Top 10 2004"</p> <p>For the "PCI DSS Compliance" template:</p> <p>"3.2 Compliance", "3.1 Compliance", "3.0 Compliance" and "2.0 Compliance"</p>
--IncludeDescOfKeyTerminology	Include the <i>Description of Key Terminology</i> section in the report.
--IncludeAboutFortify	Include the <i>About Fortify Solutions</i> 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_<br>project>.fpr
```

To generate an XML report, type the following command:

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

Option	Description
<code>-proxyport <port></code>	Specifies a proxy server port number.
<code>-proxyUsername <username></code>	If the proxy server requires authentication, specifies the user name.
<code>-proxyPassword <password></code>	If the proxy server requires authentication, specifies the password.
<code>-showInstalledRules</code>	Displays the currently installed Rulepacks including any custom rules or metadata.
<code>-showInstalledExternalMetadata</code>	Displays the currently installed external metadata.
<code>-url <url></code>	<p>Specifies a URL from which to download the security content. The default URL is <code>https://update.fortify.com</code> or the value set for the <code>rulepackupdate.server</code> property in the <code>server.properties</code> configuration file.</p> <p>For more information about the <code>server.properties</code> configuration file, see the <i>Micro Focus Fortify Static Code Analyzer Tools Properties Reference Guide</i>.</p>

Chapter 16: Troubleshooting

This section contains the following topics:

- Exit Codes 113
- Translation Failed Message 114
- Issue Non-Determinism 114
- C/C++ Precompiled Header Files 114
- Accessing Log Files 115
- Configuring Log Files 115
- Reporting Issues and Requesting Enhancements 116

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.

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
```

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.

Accessing Log Files

By default, Fortify Static Code Analyzer creates two log files in the following location:

- 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.

The following table describes the two log files.

Default File Name	Description
sca.log	The standard log provides a log of informational messages, warnings, and errors that occurred in the run of sourceanalyzer.
sca_FortifySupport.log	The Fortify Support log provides: <ul style="list-style-type: none">• The same log messages as the standard log file, but with additional details• Additional detailed messages that are not included in the standard log file This log file is only helpful to Micro Focus Fortify Customer Support or the development team to troubleshoot any possible issues.

If you encounter warnings or errors that you cannot resolve, provide the Fortify Support log file to Micro Focus Fortify Customer Support.

Configuring Log Files

You can configure the information that Fortify Static Code Analyzer writes to the log files by setting logging properties (see "[fortify-sca.properties](#)" on page 128). You can configure the following log file settings:

- The location and name of the log file
Property: com.fortify.sca.LogFile
- Log level (see "[Understanding Log Levels](#)" on the next page)
Property: com.fortify.sca.LogLevel
- Whether to overwrite the log files for each run of sourceanalyzer

Property: `com.fortify.sca.ClobberLog`

Command-line option: `-clobber-log`

Understanding Log Levels

The log level you select gives you all log messages equal to and greater than it. The log levels in the following table are listed in order from least to greatest. For example, the default log level of INFO includes log messages with the following levels: INFO, WARN, ERROR, and FATAL. You can set the log level with the `com.fortify.sca.LogLevel` property in the `<sca_install_dir>/Core/config/fortify.sca.properties` file or on the command-line using the `-D` option. The default log level is INFO.

Log Level	Description
DEBUG	Includes information that could be used by Micro Focus Fortify Customer Support or the development team to troubleshoot an issue
INFO	Basic information about the translation or scan process
WARN	Information about issues where the translation or scan did not stop, but might require your attention for accurate results
ERROR	Information about an issue that might require attention
FATAL	Information about an error that caused the translation or scan to abort

Reporting Issues and Requesting Enhancements

Feedback is critical to the success of this product. To request enhancements or patches, or to report issues, visit Micro Focus Fortify Customer Support at <https://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	117
Filter File Example	117

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)
```

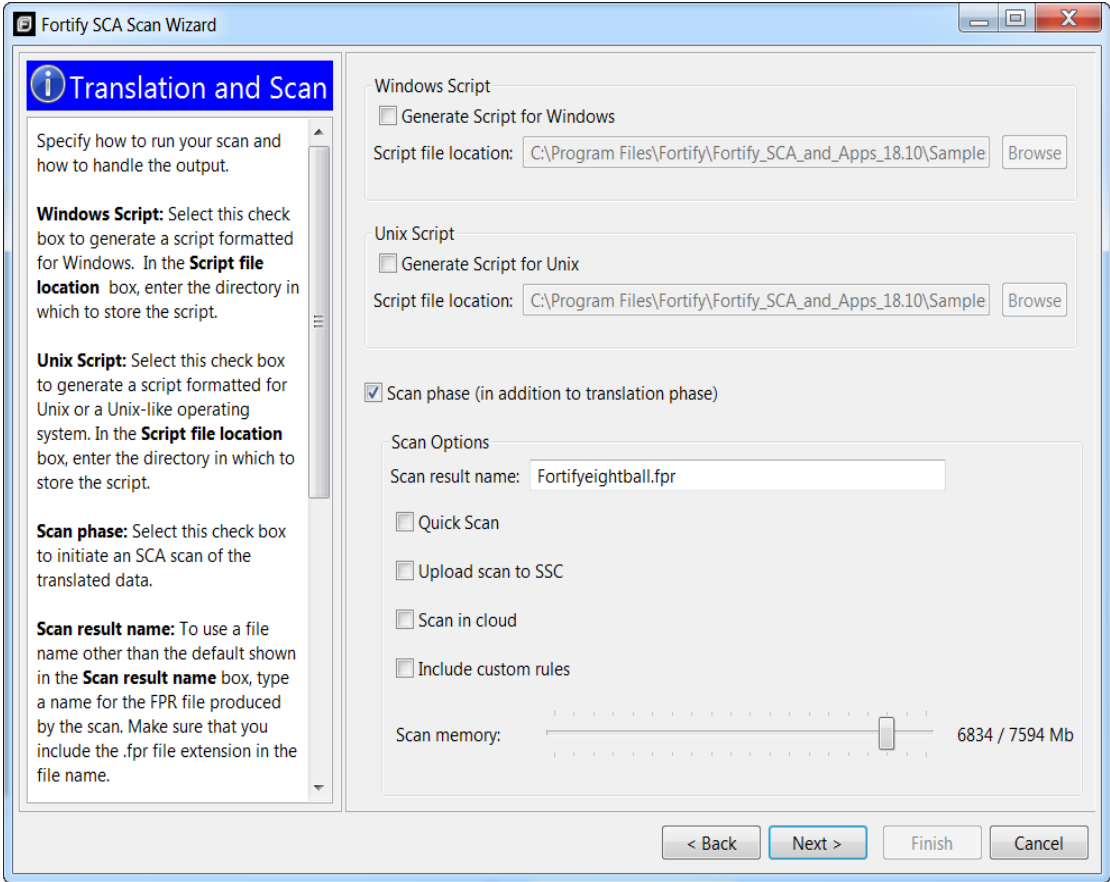
Appendix B: Scan Wizard

This section contains the following topics:

- Preparing to use the Scan Wizard120
- Starting the Scan Wizard121

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](#)123
- [Advanced Samples](#)125

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 <code>database.pks</code> sample file. This SQL sample includes issues in SQL code.	Access Control: Database

Folder Name	Description	Vulnerabilities
eightball	A Java application (<code>EightBall.java</code>) 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 <code>formatstring.c</code> file. It requires a gcc or cl compiler.	Format String
javascript	The <code>sample.js</code> JavaScript file.	Cross Site Scripting (XSS) Open Redirect
nullpointer	The <code>NullPointerSample.java</code> file.	Null Dereference
php	Two PHP files: <code>sink.php</code> and <code>source.php</code> . Analyzing <code>source.php</code> reveals simple dataflow vulnerabilities and a dangerous function.	Cross Site Scripting SQL Injection
sampleOutput	A sample output file (<code>WebGoat5.0.fpr</code>) from the WebGoat project located in the <code>Samples/advanced/webgoat</code> directory.	Example input for Audit Workbench.
stackbuffer	The <code>stackbuffer.c</code> file. It requires a gcc or cl compiler.	Buffer Overflow
toctou	The <code>toctou.c</code> file.	Time-of-Check/Time-of-Use (Race Condition)
vb6	The <code>command-injection.bas</code> file.	Command Injection SQL Injection
vbscript	The <code>source.asp</code> and <code>sink.asp</code> files.	SQL Injection

Advanced Samples

The following table describes the sample files in the `<sc_a_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++	<p>A sample solution for different supported versions of Visual Studio.</p> <p>To use this sample, you must have the following installed:</p> <ul style="list-style-type: none">• A supported version of Visual Studio Visual C/C++• Fortify Static Code Analyzer• To analyze the sample from Visual Studio as described in the <code>README.txt</code> file, you must have the Micro Focus Fortify Extension for Visual Studio installed for your Visual Studio version <p>The code includes a Command Injection issue and an Unchecked Return Value issue.</p>
configuration	A sample Java EE application that has vulnerabilities in its web module deployment descriptor <code>web.xml</code> .
crossier	<p>A sample that has vulnerabilities that span multiple application technologies (Java, PL/SQL, JSP, struts).</p> <p>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 <code>SELECT</code> statement in PL/SQL.</p>
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 <code>-filter</code> option.

Folder Name	Description
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: <code>ResourceInjection.java</code> . The result file should include a Path Manipulation, a J2EE Bad Practices, and a Poor Style vulnerability.
javaAnnotations	<p>A sample application that illustrates problems that might arise from its use and how to fix the problems with the Fortify Java Annotations.</p> <p>This example illustrates how the use of Fortify Annotations can result in increased accuracy in the reported vulnerabilities. The <code>README.txt</code> file describes the Fortify Java Annotations and the potential problems and solutions associated with the sample application.</p>
JavaDoc	JavaDoc directory for the <code>public-api</code> and <code>WSClient</code> .
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	127
fortify-sca.properties	128
fortify-sca-quickscan.properties	156

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
<code>fortify-sca.properties</code>	Defines the Fortify Static Code Analyzer configuration properties.
<code>fortify-sca-quickscan.properties</code>	Defines the configuration properties applicable for a Fortify Static Code Analyzer quick scan.

Properties File Format

In the properties file, each property consists of a pair of strings: the first string is the property name and the second string is the property value.

```
com.fortify.sca.fileextensions.htm=HTML
```

As shown above, the property sets the translation to use for `.htm` files. The property name is `com.fortify.sca.fileextension.htm` and the value is set to `HTML`.

Note: When you specify a path for Windows systems as the property value, you must escape any backslash character (`\`) with a backslash (for example:

```
com.fortify.sca.ASPVirtualRoots.Library=C:\\WebServer\\CustomerA\\inc).
```

Disabled properties are commented out of the properties file. To enable these properties, remove the comment symbol (#) and save the properties file. In the following example, the `com.fortify.sca.LogFile` property is disabled in the properties file and is not part of the configuration:

```
# default location for the log file  
#com.fortify.sca.LogFile=${com.fortify.sca.ProjectRoot}/sca/log/sca.log
```

Precedence of Setting Properties

Fortify Static Code Analyzer uses properties settings in a specific order. You can override any previously set properties with the values that you specify. Keep this order in mind when making changes to the properties files.

The following table lists the order of precedence for Fortify Static Code Analyzer properties.

Order	Property Specification	Description
1	Command line with the <code>-D</code> 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 (<code>fortify-sca-quickscan.properties</code>) have the second priority, but only if you include the <code>-quick</code> 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 (<code>fortify-sca.properties</code>) 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 156 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
<code>com.fortify.sca.BuildID</code>	Specifies the build ID of the build. Value type: String Default: (none) Command-line Option: <code>-b</code>

Property Name	Description
<p>com.fortify.sca. ProjectRoot</p>	<p>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.</p> <p>Value type: String (path)</p> <p>Default (Windows): <code>\${win32.LocalAppdata}\Fortify</code></p> <p>Note: <code>\${win32.LocalAppdata}</code> is a special variable that points to the windows Local Application Data shell folder.</p> <p>Default (Non-Windows): <code>\$home/.fortify</code></p> <p>Command-line Option: <code>-project-root</code></p> <p>Example: <code>com.fortify.sca.ProjectRoot=C:\Users\<i>user</i>\AppData\Local\</code></p>
<p>com.fortify.sca. DisableDeadCode Elimination</p>	<p>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.</p> <p>Value type: Boolean</p> <p>Default: <code>false</code></p>
<p>com.fortify.sca. DeadCodeFilter</p>	<p>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.</p> <p>Value type: Boolean</p> <p>Default: <code>true</code></p>
<p>com.fortify.sca. fileextensions.java com.fortify.sca. fileextensions.cs com.fortify.sca. fileextensions.js</p>	<p>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, TYPESCRIPT, VB, VB6,</p>

Property Name	Description
<p>com.fortify.sca.fileextensions.py</p> <p>com.fortify.sca.fileextensions.rb</p> <p>com.fortify.sca.fileextensions.aspx</p> <p>com.fortify.sca.fileextensions.php</p> <p>Note: This is a partial list. For the complete list, see the properties file.</p>	<p>VBSCRIPT, VISUAL_FORCE, and XML.</p> <p>Value type: String (valid language type)</p> <p>Default: See the fortify-sca.properties file for the complete list.</p> <p>Examples:</p> <pre>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</pre> <p>You can also specify a value of oracle: <i><path_to_script></i> 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.</p> <p>Example:</p> <pre>com.fortify.sca.fileextensions.jsp= oracle:<path_to_script></pre>
<p>com.fortify.sca.compilers.javac= com.fortify.sca.util.compilers.JavacCompiler</p> <p>com.fortify.sca.compilers.cplusplus= com.fortify.sca.util.compilers.GppCompiler</p>	<p>Specifies custom-named compilers.</p> <p>Value type: String (compiler)</p> <p>Default: See the Compilers section in the fortify-sca.properties file for the complete list.</p> <p>Example:</p> <p>To tell Fortify Static Code Analyzer that “my-gcc” is a gcc compiler:</p> <pre>com.fortify.sca.compilers.my-gcc= com.fortify.sca.util.compilers.GccCompiler</pre> <p>Notes:</p>

Property Name	Description
<p>com.fortify.sca.compilers.make= com.fortify.sca.util.compilers.TouchlessCompiler</p> <p>com.fortify.sca.compilers.mvn= com.fortify.sca.util.compilers.MavenAdapter</p> <p>Note: This is a partial list. For the complete list, see the properties file.</p>	<ul style="list-style-type: none"> • 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
<p>com.fortify.sca.UseAntListener</p>	<p>If set to true, Fortify Static Code Analyzer includes com.fortify.dev.ant.SCAListener in the compiler options.</p> <p>Value type: Boolean</p> <p>Default: false</p>
<p>com.fortify.sca.exclude</p>	<p>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).</p> <p>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.</p> <p>Value type: String (list of file names)</p> <p>Default: Not enabled</p> <p>Command-line Option: -exclude</p> <p>Example: com.fortify.sca.exclude=file1.x;file2.x</p>

Property Name	Description
<code>com.fortify.sca. InputFileEncoding</code>	<p>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 <code>-encoding</code> 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.</p> <p>Valid encoding names are from the <code>java.nio.charset.Charset</code></p> <p>Typically, if you do not specify the encoding type, Fortify Static Code Analyzer uses <code>file.encoding</code> from the <code>java.io.InputStreamReader</code> constructor with no encoding parameter. In a few cases (for example with the ActionScript parser), Fortify Static Code Analyzer defaults to UTF-8.</p> <p>Value type: String</p> <p>Default: (none)</p> <p>Command-Line Option: <code>-encoding</code></p> <p>Example: <code>com.fortify.sca.InputFileEncoding=UTF-16</code></p>
<code>com.fortify.sca. xcode.TranslateAfterError</code>	<p>Specifies whether the xcodebuild touchless adapter should continue translation if the xcodebuild subprocess exited with a non-zero exit code. If set to false, translation stops after encountering a non-zero xcodebuild exit code and the Fortify Static Code Analyzer touchless build halts with the same exit code. If set to true, the Fortify Static Code Analyzer touchless build executes translation of the build file identified prior to the xcodebuild exit, and Fortify Static Code Analyzer exits with an exit code of zero (unless some other error also occurs).</p> <p>Regardless of this setting, if xcodebuild exits with a non-zero code, then the xcodebuild exit code, stdout, and stderr are written to the log file.</p> <p>Value type: Boolean</p> <p>Default: false</p>
<code>com.fortify.sca. Apex</code>	<p>If set to true, Fortify Static Code Analyzer uses Apex translation for files with the <code>.cls</code> extension and Visualforce translation for files with the <code>.component</code> extension.</p>

Property Name	Description
	<p>Value type: Boolean</p> <p>Default: false</p> <p>Command-line Option: -apex</p>
com.fortify.sca. ApexObjectPath	<p>Specifies the absolute path of the custom sObject JSON file subjects.json.</p> <p>Value type: String</p> <p>Default: (none)</p> <p>Command-line Option: -apex-object-path</p>
com.fortify.sca. DefaultAnalyzers	<p>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.</p> <p>Value type: String</p> <p>Default: This property is commented out and all analysis types are used in scans.</p> <p>Command-line Option: -analyzers</p>
com.fortify.sca. EnableAnalyzer	<p>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.</p> <p>Value type: String</p> <p>Default: (none)</p>
com.fortify.sca. ExitCodeLevel	<p>Extends the default exit code options. See "Exit Codes" on page 113 for a description of the exit codes. The valid values are:</p> <p>The valid values are:</p> <ul style="list-style-type: none"> • 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. <p>Command-line Option: -exit-code-level</p>

Property Name	Description
<code>com.fortify.sca.IncrementalBaseScan</code>	<p>If set to true, requests a full scan of a project for which you can run subsequent incremental scans.</p> <p>Value type: Boolean</p> <p>Default: false</p>
<code>com.fortify.sca.IncrementalScan</code>	<p>If set to true, requests an incremental rescan of a previously scanned project.</p> <p>Value type: Boolean</p> <p>Default: false</p>
<code>com.fortify.sca.AddImpliedMethods</code>	<p>If set to true, Fortify Static Code Analyzer generates implied methods when it encounters implementation by inheritance.</p> <p>Value type: Boolean</p> <p>Default: true</p>
<code>com.fortify.sca.hoa.Enable</code>	<p>If set to true, higher-order analysis is enabled.</p> <p>Value type: Boolean</p> <p>Default: true</p>
<code>com.fortify.sca.Phase0HigherOrder.Languages</code>	<p>The languages for which to run phase zero higher-order analysis. Valid values are: python, swift, ruby, javascript, and typescript.</p> <p>Value type: String (comma separated list of languages)</p> <p>Default: python, ruby, swift</p>
<code>com.fortify.sca.Phase0HigherOrder.Limiter.FixpointPasses</code>	<p>Specifies the number of fixpoint subpasses to trade off the thoroughness of the higher-order analysis with the amount of time it takes.</p> <div data-bbox="711 1451 1401 1570" style="background-color: #f0f0f0; padding: 5px;"> <p>Note: Running scans with different numbers of fixpoint subpasses often produces slightly different results.</p> </div> <p>Value type: Number</p> <p>Default: 6</p>
<code>com.fortify.sca.Phase0HigherOrder.Timeout.Soft</code>	<p>Specifies the total time (in seconds) for higher-order analysis. When the analyzer reaches the soft timeout limit, it exits at the beginning of the next fixpoint subpass.</p> <div data-bbox="711 1835 1401 1877" style="background-color: #f0f0f0; padding: 5px;"> <p>Note: If the analyzer reaches the soft timeout limit</p> </div>

Property Name	Description
	<p>during the static inits or main pass, it is ignored.</p> <p>Value type: Number</p> <p>Default: 1800</p>
<p>com.fortify.sca. Phase0HigherOrder.Timeout. Hard</p>	<p>Specifies the total time (in seconds) for higher-order analysis. When the analyzer reaches the hard timeout limit, it exits immediately.</p> <p>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.</p> <p>Value type: Number</p> <p>Default: 2700</p>
<p>com.fortify.sca. TypeInferenceLanguages</p>	<p>Comma- or colon-separated list of languages that use type inference. This setting improves the precision of the analysis for dynamically-typed languages.</p> <p>Value type: String</p> <p>Default: javascript,python,ruby,typescript</p>
<p>com.fortify.sca. TypeInferencePhase0 Timeout</p>	<p>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.</p> <p>Value type: Long</p> <p>Default: 300</p>
<p>com.fortify.sca. TypeInferenceFunctionTimeout</p>	<p>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.</p> <p>Value type: Long</p> <p>Default: 60</p>
<p>com.fortify.sca. DisableFunctionPointers</p>	<p>If set to true, disables function pointers during the scan.</p> <p>Value type: Boolean</p> <p>Default: false</p>
<p>com.fortify.sca. RulesFileExtensions</p>	<p>Specifies a list of file extensions for rules files. Any files in <code><sca_install_dir>/Core/config/rules</code> (or a directory specified with the <code>-rules</code> option) whose</p>

Property Name	Description
	<p>extension is in this list is included. The <code>.bin</code> extension is always included, regardless of the value of this property. The delimiter for this property is the system path separator.</p> <p>Value type: String</p> <p>Default: <code>.xml</code></p>
<code>com.fortify.sca. RulesFile</code>	<p>Specifies a custom Rulepack or directory. If you specify a directory, all of the files in the directory with the <code>.bin</code> and <code>.xml</code> extensions are included.</p> <p>Value type: String (path)</p> <p>Default: (none)</p> <p>Command-line Option: <code>-rules</code></p>
<code>com.fortify.sca. NoDefaultRules</code>	<p>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.</p> <p>Value type: Boolean</p> <p>Default: (none)</p> <p>Command-line Option: <code>-no-default-rules</code></p>
<code>com.fortify.sca. NoDefaultIssueRules</code>	<p>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.</p> <p>Value type: Boolean</p> <p>Default: (none)</p> <p>Command-line Option: <code>-no-default-issue-rules</code></p>
<code>com.fortify.sca. NoDefaultSourceRules</code>	<p>If set to true, disables source rules in the default Rulepacks. This can be helpful when creating custom source rules.</p> <div data-bbox="711 1612 1399 1667" style="background-color: #f0f0f0; padding: 5px;"> <p>Note: Characterization source rules are not disabled.</p> </div> <p>Value type: Boolean</p> <p>Default: (none)</p> <p>Command-line Option: <code>-no-default-source-rules</code></p>

Property Name	Description
<code>com.fortify.sca.NoDefaultSinkRules</code>	<p>If set to true, disables sink rules in the default Rulepacks. This can be helpful when creating custom sink rules.</p> <p>Note: Characterization sink rules are not disabled.</p> <p>Value type: Boolean</p> <p>Default: (none)</p> <p>Command-line Option: <code>-no-default-sink-rules</code></p>
<code>com.fortify.sca.CustomRulesDir</code>	<p>Sets the directory used to search for custom rules.</p> <p>Value type: String (path)</p> <p>Default:</p> <p><code>\${com.fortify.Core}/config/customrules</code></p>
<code>com.fortify.sca.EnableFindbugs</code>	<p>If set to true, FindBugs is enabled as part of the scan.</p> <p>Value type: Boolean</p> <p>Default: true</p> <p>Command-line Option: <code>-findbugs</code></p>
<code>com.fortify.sca.findbugs.maxheap</code>	<p>Sets the maximum heap size for findbugs.</p> <p>Value type: String</p> <p>Default: Maximum heap size for Fortify Static Code Analyzer</p> <p>Example:</p> <p><code>com.fortify.sca.findbugs.maxheap=500m</code></p>
<code>com.fortify.sca.SuppressLowSeverity</code>	<p>If set to true, Fortify Static Code Analyzer ignores low severity issues found in a scan.</p> <p>Value type: Boolean</p> <p>Default: true</p>
<code>com.fortify.sca.LowSeverityCutoff</code>	<p>Specifies the cutoff level for severity suppression. Fortify Static Code Analyzer ignores any issues found with a lower severity value than the one specified for this property.</p> <p>Value type: Number</p> <p>Default: 1.0</p>
<code>com.fortify.sca.analyzer.controlflow.</code>	<p>Specifies whether to enable Control Flow Analyzer timeouts.</p>

Property Name	Description
EnableTimeout	<p>Value type: Boolean</p> <p>Default: true</p>
com.fortify.sca.RegExecutable	<p>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.</p> <p>Value type: String (path)</p> <p>Default: reg</p> <p>Example: com.fortify.sca.RegExecutable=C:\\Windows\\System32\\reg.exe</p>
com.fortify.sca.FilterFile	<p>Specifies the path to a filter file for the scan. See "Filter Files" on page 117 for more information.</p> <p>Value type: String (path)</p> <p>Default: (none)</p> <p>Command-line Option: -filter</p>
com.fortify.sca.FilteredInstanceIDs	<p>Specifies a comma-separated list of IIDs to be filtered out using a filter file.</p> <p>Value type: String</p> <p>Default: (none)</p>
com.fortify.sca.BinaryName	<p>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.</p> <p>Value type: String (path)</p> <p>Default: (none)</p> <p>Command-line Option: -bin or -binary-name</p>
com.fortify.sca.QuickScanMode	<p>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.</p> <p>Value type: Boolean</p> <p>Default: (not enabled)</p> <p>Command-line Option: -quick</p>

Property Name	Description
<code>com.fortify.sca.ProjectTemplate</code>	<p>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.</p> <p>Value type: String</p> <p>Default: (none)</p> <p>Command-line Option: <code>-project-template</code></p> <p>Example: <code>com.fortify.sca.ProjectTemplate=test_issuetemplate.xml</code></p>
<code>com.fortify.sca.alias.Enable</code>	<p>If set to true, enables alias analysis.</p> <p>Value type: Boolean</p> <p>Default: true</p>
<code>com.fortify.sca.UniversalBlacklist</code>	<p>Specifies a list of functions to blacklist from all analyzers.</p> <p>Value type: String (colon-separated list)</p> <p>Default: <code>.*yyparse.*</code></p>
<code>com.fortify.sca.MultithreadedAnalysis</code>	<p>Specifies whether or not Fortify Static Code Analyzer runs in parallel analysis mode.</p> <p>Value type: Boolean</p> <p>Default: true</p>
<code>com.fortify.sca.ThreadCount</code>	<p>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.</p> <p>Value type: Integer</p> <p>Default: (number of available processor cores)</p>
<code>com.fortify.sca.DISabledLanguages</code>	<p>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, typescript, vb</p> <p>Value type: String</p>

Property Name	Description
	<p>Default: (none)</p>
<p>com.fortify.sca. JdkVersion</p>	<p>Specifies the Java source code version to the Java translator.</p> <p>Value type: String</p> <p>Default: 1.8</p> <p>Command-line Option: -jdk</p>
<p>com.fortify.sca. JavaClasspath</p>	<p>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).</p> <p>Value type: String (paths)</p> <p>Default: (none)</p> <p>Command-line Option: -cp or -classpath</p>
<p>com.fortify.sca. Appserver</p>	<p>Specifies the application server to process JSP files. The valid values are weblogic or websphere.</p> <p>Value type: String</p> <p>Default: (none)</p> <p>Command-line Option: -appserver</p>
<p>com.fortify.sca. AppserverHome</p>	<p>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.</p> <p>Value type: String (path)</p> <p>Default: (none)</p> <p>Command-line Option: -appserver-home</p>
<p>com.fortify.sca. AppserverVersion</p>	<p>Specifies the version of the application server.</p> <p>Value type: String</p> <p>Default: (none)</p> <p>Command-line Option: -appserver-version</p>
<p>com.fortify.sca. JavaExtdirs</p>	<p>Specifies directories to include implicitly on the class path for WebLogic and WebSphere application servers.</p> <p>Value type: String</p> <p>Default: (none)</p>

Property Name	Description
	Command-line Option: -extdirs
com.fortify.sca. JavaSourcepath	<p>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.</p> <p>Value type: String (paths)</p> <p>Default: (none)</p> <p>Command-line Option: -sourcepath</p>
com.fortify.sca. JavaSourcepathSearch	<p>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.</p> <p>Value type: Boolean</p> <p>Default: true</p>
com.fortify.sca. DefaultJarsDirs	<p>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).</p> <p>Value type: String</p> <p>Default: (none)</p>
com.fortify.sca jsp.UseSecurityManager	<p>If set to true, the JSP parser uses JSP security manager.</p> <p>Value type: Boolean</p> <p>Default: true</p>
com.fortify.sca. jsp.DefaultEncoding	<p>Specifies the encoding for JSPs.</p> <p>Value type: String (encoding)</p> <p>Default: ISO-8859-1</p>
com.fortify.sca DotnetVersion	<p>Specifies the .NET framework version. See the <i>Micro Focus Fortify Software System Requirements</i> for a list of supported versions.</p> <p>Value type: String</p> <p>Default: (the latest supported version)</p> <p>Command-line Option: -dotnet-version</p>
com.fortify.sca	Specifies the .NET Core version.

Property Name	Description
DotnetCoreVersion	<p>Value type: String</p> <p>Default: (none)</p> <p>Command-line Option: -dotnet-core-version</p>
com.fortify.sca. DotnetStdVersion	<p>Specifies the .NET Standard version.</p> <p>Value type: String</p> <p>Default: (none)</p> <p>Command-line Option: -dotnet-std-version</p>
com.fortify.sca. DotnetLibdirs	<p>Specifies the semicolon-delimited list of directories where third-party DLL files are located. Used for default .NET library files.</p> <p>Value type: String (path)</p> <p>Default: (none)</p> <p>Command-line Option: -libdirs</p>
com.fortify.sca. DotnetLibdirsOnly	<p>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.</p> <p>Value type: Boolean</p> <p>Default: false</p> <p>Command-line Option: -libdirs-only</p>
com.fortify.sca. NugetCacheDir	<p>Overrides the default path to the NuGet cache directory.</p> <p>Value type: String (path)</p> <p>Default: The .nuget/packages folder in the current user's home directory (Windows environment variable: USERPROFILE)</p> <p>Command-line Option: -nuget-cache-dir</p>
com.fortify.sca. DotnetSharedFiles	<p>Specifies a semicolon-separated list of the source files for all Shared Projects included in the project you are translating.</p> <p>Value type: String (files)</p> <p>Default: (none)</p> <p>Command-line Option: -dotnet-shared-files</p>

Property Name	Description
com.fortify.sca.DotnetOutputDir	<p>Specifies the output directory where the binary (EXE or DLL) built from the project is placed.</p> <p>Value type: String (path)</p> <p>Default: (none)</p> <p>Command-line Option: -dotnet-output-dir</p>
com.fortify.sca.DotnetWebRoot	<p>Specifies the root (home) directory of an ASP.NET project.</p> <p>Value type: String (path)</p> <p>Default: (none)</p> <p>Command-line Option: -dotnetwebroot</p>
com.fortify.sca.WebSiteProject	<p>If set to true, the project is of type WebSite.</p> <p>Value type: Boolean</p> <p>Default: false</p> <p>Command-line Option: -dotnet-website</p>
com.fortify.sca.DotnetWebAppLibs	<p>Specifies a semicolon-separated list of additional reference DLLs needed to translate ASP.NET pages.</p> <p>Value type: String (path)</p> <p>Default: (none)</p> <p>Command-line Option: -dotnet-applibs</p>
com.fortify.sca.DotnetCodeBehind	<p>Specifies a semicolon-separated list of source files that are bound to ASP.NET pages (referred to as <i>code-behind files</i>).</p> <p>Value type: String (path)</p> <p>Default: (none)</p> <p>Command-line Option: -dotnet-codebehind</p>
com.fortify.sca.AspNetCore	<p>If set to true, indicates a web project (ASP.NET or ASP.NET Core) that targets the .NET Core or .NET Standard framework.</p> <p>Value type: Boolean</p> <p>Default: false</p> <p>Command-line Option: -aspnetcore</p>
com.fortify.sca.DotnetAssemblyName	<p>Specifies the name of the target .NET assembly as specified in Visual Studio project settings.</p>

Property Name	Description
	<p>Value type: String</p> <p>Default: (none)</p> <p>Command-line Option: -dotnet-assembly-name</p>
<p>com.fortify.sca. DotnetAlias</p>	<p>Specifies a list of external aliases for a specified DLL file in the following format: <i><alias1></i>,<i><alias2></i>,..<i>=<path_to_dll></i>. You can include multiple aliases as a semicolon-separated list of pairs.</p> <p>Value type: String</p> <p>Default: (none)</p> <p>Command-line Option: -cs-extern-alias</p>
<p>com.fortify.sca. DotnetPreprocessorSymbols</p>	<p>Specifies a semicolon-separated list of preprocessor symbols used in the source code.</p> <p>Value type: String</p> <p>Default: (none)</p> <p>Command-line Option: -dotnet-preproc-symbols</p>
<p>com.fortify.sca. VBCompileOptions</p>	<p>Specifies any special compilation options required for the correct translation of the source code, such as OptionStrict, OptionInfer, and OptionExplicit.</p> <p>Value type: String</p> <p>Default: (none)</p> <p>Command-line Option: -vb-compile-options</p>
<p>com.fortify.sca. VBGlobalImports</p>	<p>Specifies a semicolon-separated list of namespaces imported for all source files in the project.</p> <p>Value type: String</p> <p>Default: (none)</p> <p>Command-line Option: -vb-imports</p>
<p>com.fortify.sca. VBMyType</p>	<p>Specifies the value for the <i>_MYTYPE</i> preprocessor symbol that is specified in the <i><MyType></i> tag in the project settings. This is required if the source code to be translated uses My namespace.</p> <p>Value type: String</p> <p>Default: (none)</p> <p>Command-line Option: -vb-mytype</p>

Property Name	Description
com.fortify.sca.VBRootNamespace	<p>Specifies the root namespace for the project as specified in Visual Studio project settings.</p> <p>Value type: String</p> <p>Default: (none)</p> <p>Command-line Option: -vb-root</p>
com.fortify.sca.XamarinAndroidVersion	<p>Specifies the target Android SDK version for Xamarin Android projects.</p> <p>Value type: String</p> <p>Default: (the latest installed version)</p> <p>Command-line Option: -xamarin-android-version</p>
com.fortify.sca.XamariniOSVersion	<p>Specifies the target iOS SDK version for Xamarin iOS projects.</p> <p>Value type: String</p> <p>Default: (the latest installed version)</p> <p>Command-line Option: -xamarin-ios-version</p>
<p>WinForms.TransformDataBindings</p> <p>WinForms.TransformMessageLoops</p> <p>WinForms.TransformChangeNotificationPattern</p> <p>WinForms.CollectionMutationMonitor.Label</p> <p>WinForms.ExtractEventHandlers</p>	<p>Set various .NET options.</p> <p>Value type: Boolean and String</p> <p>Defaults and Examples:</p> <p>WinForms.TransformDataBindings=true</p> <p>WinForms.TransformMessageLoops=true</p> <p>WinForms.TransformChangeNotificationPattern = true</p> <p>WinForms.CollectionMutationMonitor.Label=WinFormsDataSource</p> <p>WinForms.ExtractEventHandlers=true</p>
com.fortify.sca.EnabledDOMModeling	<p>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 HTML files that have embedded or referenced JavaScript code.</p>

Property Name	Description
	<p>Note: Enabling this property can increase the translation time.</p> <p>Value type: Boolean</p> <p>Default: false</p>
<p>com.fortify.sca DOMModeling.tags</p>	<p>If you set the <code>com.fortify.sca.EnableDOMModeling</code> property to true, you can specify additional HTML tags for Fortify Static Code Analyzer to include in the DOM modeling.</p> <p>Value type: String (comma-separated HTML tag names)</p> <p>Default: body, button, div, form, iframe, input, head, html, and p.</p> <p>Example: com.fortify.sca.DOMModeling.tags=ul,li</p>
<p>com.fortify.sca. JavaScript.src.domain. whitelist</p>	<p>Specifies trusted domain names where Fortify Static Code Analyzer can download referenced JavaScript files for the scan. Delimit the URLs with vertical bars.</p> <p>Value type: String</p> <p>Default: (none)</p> <p>Example: com.fortify.sca.JavaScript. src.domain.whitelist= http://www.xyz.com http://www.123.org</p>
<p>com.fortify.sca. DisableJavascript Extraction</p>	<p>If set to true, JavaScript code embedded in JSP, JSPX, PHP, and HTML files is not extracted and not scanned.</p> <p>Value type: Boolean</p> <p>Default: false</p>
<p>com.fortify.sca. skip.libraries.AngularJS com.fortify.sca. skip.libraries.ES6 com.fortify.sca. skip.libraries.jQuery com.fortify.sca. skip.libraries.javascript com.fortify.sca. skip.libraries.typescript</p>	<p>Specifies a list of comma- or colon-separated JavaScript technology library files that are not translated. You can use regular expressions in the file names. Note that the regular expression <code>'(-\d\.\d\.\d)?'</code> is automatically inserted before <code>.min.js</code> or <code>.js</code> for each file name included in the <code>com.fortify.sca.skip.libraries.jQuery</code> property value.</p> <p>Value type: String</p> <p>Defaults:</p> <ul style="list-style-type: none"> AngularJS: angular.js, angular.min.js,

Property Name	Description
	<p>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</p> <ul style="list-style-type: none"> • ES6: es6-shim.min.js, system-polyfills.js, shims_for_IE.js • jQuery: jquery.js, jquery.min.js, jquery-migrate.js, jquery-migrate.min.js, jquery-ui.js, jquery-ui.min.js, jquery.mobile.js, jquery.mobile.min.js, jquery.color.js, jquery.color.min.js, jquery.color.svg-names.js, jquery.color.svg-names.min.js, jquery.color.plus-names.js, jquery.color.plus-names.min.js, jquery.tools.min.js • javascript: bootstrap.js, bootstrap.min.js, typescript.js, typescriptServices.js • typescript: typescript.d.ts, typescriptServices.d.ts
<p>com.fortify.sca. PHPVersion</p>	<p>Specifies the PHP version. For a list of valid versions, see the <i>Micro Focus Fortify Software System Requirements</i>.</p> <p>Value type: String</p> <p>Default: 7.0</p> <p>Command-Line Option: -php-version</p>
<p>com.fortify.sca. PHPSourceRoot</p>	<p>Specifies the PHP source root.</p> <p>Value type: Boolean</p> <p>Default: (none)</p> <p>Command-Line Option: -php-source-root</p>
<p>com.fortify.sca. PythonPath</p>	<p>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.</p>

Property Name	Description
	<p>Value type: String (path)</p> <p>Default: (none)</p> <p>Command-line Option: -python-path</p>
com.fortify.sca. PythonVersion	<p>Specifies the Python source code version you want to scan. The valid values are 2 and 3.</p> <p>Value type: Number</p> <p>Default: 2</p> <p>Command-Line Option: -python-version</p>
com.fortify.sca. DjangoTemplateDirs	<p>Specifies path to Django templates. Fortify Static Code Analyzer does not use the TEMPLATE_DIRS setting from the Django settings.py file.</p> <p>Value type: String (paths)</p> <p>Default: (none)</p> <p>Command-line Option: -django-template-dirs</p>
com.fortify.sca. RubyLibraryPaths	<p>Specifies one or more paths to directories that contain Ruby libraries.</p> <p>Value type: String (path)</p> <p>Default: (none)</p> <p>Command-line Option: -ruby-path</p>
com.fortify.sca. RubyGemPaths	<p>Specifies the path(s) to a RubyGems location. Set this value if the project has associated gems to scan.</p> <p>Value type: String (path)</p> <p>Default: (none)</p> <p>Command-line Option: -rubygem-path</p>
com.fortify.sca. FlexLibraries	<p>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.</p> <p>Value type: String (path)</p> <p>Default: (none)</p> <p>Command-line Option: -flex-libraries</p>

Property Name	Description
<p>com.fortify.sca. FlexSdkRoot</p>	<p>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 mxm1c executable.</p> <p>Value type: String (path)</p> <p>Default: (none)</p> <p>Command-line Option: -flex-sdk-root</p>
<p>com.fortify.sca. FlexSourceRoots</p>	<p>Specifies any additional source directories for a Flex project. Separate the list of directories with semicolons (Windows) or colons (non-Windows systems).</p> <p>Value type: String (path)</p> <p>Default: (none)</p> <p>Command-line Option: -flex-source-root</p>
<p>com.fortify.sca. AbapDebug</p>	<p>If set to true, Fortify Static Code Analyzer adds ABAP statements to debug messages.</p> <p>Value type: String (statement)</p> <p>Default: (none)</p>
<p>com.fortify.sca. AbapIncludes</p>	<p>When Fortify Static Code Analyzer encounters an ABAP 'INCLUDE' directive, it looks in the named directory.</p> <p>Value type: String (path)</p> <p>Default: (none)</p>
<p>com.fortify.sca. CobolFixedFormat</p>	<p>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.</p> <p>Value type: Boolean</p> <p>Default: false</p> <p>Command-line Option: -fixed-format</p>
<p>com.fortify.sca. SqlLanguage</p>	<p>Sets the SQL language variant. The valid values are PLSQL (for Oracle PL/SQL) and TSQL (for Microsoft T-SQL).</p> <p>Value type: String (SQL language type)</p> <p>Default: TSQL</p> <p>Command-line Option: -sql-language</p>
<p>com.fortify.sca.</p>	<p>If set to true, Fortify Static Code Analyzer treats</p>

Property Name	Description
CfmlUndefinedVariablesAreTainted	<p>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.</p> <p>Value type: Boolean</p> <p>Default: false</p>
com.fortify.sca. CaseInsensitiveFiles	<p>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.</p> <p>Value type: Boolean</p> <p>Default: (not enabled)</p>
com.fortify.sca. SourceBaseDir	<p>Specifies the base directory for ColdFusion projects.</p> <p>Value type: String (path)</p> <p>Default: (none)</p> <p>Command-Line Option: -source-base-dir</p>
com.fortify.sca. FVDLDisableDescriptions	<p>If set to true, excludes descriptions from the analysis results file (FVDL).</p> <p>Value type: Boolean</p> <p>Default: false</p> <p>Command-line Option: -fvd1-no-description</p>
com.fortify.sca. FVDLDisableProgramData	<p>If set to true, excludes the ProgramData section from the analysis results file (FVDL).</p> <p>Value type: Boolean</p> <p>Default: false</p> <p>Command-line Option: -fvd1-no-progdata</p>
com.fortify.sca. FVDLDisableEngineData	<p>If set to true, excludes the engine data from the analysis results file (FVDL).</p> <p>Value type: Boolean</p> <p>Default: false</p> <p>Command-line Option: -fvd1-no-enginedata</p>
com.fortify.sca.	<p>If set to true, excludes code snippets from the analysis</p>

Property Name	Description
FVDLDisableSnippets	<p>results file (FVDL).</p> <p>Value type: Boolean</p> <p>Default: false</p> <p>Command-line Option: -fvd1-no-snippets</p>
com.fortify.sca.FVDLDisableLabelEvidence	<p>If set to true, excludes the label evidence from the analysis results file (FVDL).</p> <p>Value type: Boolean</p> <p>Default: false</p>
com.fortify.sca.FVDLStylesheet	<p>Specifies location of the style sheet for the analysis results.</p> <p>Value type: String (path)</p> <p>Default: \${com.fortify.Core}/resources/sca/fvd12html.xsl</p>
com.fortify.sca.ResultsFile	<p>The file to which results are written.</p> <p>Value type: String</p> <p>Default: (none)</p> <p>Command-line Option: -f</p> <p>Example: com.fortify.sca.ResultsFile=results.fpr</p>
com.fortify.sca.OutputAppend	<p>If set to true, Fortify Static Code Analyzer appends results to an existing results file.</p> <p>Value type: Boolean</p> <p>Default: false</p> <p>Command-line Option: -append</p>
com.fortify.sca.Renderer	<p>Controls the output format. The valid values are fpr, fvd1, text, and auto. The default of auto selects the output format based on the file extension of the file provided with the -f option.</p> <p>Value type: String</p> <p>Default: auto</p> <p>Command-line Option: -format</p>
com.fortify.sca.	<p>If set to true, Fortify Static Code Analyzer prints results as</p>

Property Name	Description
ResultsAsAvailable	<p>they become available. This is helpful if you do not specify the -f option (to specify an output file) and print to stdout.</p> <p>Value type: Boolean</p> <p>Default: false</p>
com.fortify.sca. BuildProject	<p>Specifies a name for the scanned project. Fortify Static Code Analyzer does not use this name but includes it in the results.</p> <p>Value type: String</p> <p>Default: (none)</p> <p>Command-line Option: -build-project</p>
com.fortify.sca. BuildLabel	<p>Specifies a label for the scanned project. Fortify Static Code Analyzer does not use this label but includes it in the results.</p> <p>Value type: String</p> <p>Default: (none)</p> <p>Command-line Option: -build-label</p>
com.fortify.sca. BuildVersion	<p>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.</p> <p>Value type: String</p> <p>Default: (none)</p> <p>Command-line Option: -build-version</p>
com.fortify.sca. MachineOutputMode	<p>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.</p> <p>Value type: Boolean</p> <p>Default: (not enabled)</p>
com.fortify.sca. SnippetContextLines	<p>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.</p> <p>Value type: Number</p>

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. ClobberLogFile	If set to true, Fortify Static Code Analyzer overwrites the log file for each run of sourceanalyzer. Value type: Boolean Default: false Command-line Option: -clobber-log
com.fortify.sca. LogFile	Specifies the default log file name and location. Value type: String (path) Default: \${com.fortify.sca.ProjectRoot}/log/sca.log and \${com.fortify.sca.ProjectRoot}/log/sca_FortifySupport.log Command-line Option: -logfile
com.fortify.sca. LogLevel	Specifies the minimum log level for both log files. The valid values are: DEBUG, INFO, WARN, ERROR, and FATAL. For more information, see "Accessing Log Files" on page 115 and "Configuring Log Files" on page 115 . Value type: String Default: INFO
com.fortify.sca. PrintPerformanceDataAfterScan	If set to true, Fortify Static Code Analyzer writes performance-related data to the Fortify Support log file after the scan is complete. This value is automatically set to true when in debug mode.

Property Name	Description
	<p>Value type: Boolean</p> <p>Default: false</p>
<p>com.fortify.sca. Debug</p>	<p>Includes debug information in the Fortify Support log file, which is only useful for Micro Focus Fortify Customer Support to help troubleshoot.</p> <p>Value type: Boolean</p> <p>Default: false</p> <p>Command-line Option: -debug</p>
<p>com.fortify.sca. DebugVerbose</p>	<p>This is the same as the com.fortify.sca.Debug property, but it includes more details, specifically for parse errors.</p> <p>Value type: Boolean</p> <p>Default: (not enabled)</p> <p>Command-line Option: -debug-verbose</p>
<p>com.fortify.sca. Verbose</p>	<p>If set to true, includes verbose messages in the Fortify Support log file.</p> <p>Value type: Boolean</p> <p>Default: (not enabled)</p> <p>Command-line Option: -verbose</p>
<p>com.fortify.sca. DebugTrackMem</p>	<p>If set to true, enables additional debugging for performance information to be written to the Fortify Support log.</p> <p>Value type: Boolean</p> <p>Default: (not enabled)</p> <p>Command-line Option: -debug-mem</p>
<p>com.fortify.sca. CollectPerformanceData</p>	<p>If set to true, enables additional timers to track performance.</p> <p>Value type: Boolean</p> <p>Default: (not enabled)</p>
<p>com.fortify.sca. Quiet</p>	<p>If set to true, disables the command-line progress information.</p> <p>Value type: Boolean</p> <p>Default: false</p>

Property Name	Description
	Command-line Option: -quiet
com.fortify.sca.MonitorSca	<p>If set to true, Fortify Static Code Analyzer monitors its memory use and warns when JVM garbage collection becomes excessive.</p> <p>Value type: Boolean</p> <p>Default: true</p>
com.fortify.sca.cpfe.command	<p>Sets the location of the CPFE binary to use in the translation phase.</p> <p>Value type: String (path)</p> <p>Default: \${com.fortify.Core}/private-bin/sca/cpfe48</p>
com.fortify.sca.cpfe.441	<p>If set to true, Fortify Static Code Analyzer uses CPFE version 4.4.1.</p> <p>Value type: Boolean</p> <p>Default: false</p>
com.fortify.sca.cpfe.441.command	<p>Sets the location of the CPFE binary (version 4.4.1) to use in the translation phase.</p> <p>Value type: String (path)</p> <p>Default: \${com.fortify.Core}/private-bin/sca/cpfe441.rfct</p>
com.fortify.sca.cpfe.options	<p>Adds options to the CPFE command line to use when translating C/C++ code.</p> <p>Value type: String</p> <p>Default: --remove_unneeded_entities --suppress_vtbl -tused</p>
com.fortify.sca.cpfe.file.option	<p>Sets the name of CPFE option that specifies the output (for example NST) file name.</p> <p>Value type: String</p> <p>Default: --gen_c_file_name</p> <p>Example: com.fortify.sca.cpfe.file.option= --gen_c_file_name</p>

Property Name	Description
<code>com.fortify.sca.cpfe.multibyte</code>	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
<code>com.fortify.sca.cpfe.CaptureWarnings</code>	If set to true, any CPFE warnings are included in the Fortify Static Code Analyzer log. Value type: Boolean Default: false
<code>com.fortify.sca.cpfe.FailOnError</code>	If set to true, CPFE fails if there is an error. Value type: Boolean Default: false
<code>com.fortify.sca.cpfe.IgnoreFileOpen Failures</code>	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
<code>com.fortify.sca.ASPVirtualRoots.<virtual_path></code>	Specifies a semicolon delimited list of full paths to virtual roots used. Value type: String Default: (none) Example: <code>com.fortify.sca.ASPVirtualRoots.Library=c:\\WebServer\\CustomerTwo\\Stuff</code> <code>com.fortify.sca.ASPVirtualRoots.Include=c:\\WebServer\\CustomerOne\\inc</code>
<code>com.fortify.sca.DisableASPEXternalEntries</code>	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
<code>com.fortify.sca.CtrlflowMaxFunctionTime</code>	<p>Sets the time limit (in milliseconds) for Control Flow analysis on a single function.</p> <p>Value type: Integer</p> <p>Quick scan default: 30000</p> <p>Default: 600000</p>
<code>com.fortify.sca.DisableAnalyzers</code>	<p>Specifies a comma- or colon-separated list of analyzers to disable during a scan. The valid values for this property are: <code>buffer</code>, <code>content</code>, <code>configuration</code>, <code>controlflow</code>, <code>dataflow</code>, <code>findbugs</code>, <code>nullptr</code>, <code>semantic</code>, and <code>structural</code>.</p> <p>Value type: String</p> <p>Quick scan default: <code>controlflow:buffer</code></p> <p>Default: (none)</p>
<code>com.fortify.sca.FilterSet</code>	<p>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 <code>com.fortify.sca.ProjectTemplate</code> described in "fortify-sca.properties" on page 128 to specify an issue template that contains the filter set to use.</p> <p>When set to <code>Quick View</code>, 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 <i>Micro Focus Fortify Audit Workbench User Guide</i>.</p> <p>Value type: String</p> <p>Quick scan default: <code>Quick View</code></p> <p>Default: (none)</p>

Property Name	Description
<code>com.fortify.sca.FPRDisableMetatable</code>	<p>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.</p> <p>Value type: Boolean</p> <p>Quick scan default: true</p> <p>Default: false</p>
<code>com.fortify.sca.FPRDisableSourceBundling</code>	<p>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.</p> <p>Value type: Boolean</p> <p>Quick scan default: true</p> <p>Default: false</p>
<code>com.fortify.sca.NullPtrMaxFunctionTime</code>	<p>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.</p> <p>Value type: Integer</p> <p>Quick scan default: 10000</p> <p>Default: 300000</p>
<code>com.fortify.sca.TrackPaths</code>	<p>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.</p> <p>Value type: String</p> <p>Quick scan default: (none)</p> <p>Default: NoJSP</p>
<code>com.fortify.sca.limiters.ConstraintPredicateSize</code>	<p>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.</p>

Property Name	Description
	<p>Value type: Integer</p> <p>Quick scan default: 10000</p> <p>Default: 500000</p>
<code>com.fortify.sca. limiters.MaxChainDepth</code>	<p>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.</p> <p>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 <code>main()</code>.</p> <p>Value type: Integer</p> <p>Quick scan default: 3</p> <p>Default: 5</p>
<code>com.fortify.sca. limiters.MaxFunctionVisits</code>	<p>Sets the number of times taint propagation analyzer visits functions.</p> <p>Value type: Integer</p> <p>Quick scan default: 5</p> <p>Default: 50</p>
<code>com.fortify.sca. limiters.MaxPaths</code>	<p>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.</p> <p>Note: Fortify does not recommend setting this property to a value larger than 5 because it might increase the scan time.</p> <p>Value type: Integer</p> <p>Quick scan default: 1</p> <p>Default: 5</p>
<code>com.fortify.sca. limiters.MaxTaintDefForVar</code>	<p>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.</p> <p>Value type: Integer</p>

Property Name	Description
	Quick scan default: 250 Default: 1000
<code>com.fortify.sca. limiters.MaxTaintDefForVarAbort</code>	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 18.20)

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!