



Hewlett Packard
Enterprise

HPE Security

Fortify Static Code Analyzer

Software Version: 16.20

User Guide

Document Release Date: December 2016
Software Release Date: December 2016

Legal Notices

Warranty

The only warranties for Hewlett Packard Enterprise Development products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HPE 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. Valid license from HPE 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.

The software is restricted to use solely for the purpose of scanning software for security vulnerabilities that is (i) owned by you; (ii) for which you have a valid license to use; or (iii) with the explicit consent of the owner of the software to be scanned, and may not be used for any other purpose.

You shall not install or use the software on any third party or shared (hosted) server without explicit consent from the third party.

Copyright Notice

© Copyright 2003 - 2016 Hewlett Packard Enterprise Development LP

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.protect724.hpe.com/community/fortify/fortify-product-documentation>

You will receive updated or new editions if you subscribe to the appropriate product support service. Contact your HPE sales representative for details.

Contents

Preface	8
Contacting HPE Security Fortify Support	8
For More Information	8
About the Documentation Set	8
Change Log	9
Chapter 1: Introduction	11
HPE Security Fortify Static Code Analyzer	11
HPE Security Fortify CloudScan	11
HPE Security Fortify Scan Wizard	12
About the Analyzers	12
Related Documents	13
All Products	14
HPE Security Fortify Software Security Center	14
HPE Security Fortify Static Code Analyzer	16
Technology Previews	17
Chapter 2: Analysis Process Overview	18
Analysis Process	18
Translation Phase	19
Mobile Build Sessions	20
Mobile Build Session Version Compatibility	20
Creating a Mobile Build Session	20
Importing a Mobile Build Session	20
Analysis Phase	21
Incremental Analysis	21
Parallel Processing	22
Translation and Analysis Phase Verification	22
Chapter 3: Translating Java Code	23
Java Command-Line Syntax	23
Java Command-Line Options	24
Java Command-Line Examples	25
Handling Resolution Warnings	26
Java Warnings	26

Using FindBugs	26
Translating Java EE Applications	27
Translating the Java Files	28
Translating JSP Projects, Configuration Files, and Deployment Descriptors	28
Java EE Translation Warnings	28
Translating Java Bytecode	28
Chapter 4: Translating .NET Code	30
.NET Command-Line Syntax	30
.NET Command-Line Options	31
Translating Simple .NET Applications	32
Handling Translation Errors	33
.NET Translation Errors	33
ASP.NET Errors	33
Chapter 5: Translating C and C++ Code	34
Prerequisites	34
C and C++ Command-Line Syntax	34
Scanning Pre-processed C and C++ Code	35
Chapter 6: Translating JavaScript Code	36
Translating Pure JavaScript Projects	36
Translating JavaScript Projects with HTML Files	36
Including External JavaScript or HTML in the Translation	37
Scanning JavaScript Code	38
Chapter 7: Translating Ruby Code	39
Ruby Command-Line Syntax	39
Ruby Command-Line Options	39
Adding Libraries	40
Adding Gem Paths	40
Chapter 8: Translating ABAP Code	41
About Scanning ABAP Code	41
INCLUDE Processing	42
Importing the Transport Request	42
Adding Fortify Static Code Analyzer to Your Favorites List	43
Running the HPE Security Fortify ABAP Extractor	44

Chapter 9: Translating Code for Mobile Platforms	47
Translating Apple iOS Projects	47
Prerequisites	47
Xcodebuild Integration Command-Line Syntax	47
Translating Android Projects	48
Chapter 10: Translating Flex and ActionScript	49
ActionScript Command-Line Syntax	49
Flex and ActionScript Command-Line Options	49
ActionScript Command-Line Examples	50
Handling Resolution Warnings	51
ActionScript Warnings	51
Chapter 11: Translating COBOL Code	52
Preparing COBOL Source Files for Translation	52
COBOL Command-Line Syntax	53
Chapter 12: Translating Other Languages	55
Translating Python Code	55
Python Command-Line Options	56
Translating ColdFusion Code	56
ColdFusion Command-Line Syntax	56
ColdFusion Command-Line Options	57
Translating SQL	57
PL/SQL Command-Line Example	57
T-SQL Command-Line Example	58
Translating ASP/VBScript Virtual Roots	58
Classic ASP Command-Line Example	60
VBScript Command-Line Example	60
PHP Command-Line Example	60
Chapter 13: Integrating into a Build	61
Build Integration	61
Make Example	62
Devenv Example	62
Modifying a Build Script to Invoke Fortify Static Code Analyzer	62
Touchless Build Integration	63
Ant Integration	63
Gradle Integration	64
Maven Integration	64

Installing and Updating the Maven Plugin	64
Testing the Maven Plugin Installation	65
Using the Maven Plugin	66
Excluding Files from the Scan	67
MSBuild Integration	67
Setting Windows Environment Variables for Touchless MSBuild Integration	68
Chapter 14: Command-Line Interface	70
Output Options	70
Translation Options	72
Analysis Options	73
Other Options	75
Directives	76
Specifying Files	77
Chapter 15: Command-Line Utilities	79
Fortify Static Code Analyzer Utilities	79
Other Command-Line Utilities	80
Checking the Fortify Static Code Analyzer Scan Status	80
SCASState Utility Command-Line Options	81
Working with FPR Files from the Command Line	82
Merging FPR Files	83
Displaying Analysis Results for an FPR File	84
Migrating Audit Data from Previous FPR Versions	86
Extracting a Source Archive from an FPR File	87
Generating Reports from the Command Line	88
Generating a BIRT Report	88
Generating a Legacy Report	90
About Updating Security Content	90
Updating Security Content	91
Chapter 16: Troubleshooting	93
Exit Codes	93
Using the Log File to Debug Problems	94
Translation Failed Message	94
JSP Translation Problems	95
C/C++ Precompiled Header Files	95
Reporting Issues and Requesting Enhancements	96
Appendix A: Parallel Analysis Mode	97

Configuring Parallel Analysis Mode	97
Running in Parallel Analysis Mode	98
Strategies for Best Use	98
Appendix B: Filtering the Analysis	100
Filter Files	100
Filter File Example	100
Appendix C: Scan Wizard	103
Preparing to use the Scan Wizard	103
Starting the Scan Wizard	104
Starting Scan Wizard on a System with Fortify SCA and Applications Installed	104
Starting Scan Wizard as a Stand-Alone Utility	105
Appendix D: Sample Files	106
Basic Samples	106
Advanced Samples	107
Appendix E: Issue Tuning	110
Wrapper Detection	110
Interprocedural Constant Propagation	111
Selective Map Operation Tracking	111
Appendix F: Configuration Options	113
Fortify Static Code Analyzer Properties Files	113
Properties File Format	113
Precedence of Setting Properties	114
fortify-sca.properties	114
fortify-sca-quickscan.properties	135
Send Documentation Feedback	140

Preface

Contacting HPE Security Fortify Support

If you have questions or comments about using this product, contact HPE Security Fortify Technical Support using one of the following options.

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

<https://support.fortify.com>

To Email Support

fortifytechsupport@hpe.com

To Call Support

1.844.260.7219

For More Information

For more information about HPE Security software products: <http://www.hpe.com/software/fortify>

About the Documentation Set

The HPE Security Fortify Software documentation set contains installation, user, and deployment guides for all HPE Security 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 HPE Security user community website:

<https://www.protect724.hpe.com/community/fortify/fortify-product-documentation>

You will need to register for an account.

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
16.20	<p>Added:</p> <ul style="list-style-type: none">• "Incremental Analysis" on page 21 - New feature• "Gradle Integration" on page 64 - New feature• "Exit Codes" on page 93 <p>Updated:</p> <ul style="list-style-type: none">• "Translating .NET Code" on page 30 - New implementation for .NET translation• "Integrating into a Build" on page 61 - Combined all the build integration information into one chapter• "Including External JavaScript or HTML in the Translation" on page 37 - Added how to specify proxy server information• "Translation Options" on page 72 - Added more information about the -encoding option• "fortify-sca.properties" on page 114 - Removed <code>com.fortify.sca.jsp.UseNativeParser</code> property because the Jasper-based non-native JSP parser was removed, added equivalent command-line options for relevant properties• Command-line option descriptions now include equivalent property names where applicable <p>Removed:</p> <ul style="list-style-type: none">• "Prerequisite for Translating Code Using Legacy Versions of the J2EE SDK" (no longer supported)• "Using the Sourceanalyzer Ant Task" (no longer supported)• "Precompiling MS Visual Studio 2003 ASP.NET Pages" (no longer supported)• "Translating ASP.NET 1.1 (Visual Studio Version 2003)" Projects (no longer supported)• The <code>com.fortify.sca.IldasmPath</code> property (no longer used with .NET translation)
16.10	<p>Added:</p> <ul style="list-style-type: none">• "Translating JavaScript Code" on page 36 <p>Updated:</p>

Software Release / Document Version	Changes
	<ul style="list-style-type: none">• "Translating Code for Mobile Platforms" on page 47 - Added Swift language as supported• "Updating Security Content" on page 91 - Added two new options to the fortifyupdate utility• "Generating a BIRT Report " on page 88 - Additional output format (XLS)• "Updating Security Content" on page 91 - Two new fortifyupdate utility options• "Maven Integration" on page 64• "Advanced Samples" on page 107 - Added two riches samples (Java and .NET)
4.40	<p>Added:</p> <ul style="list-style-type: none">• "Fortify Static Code Analyzer Utilities" on page 79 (was a separate user guide)• "Scan Wizard" on page 103 (was a separate Technical Note)• "Generating a BIRT Report " on page 88 command-line utility <p>Updated:</p> <ul style="list-style-type: none">• "Maven Integration" on page 64• "Translating Java Bytecode" on page 28• "SCAState Utility Command-Line Options" on page 81• "Configuration Options" on page 113 <p>Removed: "About Command Line Builds in Visual Studio 6.0" (no longer supported)</p>

Chapter 1: Introduction

This guide provides instructions for using HPE Security Fortify Static Code Analyzer (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:

HPE Security Fortify Static Code Analyzer	11
About the Analyzers	12
Related Documents	13

HPE Security 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 HPE Security Fortify Audit Workbench or uploading them to HPE Security Fortify Software Security Center for analysis, or directly with the results displayed on screen

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

HPE Security Fortify CloudScan

You can use HPE Security Fortify CloudScan (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 *HPE Security Fortify CloudScan Installation, Configuration, and Usage Guide*.

HPE Security Fortify Scan Wizard

HPE Security 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 HPE Security 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 103](#).

About the Analyzers

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

The installation process downloads and updates the HPE Security Fortify security content (secure coding Rulepacks and external metadata) that Fortify Static Code Analyzer uses on your system. The Fortify Customer Portal provides updated security content on a regular basis.

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.

Analyzer	Description
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.
Semantic	The Semantic Analyzer detects potentially dangerous uses of functions and APIs at the intra-procedural level. Its specialized logic searches for buffer overflow, format string, and execution path issues, but is not limited to these categories. For example, the Semantic Analyzer detects deprecated functions in Java and unsafe functions in C/C++, such as <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 HPE Security Fortify Static Code Analyzer.

Note: The Protect724 site location is <https://www.protect724.hpe.com/community/fortify/fortify-product-documentation>.

All Products

The following documents provide general information for all products.

Document / File Name	Description	Location
<i>HPE Security Fortify Software System Requirements</i> HPE_Sys_Reqs_<version>.pdf	This document provides the details about the environments and products supported for this version of HPE Security Fortify Software.	Included with product download and on the Protect724 site
<i>HPE Security Fortify Software Release Notes</i> HPE_FortifySW_RN_<version>.txt	This document provides an overview of the changes made to HPE Security Fortify Software for this release and important information not included elsewhere in the product documentation.	Included on the Protect724 site
<i>What's New in HPE Security Fortify Software <version></i> HPE_Whats_New_<version>.pdf	This document describes the new features in HPE Security Fortify Software products.	Included on the Protect724 site
<i>HPE Security Fortify Open Source and Third-Party License Agreements</i> HPE_OpenSrc_<version>.pdf	This document provides open source and third-party software license agreements for software components used in HPE Security Fortify Software.	Included with product download and on the Protect724 site
<i>HPE Security Fortify Glossary</i> HPE_Glossary.pdf	This document provides definitions for HPE Security Fortify Software terms.	Included with product download and on the Protect724 site

HPE Security Fortify Software Security Center

The following documents provide information about HPE Security Fortify Software Security Center.

Document / File Name	Description	Location
<i>HPE Security Fortify Software Security Center User Guide</i> HPE_SSC_Guide_<version>.pdf HPE_SSC_Help_<version>	This document provides Fortify Software Security Center users with detailed information about how to use Fortify Software Security Center. It is intended for use by enterprise security leads,	Included with product download and on the Protect724 site

Document / File Name	Description	Location
	development team managers, and developers. Fortify Software Security Center provides security team leads with a high-level overview of the history and current status of a project.	
<i>HP Fortify Software Security Center User Guide: Legacy User Interface</i> HP_Fortify_SSC_User_Guide_Legacy.pdf PDF only; no help file	This document is the user guide for HP Software Security Center version 4.30. The legacy (4.30) user interface is available from the Fortify Software Security Center version 16.20 user interface. Specific areas of functionality are available only in the 4.30 interface.	Included with product download and on the Protect724 site
<i>HPE Security Fortify Software Security Center Process Designer Guide: Legacy User Interface</i> HPE_SSC_Proc_Design_Guide_Legacy_<version>.pdf HPE_SSC_Proc_Design_Help_<version>	This document provides information about how to start the Process Designer, configure its connection to your Fortify Software Security Center instance, and then use it to work with Fortify Software Security Center process templates, which are used only in the Fortify Software Security Center legacy (version 4.30) user interface.	Included with product download and on the Protect724 site
<i>HPE Security Fortify Software Security Center Installation and Configuration Guide</i> HPE_SSC_Install_<version>.pdf HPE_SSC_Install_Help_<version>	This document is written for users who are responsible for deploying and maintaining Fortify Software Security Center. It provides all of the information you need to acquire, install, and configure Fortify Software Security Center. This document is intended for users who are moderately knowledgeable about enterprise application development and skilled in enterprise system and database administration. It is written for: <ul style="list-style-type: none"> • System and instance administrators • Database administrators (DBAs) 	Included with product download and on the Protect724 site

Document / File Name	Description	Location
<i>HP Fortify Software Security Center Installation and Configuration Guide: Legacy User Interface</i> HP_Fortify_SSC_Install_and_Config_Guide_Legacy.pdf PDF only; no help file	This document provides system and database administrators with complete instructions on how to configure Fortify Software Security Center server software using the legacy (v4.30) user interface.	Included with product download and on the Protect724 site
<i>HPE Security Fortify Software Security Center Process Designer Guide: Legacy User Interface</i> HPE_SSC_Proc_Design_Guide_Legacy_<version>.pdf HPE_SSC_Proc_Design_Help_<version>	This legacy document provides information about how to start the Process Designer, configure its connection to your Fortify Software Security Center instance, and then use it to work with Fortify Software Security Center process templates.	Included with product download and on the Protect724 site

HPE Security Fortify Static Code Analyzer

The following documents provide information about Static Code Analyzer.

Document / File Name	Description	Location
<i>HPE Security Fortify Static Code Analyzer User Guide</i> HPE_SCA_Guide_<version>.pdf HPE_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.	Included with product download and on the Protect724 site
<i>HPE Security Fortify Static Code Analyzer Installation Guide</i> HPE_SCA_Install_<version>.pdf HPE_SCA_Install_Help_<version>	This document contains installation instructions for Fortify Static Code Analyzer and Applications.	Included with product download and on the Protect724 site

Document / File Name	Description	Location
<i>HPE Security Fortify Static Code Analyzer Performance Guide</i> HPE_SCA_Perf_Guide_ <version>.pdf PDF only; no help file	This document provides guidelines for selecting hardware to scan different types of codebases and offers tips for optimizing memory usage and performance.	Included with product download and on the Protect724 site
<i>HPE Security Fortify Static Code Analyzer Custom Rules Guide</i> HPE_SCA_Cust_Rules_Guide_ <version>.zip PDF only; no help file	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.	Included with product download

Technology Previews

Document / File Name	Description	Location
<i>HPE Security Fortify Static Code Analyzer Higher Order Analysis Technology Preview</i> HPE_SCA_HighOrderAnalysis_ TP_<version>.pdf PDF only; no help file	This document describes the Fortify Static Code Analyzer Higher Order Analyzer.	Included with product download and on the Protect724 site

Chapter 2: Analysis Process Overview

This section contains the following topics:

Analysis Process	18
Translation Phase	19
Mobile Build Sessions	20
Analysis Phase	21
Incremental Analysis	21
Parallel Processing	22
Translation and Analysis Phase Verification	22

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 61](#).
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 a .fpr file extension. For more information, see ["Analysis Phase" on page 21](#).
4. **Verification of the 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 22](#).

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).

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 type of source code describes 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 `-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 23](#)
- ["Translating .NET Code" on page 30](#)
- ["Translating C and C++ Code" on page 34](#)
- ["Translating JavaScript Code" on page 36](#)
- ["Translating Ruby Code" on page 39](#)
- ["Translating ABAP Code" on page 41](#)
- ["Translating Code for Mobile Platforms" on page 47](#)
- ["Translating Flex and ActionScript" on page 49](#)
- ["Translating COBOL Code" on page 52](#)
- ["Translating Other Languages" on page 55](#)

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.

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, 16.20.0140). 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, 16.10 and 16.1x 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 73](#) and ["Output Options" on page 70](#)).

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 *HPE Security 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
```

Parallel Processing

Fortify Static Code Analyzer supports parallel processing. If your project scan takes longer than an hour or two to complete, you can decrease the scan time by enabling parallel processing. Parallel processing takes advantage of multiple CPUs and cores within a single machine and automatic memory tuning.

For information about enabling parallel processing for your projects, see ["Parallel Analysis Mode" on page 97](#).

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 *HPE Security Fortify Audit Workbench User Guide*.

Chapter 3: Translating Java Code

This section contains the following topics:

Java Command-Line Syntax	23
Handling Resolution Warnings	26
Using FindBugs	26
Translating Java EE Applications	27
Translating Java Bytecode	28

Java Command-Line Syntax

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

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

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

With Java code, Fortify Static Code Analyzer can either emulate the compiler, which might be convenient for build integration, or 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 63.

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
```

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

- From the `-cp` option
- From `jre/lib`
- 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 more information, 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 Options	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>HPE Security Fortify Software System Requirements</i> document for supported versions. Equivalent property name: <code>com.fortify.sca.AppServerVersion</code>
<code>-cp <classpath> </code> <code>-classpath <classpath></code>	Specifies the class path to use for analyzing Java source code. The format is same as <code>javac</code> : a colon- or semicolon-separated list of paths. You can use Fortify Static Code Analyzer file specifiers as shown in the following example:

Java/Java EE Options	Description
	<pre>-cp "build/classes:lib/*.jar"</pre> <p>For information about file specifiers, see "Specifying Files" on page 77.</p> <p>Equivalent property name: <code>com.fortify.sca.JavaClasspath</code></p>
<pre>-extdirs <dirs></pre>	<p>Similar to the <code>javac 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>
<pre>-java-build-dir <dirs></pre>	<p>Specifies one or more directories to which Java sources have been compiled. You must specify this for FindBugs results as described in "Analysis Options" on page 73.</p>
<pre>-source <version> -jdk <version></pre>	<p>Indicates the JDK version for which the Java code is written. Valid values for <code><version></code> are 1.5, 1.6, 1.7, and 1.8. The default is 1.8.</p> <p>Equivalent property name: <code>com.fortify.sca.JdkVersion</code></p>
<pre>-sourcepath <dir></pre>	<p>Specifies the location of source files that are not included in the scan but are used for name resolution. The source path is similar to class path, except it uses source files rather than class files for resolution.</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 Fortify Static Code Analyzer 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 for translating Java files. For examples, see ["Java Command-Line Examples" on page 25](#).

Translating JSP Projects, Configuration Files, and Deployment Descriptors

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

For example:

```
sourceanalyzer -b <build_id> /**/*.jsp /**/*.xml
```

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

Java EE Translation Warnings

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

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

```
<list of .jsp file names>
```

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, HPE recommends that the bytecode is compiled with full debug information (`javac -g`).

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, include 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"
```

HPE 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 that you want scanned.

Chapter 4: Translating .NET Code

This chapter describes how to use Fortify Static Code Analyzer to translate Visual Studio .NET and ASP.NET applications built with Visual Studio. See the *HPE Security Fortify Software System Requirements* document for supported versions of Visual Studio.

Fortify Static Code Analyzer analyzes code written in C#, VB.NET, and ASP.NET (including .cshtml, .vbhtml, and .xaml files).

Note: The easiest way to analyze a .NET application is to use the HPE Security Fortify Package for Visual Studio, which automatically gathers all the required project information.

This section contains the following topics:

.NET Command-Line Syntax	30
Translating Simple .NET Applications	32
Handling Translation Errors	33

.NET Command-Line Syntax

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

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

```
sourceanalyzer -b my_buildid devenv Sample1.sln /REBUILD debug
```

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

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

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

```
sourceanalyzer -b my_buildid -scan -f results.fpr
```

.NET Command-Line Options

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

Note: These options are not required if you translate the code with the Visual Studio Command Prompt and you have HPE Security Fortify Package for Visual Studio installed.

.NET Options	Description
<code>-dotnet-version <version></code>	Specifies the .NET framework version. See the <i>HPE Security Fortify Software System Requirements</i> for a list of supported versions. This adds the location of .NET framework libraries (DLLs) for the specified .NET framework version to the list of directories/paths specified by the <code>-libdirs</code> option, unless the <code>-libdirs-only</code> option is specified.
<code>-libdirs <dirs> <paths></code>	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.
<code>-libdirs-only</code>	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.
<code>-dotnet-preproc-symbols <symbols></code>	Specifies a semicolon-separated list of preprocessor symbols used in the source code. For example: <div><code>-dotnet-preproc-symbols "DEBUG;TRACE"</code></div>
<code>-dotnet-assembly-name <assembly_name></code>	Specifies the name of the target .NET assembly as specified in Visual Studio project settings.
<code>-dotnetwebroot <root_dir></code>	.NET Web projects only. Specifies the home directory of an ASP.NET project.
<code>-cs-extern-alias <aliases_path_pairs></code>	C# projects only. Specifies a list of external aliases for a specified DLL file in the following format: <code>alias1,alias2,...=<path_to_DLL></code> . If multiple DLLs are assigned external aliases, specify multiple <code>-cs-extern-alias</code> options on the command line.
<code>-vb-root <namespace></code>	.VB.NET projects only. Specifies the root namespace for the project as specified in Visual Studio project settings.
<code>-vb-imports <namespaces></code>	.VB.NET projects only. Specifies a semicolon-separated list of namespaces imported for all source files in the project.

.NET Options	Description
-vb-mytype <symbol>	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 source code to be translated uses <code>My</code> namespace.
-vb-webproject	VB.NET projects only. Indicates that the project is a pure Web project (no code-behind the source files).
-vb-compile- options <compile_ options>	VB.NET projects only. Specifies any special compilation options required for the correct translation of the source code, such as <code>OptionStrict</code> , <code>OptionInfer</code> , and <code>OptionExplicit</code> . The format for <code><compile_options></code> is a comma-separated list of: <code><option>=On Off</code> . For example: <pre>-vb-compile-options "OptionStrict=On,OptionExplicit=Off"</pre>
-vsversion <version>	(Deprecated - Replaced by <code>-dotnet-version</code> option) Specifies the version number that corresponds to your Visual Studio version. <ul style="list-style-type: none">• Visual Studio 2012: 11.0• Visual Studio 2013: 12.0• Visual Studio 2015: 14.0

Translating Simple .NET Applications

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

To prepare your application for analysis, you need:

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

Note: To translate binaries instead of source files, perform a complete rebuild of your project with the Debug configuration enabled. Include the PDB and binary files in the Fortify Static Code Analyzer translation.

Run Fortify Static Code Analyzer to translate the .NET application from the command line as follows:

```
sourceanalyzer -dotnet-version <version> -b <build_id>  
-libdirs <ProjOne/Lib;ProjTwo/Lib> <project_1_source_files> <project_2_  
source_files>
```

Note: You can improve the translation by providing information using the command-line options described in [".NET Command-Line Options" on the previous page](#). Unless you specify the `-libdirs-only` option, Fortify Static Code Analyzer uses the .NET framework DLLs that

correspond to the target .NET framework version your project uses (specified with the `-dotnet-version` option) so you do not need to include them with the `-libdirs` option on the command line.

If your project is large, you can perform the translation phase separately for each project and use the same build ID, as follows:

```
sourceanalyzer -dotnet-version <version> -b <build_id>
-libdirs <paths> <project_1_source_files>
...
sourceanalyzer -dotnet-version <version> -b <build_id>
-libdirs <paths> <project_n_source_files>
```

where `<project_1_source_files>` and `<project_n_source_files>` are the output projects.

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 the Fortify Static Code Analyzer could not successfully translate all the source files in your project. Report this issue to HPE Security Fortify Technical 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 the Fortify Static Code Analyzer could not successfully translate all the ASP.NET pages in your project. Report this issue to HPE Security Fortify Technical Support for investigation.

Chapter 5: Translating C and C++ Code

This section contains the following topics:

Prerequisites	34
C and C++ Command-Line Syntax	34
Scanning Pre-processed C and C++ Code	35

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. To be sure that Fortify Static Code Analyzer interprets your source code in the same way as the compiler, the translation step 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 want to use, such as `gcc`, `g++`, or `cl`. See the *HPE Security 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 `sourceanalyzer` 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 my_buildid gcc -I. -o hello.o -c helloworld.c
```

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

Scanning Pre-processed C and C++ Code

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

Chapter 6: Translating JavaScript Code

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

This section contains the following topics:

Translating Pure JavaScript Projects	36
Translating JavaScript Projects with HTML Files	36
Including External JavaScript or HTML in the Translation	37
Scanning JavaScript Code	38

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>`.

Translating JavaScript Projects with HTML Files

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

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

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

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

If you set the `com.fortify.sca.EnableDOMModeling` property to `true`, you can also specify additional HTML tags for Fortify Static Code Analyzer to include in the DOM modeling using the `com.fortify.sca.DOMModeling.tags` property. By default, Fortify Static Code Analyzer includes a limited set of DOM element tags such as `html`, `head`, `input`, `button`, and `iframe`.

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

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

Including External JavaScript or HTML in the Translation

To include external JavaScript or HTML files that are specified with the `src` attribute, you can whitelist specific domains so that Fortify Static Code Analyzer downloads and includes them in the 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, if your HTML file includes the following:

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

If you are confident that the `xyzdomain` 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="xyz_domain/foo"
```

Fortify Static Code Analyzer supports subdomains. For example, if the `src` tag in the original HTML file specifies to download files from `foo.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/foo|abcdomain.com|123domain.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 at <https://docs.oracle.com/javase/8/docs/api/java/net/doc-files/net-properties.html>.

Scanning JavaScript Code

You can configure the Higher Order Analyzer for JavaScript analysis which improves the ability to track dataflow through higher-order code. However, adding analysis of JavaScript might result in long scan times. To configure the Higher Order Analyzer for JavaScript, set the `com.fortify.sca.Phase0HigherOrder.Languages` property 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
```

For more information about the Higher Order Analyzer, see the *HPE Security Fortify Static Code Analyzer Higher Order Analysis Technology Preview*.

Chapter 7: Translating Ruby Code

This section contains the following topics:

Ruby Command-Line Syntax	39
Adding Libraries	40
Adding Gem Paths	40

Ruby Command-Line Syntax

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

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

where *<rb_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	<p>Specifies one or more paths to directories that contain Ruby libraries (see "Adding Libraries" on the next page)</p> <p>Equivalent property name: <code>com.fortify.sca.RubyLibraryPaths</code></p>
-rubygem-path	<p>Specifies the path(s) to a RubyGems location (see "Adding Gem Paths" on the next page)</p> <p>Equivalent property name: <code>com.fortify.sca.RubyGemPaths</code></p>

Adding Libraries

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

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

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

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

Adding Gem Paths

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

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

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

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

If you have multiple `gems` directories, add them by specifying a delimited list of paths such as:

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

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

Chapter 8: Translating ABAP Code

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

This section contains the following topics:

About Scanning ABAP Code	41
Importing the Transport Request	42
Adding Fortify Static Code Analyzer to Your Favorites List	43
Running the HPE Security Fortify ABAP Extractor	44

About Scanning ABAP Code

To translate ABAP code, the HPE Security 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 permissions to download files to the local system and execute operating system commands.

Because the extractor program is executed online, you might receive a `max dialog work process time reached` exception if the volume of source files selected for extraction exceeds the allowable process run time. To work around this, download large projects as a series of smaller Extractor tasks. For example, if your project consists of four different packages, download each package separately into the same project directory.

If the exception occurs frequently, work with your SAP Basis administrator to increase the maximum time limit (`rdisp/max_wprun_time`).

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

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

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

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

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

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

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

INCLUDE Processing

As source code is downloaded, the HPE Security Fortify ABAP Extractor checks for `INCLUDE` statements in the source. When found, it downloads the include targets to the local machine for analysis as well.

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 HPE Security Fortify ABAP Extractor transport request on your SAP Server.

The HPE 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 HPE Security 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 and you must import it 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. Once imported into your system and properly configured, you can extract your code from the SAP database and prepare it for Fortify Static Code Analyzer scanning.

Installation Note

The HPE Security 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 a transport request import error: `Install release does not match the current version`.

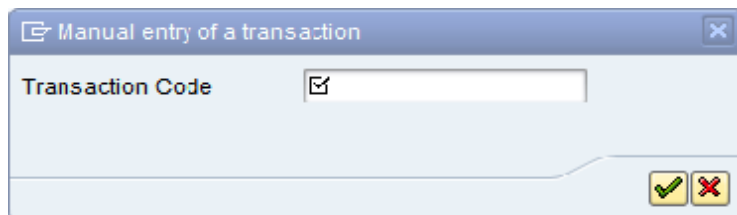
This causes the installation to fail. To resolve this issue:


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

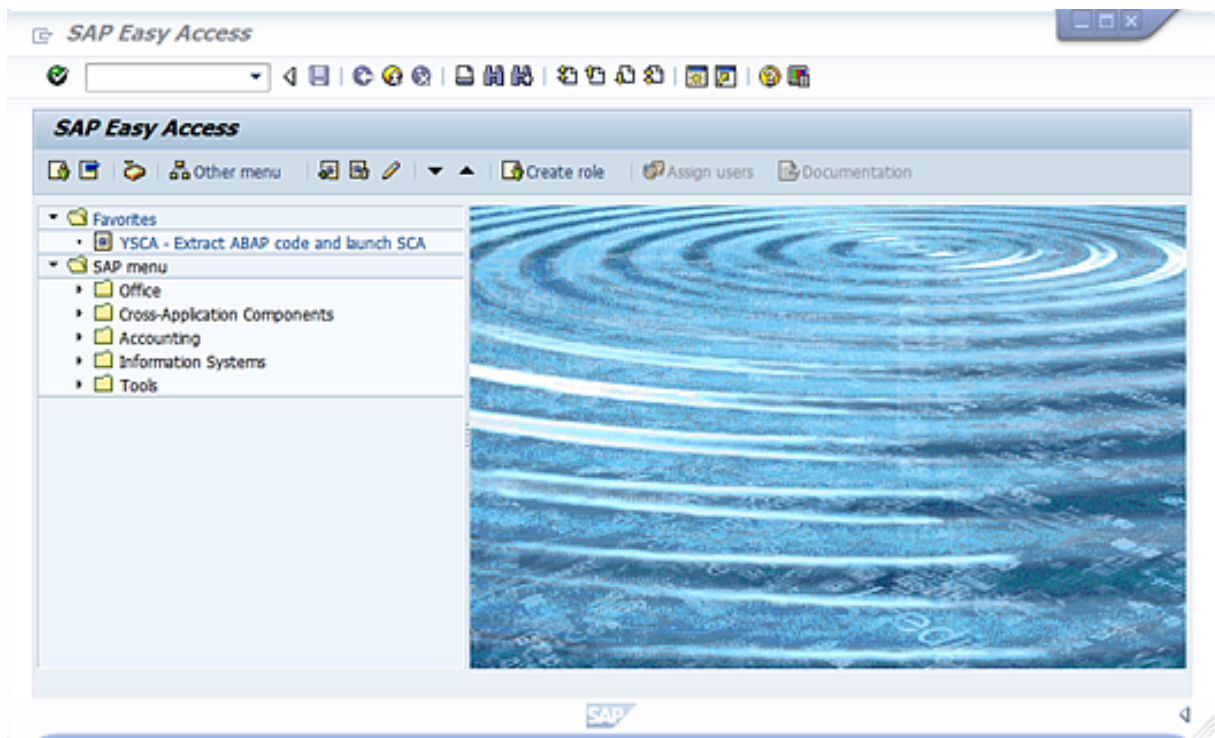
Adding Fortify Static Code Analyzer to your Favorites list is optional, but doing so can make it quicker to access and launch Fortify Static Code Analyzer scans. The following steps assume that you use the user menu in your day-to-day work. If your work is done from a different menu, add the Favorites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

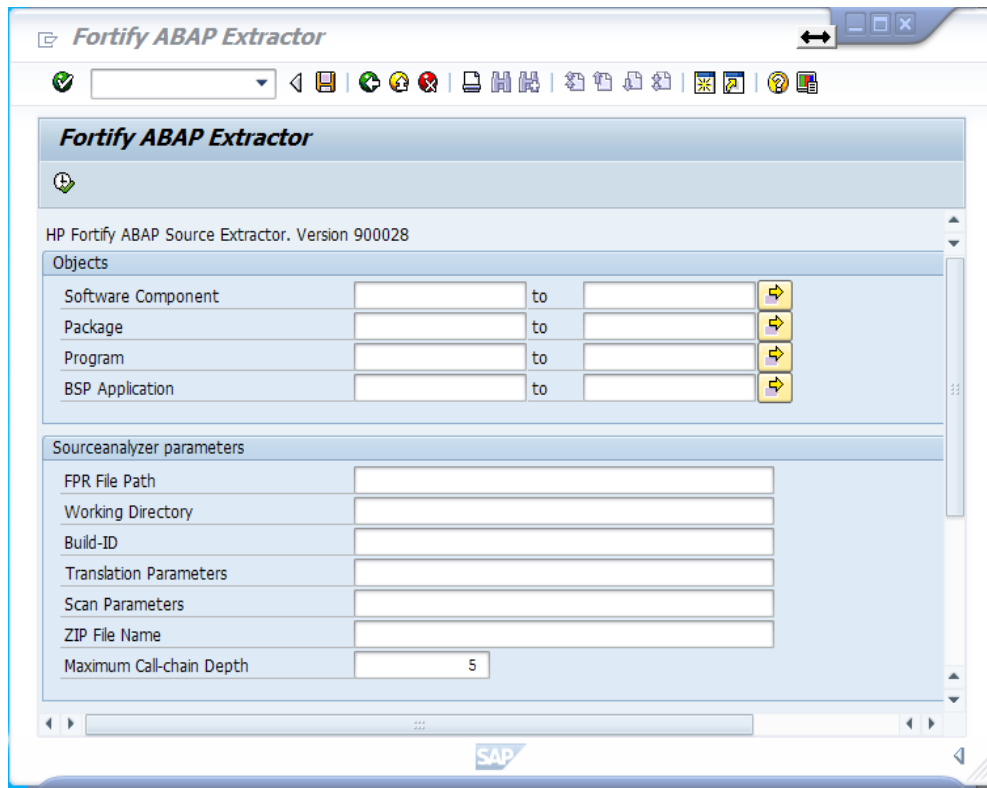


5. Click the **Extract ABAP code and launch SCA** link to launch the HPE Security Fortify ABAP Extractor.

Running the HPE Security Fortify ABAP Extractor

To run the HPE Security Fortify ABAP Extractor:

1. Start the program from the **Favorites** link, the transaction code, or manually start the YHP_FORTIFY_SCA object.



2. Fill in 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 you want assigned to 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: List any optional sourceanalyzer translation options.</p> <p>Scan Parameters: List any optional sourceanalyzer 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 which starts in explicitly-selected code and whose length is this number or less.</p>

Section	Data
Actions	<p>Download: Select this check box so that Fortify Static Code Analyzer downloads the source code extracted from your SAP database.</p> <p>Build: Select this check box so that Fortify Static Code Analyzer translates all downloaded ABAP code and stores it under 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**.

Chapter 9: Translating Code for Mobile Platforms

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

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

This section contains the following topics:

Translating Apple iOS Projects	47
Translating Android Projects	48

Translating Apple iOS Projects

This section describes how to translate Swift, Objective-C, and Objective-C++ source code for iOS applications. Fortify Static Code Analyzer automatically integrates with the Apple Xcode command-line tool to identify the project source files.

Prerequisites

- Install Xcode command-line tools on the path.
- Objective-C++ projects must use the non-fragile Objective-C runtime (ABI version 2 or 3).
- Use Apple's `xcode-select` command-line tool to set your Xcode path. Fortify Static Code Analyzer uses the system global Xcode configuration to find the Xcode toolchain and headers.
- Ensure that you have any dependencies required to build the project available.
- To translate Swift code, make sure that all third party modules, including cocoapods are available. Bridging headers must also be available, however, Xcode usually generates them automatically during the build.
- To translate Objective-C projects, ensure that the headers for third-party libraries are available. Fortify Static Code Analyzer translation does not require object files and static/dynamic library files.
- To translate WatchKit applications, make sure that you translate both the iPhone application target and the WatchKit extension target.

Xcodebuild Integration Command-Line Syntax

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

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

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

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

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

```
xcodebuild [<options>] clean  
sourceanalyzer -b my_buildid -clean  
sourceanalyzer -b my_buildid xcodebuild [<options>]
```

To scan the application artifact files, type:

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

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

Translating Android Projects

Translating Android projects is similar to translating Java code. See ["Translating Java Code" on page 23](#) for more information.

Chapter 10: Translating Flex and ActionScript

This section contains the following topics:

ActionScript Command-Line Syntax	49
Handling Resolution Warnings	51

ActionScript Command-Line Syntax

The basic command-line syntax for translating ActionScript is:

```
sourceanalyzer -b <build_id> -flex-libraries <listOfLibraries>  
<listOfFiles>
```

where:

<listOfLibraries> is a semicolon-separated list (Windows) or a colon-separated list (non-Windows systems) of library names to which you want to "link" and <listOfFiles> 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.

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>

Option	Description
<code>-flex-source-roots</code>	<p>A colon- or semicolon-separated list of root directories in which MXML sources are located. Normally, these contain a subfolder named <code>com</code>. For instance, if a Flex source root is given that is pointing to <code>foo/bar/src</code>, then <code>foo/bar/src/com/fortify/manager/util/Foo.mxml</code> is transformed into an object named <code>com.fortify.manager.util.Foo</code> (an object named <code>Foo</code> in the package <code>com.fortify.manager.util</code>).</p> <p>Equivalent property name: <code>com.fortify.sca.FlexSourceRoots</code></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 run-time model. As a consequence, you might get parse errors with MXML files. For instance, the XML parsing could fail, the 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 HPE Security Fortify Technical Support.

ActionScript Command-Line Examples

The following examples illustrate command-line syntax for typical scenarios for translating ActionScript.

Example 1

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

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

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

Example 2

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

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

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

Example 3

In this example, the Flex SDK root and Flex libraries are specified in a properties file because typing in the data is time consuming and the data is generally constant. Divide the application into two sections and store them in folders: a main section folder and a modules folder. Each folder contains a `src` folder where the paths start. File specifiers contain wild cards to pick up all the `.mxml` and `.as` files in both `src` folders. An MXML file in `main/src/com/foo/util/Foo.mxml` is translated as an ActionScript class named `Foo` in the package `com.foo.util`, for example, with the source roots specified here:

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

Handling Resolution Warnings

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

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

ActionScript Warnings

You might receive a message similar to the following:

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

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

Chapter 11: Translating COBOL Code

This section contains the following topics:

Preparing COBOL Source Files for Translation	52
COBOL Command-Line Syntax	53

For a list of supported technologies for translating COBOL code, see the *HPE Security 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 HPE license specific for COBOL scanning capabilities. Contact HPE Security for more information about scanning COBOL and the necessary license required.

Preparing COBOL Source Files for Translation

Fortify Static Code Analyzer runs only on the supported systems listed in the *HPE Security 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 running 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. HPE suggests 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 your translate 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 translate command structure, using a directory-file-path structure rather than 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 <directory-file-path>
```

Specify the directory and folder with all COBOL files as the parameter to Fortify Static Code Analyzer, and Fortify Static Code Analyzer processes 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. You can specify one or more file extensions (separate multiple extensions with colons). For example:

```
-copy-extensions <ext1>:<ext2>
```

COBOL Command-Line Syntax

Free-format COBOL is the default translation and scan mode for Fortify Static Code Analyzer. 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>
```

Working with Fixed-Format COBOL

Fortify Static Code Analyzer also supports fixed-format COBOL. When you translate and scan fixed-format COBOL, both the translation and scan command lines must include the `-fixed-format` command-line option. For example:

```
sourceanalyzer -b <build_id> -fixed-format
```

An example of the scan command-line syntax is:

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

If your COBOL code is IBM Enterprise COBOL, then it is most likely fixed-format. If the COBOL translation command appears to hang indefinitely, press **Ctrl + C** several times to terminate the translation, and then repeat the translation command with the `-fixed-format` option.

Searching for COBOL Copybooks

Use the `copydirs` command-line option to direct Fortify Static Code Analyzer to search a list of paths for copybooks and SQL `INCLUDE` files. For example:

```
sourceanalyzer -b <build_id> -copydirs c:\cobol\copybooks
```

Chapter 12: Translating Other Languages

This section contains the following topics:

Translating Python Code	55
Translating ColdFusion Code	56
Translating SQL	57
Translating ASP/VBScript Virtual Roots	58
Classic ASP Command-Line Example	60
VBScript Command-Line Example	60
PHP Command-Line Example	60

Translating Python Code

Fortify Static Code Analyzer translates Python applications, and processes files with the .py extension as Python source code. 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 provide this information to Fortify Static Code Analyzer with the -python-path option. In addition, some applications add additional import directories during runtime initialization.

To add paths for additional import directories, use the following Fortify Static Code Analyzer command-line option:

```
-python-path <pathname>
```

Note: Fortify Static Code Analyzer translates Python applications using all import files located in the directory path defined using the -python-path <pathname> option. Subsequently, translation might take a long time to complete.

Using the Django Framework with Python

To scan code created using the Django framework, set the following properties in the fortify-sca.properties configuration file:

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

For the translation phase, use the following option:

```
-django-template-dirs <path/to/template/dirs>
```

Python Command-Line Options

The following table describes the Python options.

Python Option	Description
<code>-python-path <path></code>	<p>Specifies the path for additional import directories. Fortify Static Code Analyzer does not respect the PYTHONPATH environment variable that the Python runtime system uses to find imported files. Use the <code>-python-path</code> option to specify additional import directories.</p> <p>Equivalent property name: <code>com.fortify.sca.PythonPath</code></p>
<code>-django-template-dirs <path></code>	<p>Specifies to scan code created using the Django framework where <code><path></code> is the location of the Django template directories.</p> <p>Equivalent property name: <code>com.fortify.sca.DjangoTemplateDirs</code></p>

Translating ColdFusion Code

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

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

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

ColdFusion Command-Line Syntax

Type the following to translate ColdFusion source code:

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

where:

- `<build_id>` specifies the build ID for the project
- `<dir>` specifies the root directory of the web application
- `<files> | <file specifiers>` specifies the CFML source code files

Note: Fortify Static Code Analyzer calculates the relative path to each CFML source file with the

-source-base-dir directory as the starting point. Fortify Static Code Analyzer uses these relative paths when it generates instance IDs. If you move the entire application source tree to a different directory, the Fortify Static Code Analyzer-generated instance IDs remain the same if you specify an appropriate argument for the -source-base-dir option.

For a description of how to use *<file specifiers>*, see ["Specifying Files" on page 77](#).

ColdFusion Command-Line Options

The following table describes the ColdFusion options.

ColdFusion Option	Description
-source-base-dir <i><web_app_root_dir></i> <i><files></i> <i><file specifiers></i>	The web application root directory. Equivalent property name: com.fortify.sca.SourceBaseDir

Translating SQL

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

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

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

or

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

PL/SQL Command-Line Example

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

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

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

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

T-SQL Command-Line Example

The following example shows the syntax 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 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.<name_of_virtual_directory>=<full path to corresponding physical directory>
```

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

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

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

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

Note: The previous version of the ASPVirtualRoot property is still valid. You can use it on the Fortify Static Code Analyzer command line as follows:

```
-Dcom.fortify.sca.ASPVirtualRoots=C:\WebServer\CustomerTwo\Stuff;  
C:\WebServer\CustomerOne\inc
```

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

Using Virtual Roots Example

You have a file as follows:

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

To specify this file, use the following include:

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

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

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

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

The sequence to specify virtual roots is as follows:

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

Classic ASP Command-Line Example

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

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

VBScript Command-Line Example

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

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

PHP Command-Line Example

To translate a single PHP file named `MyPHP.php`, type:

```
sourceanalyzer -b mybuild "MyPHP.php"
```

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

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

where `<path>` is the absolute or relative path to the project root directory. The relative path name expands from the PHP project root directory.

Chapter 13: Integrating into a Build

You can integrate the analysis into supported build tools.

This section contains the following topics:

Build Integration	61
Modifying a Build Script to Invoke Fortify Static Code Analyzer	62
Touchless Build Integration	63
Ant Integration	63
Gradle Integration	64
Maven Integration	64
MSBuild Integration	67

Build Integration

You can translate entire projects in a single operation. Prefix your original build operation with the `sourceanalyzer` command followed by the Fortify Static Code Analyzer options. For information about integrating with Xcodebuild, see ["Xcodebuild Integration Command-Line Syntax" on page 47](#).

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 *HPE Security 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 subprocess that then executes the compiler

If you cannot meet these requirements in your environment, see ["Modifying a Build Script to Invoke Fortify Static Code Analyzer" on the next page](#).

Make Example

If you build your project with the following build commands:

```
make clean
make
make install
```

then you can simultaneously translate and compile the entire project with the following commands:

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

Devenv Example

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

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

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

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

Note: Integration with Visual Studio .NET requires that you have installed the HPE Security Fortify Package for Visual Studio for your specific version of Visual Studio.

Modifying a Build Script to Invoke Fortify Static Code Analyzer

As an alternative to build integration, you can modify your build script to prefix each compiler, linker, and archiver operation with the `sourceanalyzer` command. For example, a makefile often defines variables for the names of these tools:

```
CC=gcc
CXX=g++
LD=ld
AR=ar
```

You can prepend the tool references in the makefile with the `sourceanalyzer` command and the appropriate Fortify Static Code Analyzer options.

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

When you use the same build ID for each operation, Fortify Static Code Analyzer automatically combines each of the separately-translated files into a single translated project.

Touchless Build Integration

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

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

For example, if you 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 system described earlier in this chapter 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 requiring 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 *HPE Security Fortify Software System Requirements* document for platforms and languages supported specifically for Gradle integration. Any files in the project that are use 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

Fortify Static Code Analyzer includes a Maven plugin which provides a way for you 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 Maven Plugin

The 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 *HPE Security Fortify Software System Requirements* document.

If you have a previous version of the Maven Plugin installed, install the new version to upgrade to the latest version.

Uninstalling the Maven Plugin

To uninstall the Maven Plugin, manually delete all files from the `<Maven_Local_Repository>/repository/com/fortify/ps/maven/plugin` directory.

Testing the Maven Plugin Installation

After you install the Maven Plugin, use one of the included sample files to be sure your installation works properly.

To test the 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.
It should display the Fortify Static Code Analyzer version information.
3. Navigate to the sample Eightball directory: `<root_directory>/samples/EightBall`.
4. Type the following command:

```
mvn com.hpe.security.fortify.maven.plugin:sca-maven-plugin:<ver>:clean
```

where `<ver>` is the version of the Maven Plugin you are using. If the version is not specified, Maven uses the latest version of the Maven Plugin that is installed in the local repository.

Note: To determine the version of the Maven Plugin, open the `pom.xml` file located that you extracted in `<root_directory>` in a text editor. The Maven Plugin version is specified in the `<version>` element.

5. If the command in step 4 completed successfully, then the Maven Plugin is installed correctly. The Maven Plugin is not installed correctly, if you get the following error message:

```
[ERROR] Error resolving version for plugin  
'com.hpe.security.fortify.maven.plugin:sca-maven-plugin' from the  
repositories
```

Check the Maven local repository and try to install the Maven Plugin again.

Using the Maven Plugin

You can use the 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 Maven Plugin. The following table describes where to find the documentation after the Maven Plugin is properly installed.

Package Type	Documentation Location
Binary	<code><root_directory>/docs/usage.html</code>
Source	<code><root_directory>/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 with one of the following options:

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

4. Run the scan:

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

Add the `-D` option with the following property to the translate step to specify the files you want to exclude:

```
-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 1.6 project, issue the following command to translate the source code:

```
mvn com.hpe.security.fortify.maven.plugin:sca-maven-plugin:<ver>:translate  
-Dfortify.sca.source.version=1.6 -Dfortify.sca.exclude="fileA;fileB;fileC;
```

MSBuild Integration

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

- C/C++ Console Applications (only Visual Studio 2012 and later)
- C/C++ Libraries (only Visual Studio 2012 and later)
- Visual C# and Visual Basic Websites
- Visual C# and Visual Basic Libraries
- Visual C# and Visual Basic web Applications
- Visual C# and Visual Basic Console Applications

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

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

Setting Windows Environment Variables for Touchless MSBuild Integration

When you integrate Fortify Static Code Analyzer into your MSBuild process, there are several Windows environment variables that you can set. If you do not set these variables, Fortify Static Code Analyzer assumes a default set of variables and uses those. After you set the appropriate Windows environment variables, successive builds use the same set until you change the environment variables. The following table lists the environment variables that you can set.

Environment Variable	Description	Default Value
FORTIFY_MSBUILD_BUILDID	Specifies the Fortify Static Code Analyzer build ID	Build ID specified on the command line
FORTIFY_MSBUILD_DEBUG	Puts the logger and Fortify Static Code Analyzer into debug mode	False
FORTIFY_MSBUILD_MEM	Specifies the memory used to invoke Fortify Static Code Analyzer (for example, -Xmx2000M)	Automatic allocation based on physical memory available on the system
FORTIFY_MSBUILD_LOG	Specifies the location for the log file	\${win32.LocalAppdata}/Fortify/MSBuildPlugin
FORTIFY_MSBUILD_SCALOG	Specifies the location (absolute path) for the Fortify Static Code Analyzer log	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

Touchless MSBuild integration requires the `FortifyMSBuildTouchless.dll` located in the `<sca_install_dir>/Core/lib` directory. You must run this command from a Visual Studio Command Prompt.

The following is an example of the command to run the build and a Fortify Static Code Analyzer analysis using the default environment variables, or those you have previously set:

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

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

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

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	70
Translation Options	72
Analysis Options	73
Other Options	75
Directives	76
Specifying Files	77

Output Options

The following table describes the output options.

Output Option	Description
<code>-f <file></code>	<p>Specifies the file to which results are written. If you do not specify an output file, the output is written to the terminal.</p> <p>Equivalent property name: <code>com.fortify.sca.ResultsFile</code></p>
<code>-format <format></code>	<p>Controls the output format. Valid options are <code>fpr</code>, <code>fvd1</code>, <code>text</code>, and <code>auto</code>. The default is <code>auto</code> which selects the output format based on the file extension of the file provided with the <code>-f</code> option.</p> <p>Note: If you use result certification, you must specify the <code>fpr</code> format. See the <i>HPE Security Fortify Audit Workbench User Guide</i> for information on result certification.</p> <p>Equivalent property name: <code>com.fortify.sca.Renderer</code></p>
<code>-append</code>	<p>Appends results to the file specified with the <code>-f</code> 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 <code>fpr</code> or <code>fvd1</code>. For information on the <code>-format</code> output option, see the description in this table.</p>

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, HPE 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 it is included 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 it is included 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 it is included 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 77 for more information on how to use file specifiers.</p> <p>For example:</p> <pre>sourceanalyzer -cp "**/*.jar" "**/*" -exclude "**/Test.java"</pre> <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> (https://docs.oracle.com/javase/8/docs/api/java/nio/charset/Charset.html)</p> <p>Typically, if you do not specify the encoding type, Fortify Static Code Analyzer uses <code>file.encoding</code> via the <code>java.io.InputStreamReader</code> constructor with no encoding argument. 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>
<code>-noextension-type <file_type></code>	<p>Specifies the file type for source files that have no file extension. The possible values are: ABAP, ACTIONSCRIPT, ASPNET, ARCHIVE, ASP, ASPX, BITCODE, BYTECODE, CFML, COBOL, CSHARP, HTML, JAVA, JAVA_PROPERTIES, JAVASCRIPT, JSP, JSPX, MSIL, MXML, PHP, PLSQL, PYTHON, RUBY, RUBY_ERB, SWIFT, TLD, SQL, TSQL, VB, VB6, VBSCRIPT, and XML.</p>

Analysis Options

The following table describes the analysis options.

Analysis Option	Description
<code>-b <build_id></code>	<p>Specifies the build ID.</p> <p>Equivalent property name: <code>com.fortify.sca.BuildID</code></p>
<code>-scan</code>	<p>Causes Fortify Static Code Analyzer to perform analysis for the specified build ID.</p>
<code>-quick</code>	<p>Scans the project in Quick Scan mode, using the <code>fortify-sca-quickscan.properties</code> file. By default, this scan searches for high-confidence, high-severity issues that Fortify Static Code Analyzer can discover quickly.</p> <p>Equivalent property name: <code>com.fortify.sca.QuickScanMode</code></p>
<code>-bin <binary></code>	<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. You can use this option multiple times to specify the inclusion of multiple binaries in the scan.</p> <p>Equivalent property name: <code>com.fortify.sca.BinaryName</code></p>
<code>-disable-default-rule-type <type></code>	<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 <code>Rule</code>. For example, use <code>DataflowSource</code> for <code>DataflowSourceRule</code> elements. You can also specify specific sections of characterization rules, such as <code>Characterization:Control flow</code>, <code>Characterization:Issue</code>, and <code>Characterization:Generic</code>.</p> <p>The <code><type></code> argument is case-insensitive.</p>
<code>-exit-code-level</code>	<p>Extends the default exit code options. See "Exit Codes" on page 93 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.

Analysis Option	Description
	<ul style="list-style-type: none"> errors—Returns exit codes 0, 1, 2, 3, or 5. no_output_file—Returns exit codes 0, 1, 2, 3, or 6. <p>Equivalent property name: com.fortify.sca.ExitCodeLevel</p>
-filter <file>	<p>Specifies a results filter file.</p> <p>Equivalent property name: com.fortify.sca.FilterFile</p>
-findbugs	<p>Enables FindBugs analysis for Java code. You must specify the Java class directories with the -java-build-dir option, which is described in "Java Command-Line Options" on page 24.</p> <p>Equivalent property name: com.fortify.sca.EnableFindbugs</p>
-incremental-base	<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 -incremental option. See "Incremental Analysis" on page 21 for more information about performing incremental analysis.</p>
-incremental	<p>Specifies that this is a subsequent scan of a project for which you have already run an full base scan with the -incremental-base option. See "Incremental Analysis" on page 21 for more information about performing incremental analysis.</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> <div> <p>Note: This equivalent to disabling the following rule types: DataflowSink, Semantic, Controlflow, Structural, Configuration, Content, Statistical, Internal, and Characterization:Issue.</p> </div> <p>Equivalent property name: com.fortify.sca.NoDefaultIssueRules</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 no rules are processed.</p> <p>Equivalent property name: com.fortify.sca.NoDefaultRules</p>
-no-default-source-rules	<p>Disables source rules in the default Rulepacks.</p>

Analysis Option	Description
	<p>Note: Characterization source rules are not disabled.</p> <p>Equivalent property name: <code>com.fortify.sca.NoDefaultSourceRules</code></p>
<code>-no-default-sink-rules</code>	<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>
<code>-project-template</code>	<p>Specifies the issue template file you want 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>
<code>-rules <file> <directory></code>	<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 the <code>.bin</code> and <code>.xml</code> extensions.</p> <p>Equivalent property name: <code>com.fortify.sca.RulesFile</code></p>

Other Options

The following table describes other options.

Other Option	Description
<code>@<filename></code>	Reads command-line options from the specified file.
<code>-h -? -help</code>	Prints a summary of command-line options.
<code>-debug</code>	<p>includes debug information in the log file which is useful for troubleshooting.</p> <p>Equivalent property name: <code>com.fortify.sca.Debug</code></p>
<code>-debug-verbose</code>	<p>This is the same as the <code>-debug</code> option, but it includes more details, specifically for parse errors.</p> <p>Equivalent property name: <code>com.fortify.sca.DebugVerbose</code></p>

Other Option	Description
-logfile <file>	Specifies the log file that Fortify Static Code Analyzer creates. Equivalent property name: com.fortify.sca.LogFile
-clobber-log	Specifies that Fortify Static Code Analyzer overwrites the log file for each new scan. Equivalent property name: com.fortify.sca.ClobberLogFile
-quiet	Disables the command-line progress information. Equivalent property name: com.fortify.sca.Quiet
-verbose	Sends verbose status messages to the console and to the log file. Equivalent property name: com.fortify.sca.Verbose
-version	Displays the Fortify Static Code Analyzer version number.
-autoheap	Enables automatic allocation of memory based on the physical memory available on the system. This is the default memory allocation setting.
-Xmx<size>M G	Specifies the maximum amount of memory Fortify Static Code Analyzer uses. When you specify this option, make sure that you do not allocate more memory than is physically available, because this degrades performance. As a guideline, assuming no other memory intensive processes are running, do not allocate more than 2/3 of the available memory.

Directives

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

Directive	Description
-clean	Deletes all Fortify Static Code Analyzer intermediate files and build records. When a build ID is also specified, only files and build records relating to that build ID are deleted.

Directive	Description
-show-binaries	Displays all objects that were created but not used in the production of any other binaries. If fully integrated into the build, it lists all of the binaries produced.
-show-build-ids	Displays a list of all known build IDs.
-show-build-tree	When you scan with the -bin option, displays all files used to create the binary and all files used to create those files in a tree layout. If the -bin option is not present, the tree is displayed for each binary. Note: This option can generate an extensive amount of information.
-show-build-warnings	Use with -b <build_id> to show all errors and warnings from the translation phase on the console. Note: Audit Workbench also displays these errors and warnings in the results certification panel.
-show-files	Lists the files in the specified build ID. When the -bin option is present, displays only the source files that went into the binary.
-show-loc	Displays the number of lines in the code being translated.

Specifying Files

File specifiers are expressions that allow you to pass a long list of files to Fortify Static Code Analyzer using wild card characters. Fortify Static Code Analyzer recognizes two types of wild card characters: a single asterisk character (*) matches part of a file name, and double asterisk characters (**) recursively matches directories. You can specify one or more files, one or more file specifiers, or a combination of files and file specifiers.

<files> | <file specifiers>

The following table describes the different file specifiers forms.

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

File Specifier	Description
<dirname>	All files found in the named directory or any subdirectories.
<dirname>/**/Example.java	Any file named Example.java found in the named directory or any subdirectories.

File Specifier	Description
<code><dirname>/*.java</code>	Any file with the extension <code>.java</code> found in the named directory.
<code><dirname>/**/*.java</code>	Any file with the extension <code>.java</code> found in the named directory or any subdirectories.
<code><dirname>/**/*</code>	All files found in the named directory or any subdirectories (same as <code><dirname></code>).

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	79
Checking the Fortify Static Code Analyzer Scan Status	80
Working with FPR Files from the Command Line	82
Generating Reports from the Command Line	88
About Updating Security Content	90

Fortify Static Code Analyzer Utilities

Fortify Static Code Analyzer command-line utilities enable you to manage Rulepacks, FPR files, run reports, and other system-level processes. These utilities are located in `<sca_install_dir>/bin`. The utilities are provided as `.bat` or `.cmd` files (Windows) or `.sh` scripts (Unix).

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	Merges audited projects, verifies FPR signatures, migrates audit projects from previous versions to newer formats, displays mappings for a migrated project, displays any errors associated with an FPR, displays the number of issues in an FPR, combines or splits source code files and audit projects into FPR files.	"Working with FPR Files from the Command Line" on page 82
BIRTReportGenerator and ReportGenerator	Generates BIRT reports and legacy reports from FPR files.	"Generating Reports from the Command Line" on page 88

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 90

Other Command-Line Utilities

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

Utility	Description	For More Information
fortifyclient	Use this utility to create fortifyclient authentication tokens, upload and download FPR files, and archive and restore runtime events.	See the <i>HPE Security Fortify Software Security Center Installation and Configuration 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.	See the <i>HPE Security 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 from which 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.
-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. HPE 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 for an FPR File" on page 84](#)
- ["Migrating Audit Data from Previous FPR Versions" on page 86](#)
- ["Extracting a Source Archive from an FPR File" on page 87](#)

Merging FPR Files

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

To merge FPR files:

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

To merge FPR files and set instance ID migrator options:

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

FPRUtility Data Merge Options

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

Option	Description
-merge	Merges the specified project and source FPR files.
-project <primary.fpr>	Specifies the primary FPR file that you want to merge. Conflicts are resolved using the values in this file.
-source <secondary.fpr>	Specifies the secondary FPR file that you want to merge. The primary project overrides values if conflicts exist.
-f <output.fpr>	Specifies the name of the merged output file. This file is the result of the merged files. Note: When you specify this option, neither of the original FPR files are modified. If you do not use this option, the primary FPR is overwritten with the merged results.
-forceMigration	Forces the migration, even if the engine and the Rulepack versions of the two projects are the same.
-useMigrationFile <mapping_file>	Specifies an instance ID mapping file. This enables you to modify mappings manually rather than using the migration results. Supply your own instance ID mapping file.
-useSourceIssueTemplate	Specifies to use the filter sets and folders from the issue template in the secondary FPR. By default, Fortify Static Code Analyzer uses the filter sets and folders from the issue template in the primary FPR.

Option	Description
<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

When the FPRUtility `-merge` command finishes, it displays one of the exit codes listed in the following table.

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

Displaying Analysis Results for an FPR File

The FPRUtility `-information` option displays information about the project. You can obtain information to validate signatures, view mappings for a migrated project, examine any errors associated with the FPR, and obtain the number of issues for each analyzer, vulnerability category, or custom grouping.

To display project signature information:

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

To display the migration mappings report:

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

To display a full analysis error report for the FPR:

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

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

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

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

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

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

FPRUtility Information Options

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

Option	Description
-information	Displays information for the project.
One of: -signature -mappings -errors -categoryIssueCounts -analyzerIssueCounts -search -query <search_expression> [-categoryIssueCounts] [-analyzerIssueCounts] [-includeSupressed] [-includeRemoved]	<p>The <code>-signature</code> option displays the signature.</p> <p>The <code>-mappings</code> option displays the migration mappings report.</p> <p>The <code>-errors</code> option displays a full error report for the FPR.</p> <p>The <code>-analyzerIssueCounts</code> option displays the number of issues for each analyzer.</p> <p>The <code>-categoryIssueCounts</code> option displays the number of issues for each vulnerability category.</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>-includeSupressed</code> and <code>-includeRemoved</code> options to include suppressed or removed issues.</p>

Option	Description
<code>-project <project.fpr></code>	Specifies the FPR from which to extract the results information.
<code>-f <output.txt></code>	Specifies the output file. The default is <code>System.out</code> .

FPRUtility Signature Exit Codes

When the FPRUtility `-information -signature` command finishes, one of the exit codes listed in the following table is displayed.

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.

Migrating Audit Data from Previous FPR Versions

The FPRUtility `-migrate` option uses a specified template file to migrate a specified FPR file from an earlier version and update the analysis information to the new audit project format.

To migrate audit data:

```
FPRUtility -migrate -project <OldProject.fpr> -settings  
<Settings.properties> -f <output.fpr>
```

FPRUtility Options for Migrating Audit Data

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

Option	Description
<code>-migrate</code>	Migrates the FPR to the new format.
<code>-project <Oldproject.fpr></code>	Specifies the name of the FPR you want to migrate.
<code>-settings <Settings.properties></code>	Specifies the migration settings file to use. This properties file is the template that is applied to the FPR. It contains the format settings for the new audit project.
<code>-f <output.fpr></code>	Specifies the output file.

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 either extract the source code from an FPR file, or merge an existing source archive (FSA) back into an FPR file.

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 <outputFolder>
```

To add an archive to an FPR file:

```
FPRUtility -sourceArchive -mergeArchive -project <project.fpr> \  
-source <oldSourceArchive.fsa> -f <ProjectWithArchive.fpr>
```

FPRUtility Options for Archiving FPR Files

The following table lists the FPRUtility options that apply to archiving FPR files.

Option	Description
<code>-sourceArchive</code>	Creates an FSA file so that you can extract a source archive.
One of: <code>-extract</code> <code>-mergeArchive</code>	The <code>-extract</code> option extracts the contents of the FPR file. The <code>-mergeArchive</code> option merges the contents of the FPR file with an existing archived file.
<code>-project <project.fpr></code>	Specifies the FPR that you want to archive.
<code>-source <oldSourceArchive.fsa></code>	Specifies the name of the existing archive. Use only if you are merging an FPR file with an existing archive (<code>-mergeArchive</code> option).
<code>-recoverSourceDirectory</code>	Use with the <code>-extract</code> option to extract the source as a folder with restored source files.
<code>-f <ProjectWithArchive.fpr> <outputArchive.fsa> <outputFolder></code>	Specifies the output file. You can generate an FPR, a folder, or an FSA file.

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 *HPE Security 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 <auditedProject.fpr>  
-format PDF|DOC|HTML|XLS -output <reportFile>
```

An example of how to generate a OWASP Top 10 2010 report with additional options:

```
BIRTReportGenerator -template "OWASP Top 10" -source auditedProject.fpr  
-format PDF -showSuppressed --Version "OWASP Top 10 2013"  
--UseFortifyPriorityOrder -output MyOWASP_Top10_Report.pdf
```

BIRTReportGenerator Command-Line Options

The following table lists the BIRTReportGenerator options.

Option	Description
-template <template_name>	Specifies the report template name. The valid values are: "Developer Workbook", "DISA STIG", "CWE/SANS Top 25", "FISMA Compliance", "OWASP Mobile Top 10", "OWASP Top 10", and "PCI DSS Compliance".
-source <auditedProject.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 <resultsfile.***>	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 have been marked as suppressed.

Option	Description
-showRemoved	Include issues that have been marked as removed.
-showHidden	Include issues that have been 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> <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 4.1", "DISA STIG 3.10", "DISA STIG 3.9", "DISA STIG 3.7", "DISA STIG 3.5", "DISA STIG 3.4", and "DISA STIG 3"</p> <p>For the "OWASP Top 10" template:</p> <p>"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 "Description of Key Terminology" section in the report.
--IncludeHPEnterpriseSecurity	Include the "About HPE Security" section in the report.
--SecurityIssueDetails	Provide detailed descriptions of reported issues. This option is not available for the Developer Workbook template.
--UseFortifyPriorityOrder	Use Fortify Priority Order instead of folder names to categorize issues. This option is not available for the Developer Workbook and PCI Compliance templates.

Generating a Legacy Report

To generate a report (PDF):

```
ReportGenerator -format pdf -f <resultsFile.pdf> -source  
<auditedProject.fpr>
```

To generate a report (XML):

```
ReportGenerator -format XML -f <resultsFile.xml> -source  
<auditedProject.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 <auditedProject.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 HPE Security 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 HPE 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

The `fortifyupdate` utility is located in the `<scq_install_dir>/bin` directory. To update your Fortify Static Code Analyzer installation with the latest HPE Security Fortify Secure Coding Rulepacks and external metadata, type the following command:

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

fortifyupdate Command-Line Options

The following table lists the `fortifyupdate` options.

Option	Description
<code>-import <file.zip></code>	Imports the zip file that contains archived security content.
<code>-acceptKey</code>	Accept the public key. When this is specified, you are not prompted to provide a public key.
<code>-coreDir <directory></code>	Specifies the core directory where the update is stored.
<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>HPE Security Fortify Static Code Analyzer Tools Properties Reference Guide</i> .
<code>-proxyhost <host></code>	Specifies a proxy server network name or IP address.
<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.

Option	Description
<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>HPE Security Fortify Static Code Analyzer Tools Properties Reference Guide</i>.</p>

Chapter 16: Troubleshooting

This section contains the following topics:

Exit Codes	93
Using the Log File to Debug Problems	94
Translation Failed Message	94
JSP Translation Problems	95
C/C++ Precompiled Header Files	95
Reporting Issues and Requesting Enhancements	96

Exit Codes

The following table describes the possible Fortify Static Code Analyzer exit codes.

Exit Code	Description
0	Success
1	Generic failure
2	Invalid input files (this could indicate that an attempt was made to translate a file that has a file extension that Fortify Static Code Analyzer does not support)
3	Process timed out
4	Analysis completed with numbered warning messages written to the console and/or to the log file
5	Analysis completed with numbered error messages written to the console and/or to the log file
6	Scan phase was unable to generate issue results

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

You can extend the default exit code options by setting the `com.fortify.sca.ExitCodeLevel` property in the `<sca_install_dir>/Core/Config/fortify-sca.properties` file.

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

The valid values are:

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

Using the Log File to Debug Problems

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

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

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

Email the `sca.log` file as a zip file to HPE Security Fortify Technical Support.

Translation Failed Message

If your C or C++ application builds successfully but you see one or more “translation failed” messages when building with Fortify Static Code Analyzer, 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 build 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 HPE Security Fortify Technical Support for further investigation.

JSP Translation Problems

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

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

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

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

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

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

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

For more information about scanning Java EE applications, see ["Translating Java EE Applications" on page 27](#).

C/C++ Precompiled Header Files

Some C/C++ compilers support a feature termed “precompiled header files,” which can speed up compilation. 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 the precompiled header feature of your compiler, disable precompiled headers and perform a full build to be sure that your source code compiles cleanly.

Reporting Issues and Requesting Enhancements

Feedback is critical to the success of this product. To request enhancements or patches, or to report issues, send an email to HPE Security Fortify Technical Support at FortifyTechSupport@hpe.com.

Include the following information in the email body:

- 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 fix the issue. Also include the log files, or the relevant portions of them, from when the issue occurred.

Appendix A: Parallel Analysis Mode

With parallel processing mode, you can reduce scan times by harnessing the multiple cores, memory, and processing power in your machine.

While you can enable parallel analysis mode for all scans, scans that complete in less than two hours might not warrant the higher processing power requirements. For this reason, parallel analysis mode is not the default mode of operation. You must enable parallel processing on your system and initiate it on the command line.

For the current hardware and software requirements needed to run in parallel analysis mode, see the *HPE Security Fortify Software System Requirements* document.

This section contains the following topics:

Configuring Parallel Analysis Mode	97
Running in Parallel Analysis Mode	98
Strategies for Best Use	98

Configuring Parallel Analysis Mode

After you install Fortify Static Code Analyzer and complete the post-installation steps, you can add two properties to your Fortify Static Code Analyzer configuration file to configure parallel analysis mode.

Add the following properties to your `fortify-sca.properties` file, located in the `<sca_install_dir>/core/config` directory.

Property	Description
<code>com.fortify.sca.RmiWorkerMaxHeap</code>	<p>Overrides the default heap size for the workers. The amount of memory required varies from project to project, but you do not have to allocate as much memory for the workers as you do for the master JVM. This property accepts values in kilobytes (K), megabytes (M), or Gigabytes (G).</p> <p>You might need to experiment with this property if you experience warnings about low memory, crashes, or do not achieve a speed increase.</p> <p>Value type: String</p> <p>Default: If <code>-Xmx</code> is not specified (recommended), then Fortify Static Code Analyzer automatically determines a reasonable worker heap size. If <code>-Xmx</code> is specified, then the default worker heap size is the <code>-Xmx</code> value.</p> <p>Example: <code>com.fortify.sca.RmiWorkerMaxHeap=500M</code></p>

Property	Description
<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>

Running in Parallel Analysis Mode

To run Fortify Static Code Analyzer in parallel analysis mode, add the following option to your `sourceanalyzer` command string:

```
-j <number_worker_processes>
```

The ideal number of worker processes is $n-2$, where n represents the number of processors in your machine. For example, if your machine has 8 processors, the ideal number of worker processes is 6. There is a single master process that coordinates tasks and the distribution of data to the data workers. Each Java process uses the same amount of memory (unless you overrode it with the `com.fortify.sca.RmiWorkerMaxHeap` in the `fortify-sca.properties` file). You might need to balance the `-Xmx` and `-j` options to make sure that you do not allocate more memory than is physically available.

The following is an example of the Fortify Static Code Analyzer command to analyze the build ID `MyServlet`:

```
sourceanalyzer -b MyServlet -scan -j 6
```

The minimum value for `-j` is 2, however HPE recommends that you specify 3 or higher. A value of 3 is usually faster than when not running in parallel analysis mode, but 4 or more provides the best overall speed increase.

Strategies for Best Use

You might want to experiment with the parallel analysis mode settings to achieve a good balance between the available memory and scan speed. HPE makes the following recommendations for getting the best use from these features:

Small to medium projects (< 500K LOC)

- Scan in default mode
- Allow Fortify Static Code Analyzer to spin up threads on all cores
- Set Fortify Static Code Analyzer Java heap size to all of physical memory minus about 1.5 GB for the operating system

Medium to large projects (> 500K LOC)

- Scan in parallel analysis mode
- Allow multithreaded phases to use all cores
- Use 5 to 10 workers for parallel analysis mode
- Set master heap size to double worker heap size
- Use care to make sure that you do not over-commit physical memory

Appendix B: Filtering the Analysis

This section contains the following topics:

Filter Files	100
Filter File Example	100

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: HPE recommends that you only use this feature if you are an advanced user, and that you do not use this feature during 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 sample filter file does the following:

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

The sample filter file contains the following text:

```
#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`.

Execute the following command to use the `-filter` option and the `test_filter.txt` file:

```
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 C: Scan Wizard

This section contains the following topics:

Preparing to use the Scan Wizard	103
Starting the Scan Wizard	104

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.

HPE Security Fortify SCA Scan Wizard

Translation and Scan

Specify how to run your scan and how to handle the output.

Windows Script: Select this check box to generate a script formatted for Windows. In the **Script file location** box, enter the directory in which to store the script.

Unix Script: Select this check box to generate a script formatted for Unix or a Unix-like operating system. In the **Script file location** box, enter the directory in which to store the script.

Scan phase: Select this check box to initiate an SCA scan of the translated data.

Scan result name: To use a file name other than the default shown in the **Scan result name** box, type a name for the FPR file produced by the scan. Make sure that you include the .fpr file extension in the file name.

Quick Scan: Select this check box to scan the project in Quick Scan mode.

Upload scan to SSC: Select this check box to upload the FPR file to HPE Security Fortify Software Security Center.

Scan in cloud: Select this check box to use HPE Security Fortify CloudScan to execute the scan phase in the cloud.

Include custom rules: Select this check box to include custom rules in your scan.

Scan memory: Specify the amount of memory SCA uses for scanning.

Windows Script

☒ Generate Script for Windows

Script file location: C:\Program Files\HP_Fortify\HP_Fortify_SCA_and_Apps_16.10\Samples\ba Browse

Unix Script

☐ Generate Script for Unix

Script file location: C:\Program Files\HP_Fortify\HP_Fortify_SCA_and_Apps_16.10\Samples\ba Browse

☒ Scan phase (in addition to translation phase)

Scan Options

Scan result name: Fortifyeightball.fpr

☐ Quick Scan

☐ Upload scan to SSC

☐ Scan in cloud

☐ Include custom rules

Scan memory: 7280 / 8089 Mb

< Back Next > Finish Cancel

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 login 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 login credentials.

If you do not have Fortify Software Security Center login 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 a Windows system, select **Start > All Programs > HPE Security Fortify SCA and Applications <version> > Scan Wizard**.
- On a Linux system, navigate to the `<sca_install_dir>/bin` directory, and then run the `ScanWizard` file from the command line.
- On a Macintosh system, 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 (HPE_Security_Fortify_Scan_Wizard_<version>_<OS_platform>.zip) for your operating system.
2. Navigate to the HPE-Security-Fortify-<version>-Scan-Wizard/bin directory.
3. Do one of the following:
 - On a Windows system, double-click the ScanWizard.cmd file.
 - On a Linux system, run the ScanWizard file.
 - On a Macintosh system, double-click the ScanWizard file.

Appendix D: Sample Files

The Fortify SCA and Applications installation includes a number of sample files that you can use to test or learn to use Fortify Static Code Analyzer. The sample files 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 samples. The advanced subdirectory includes more advanced samples and code samples that enable you to integrate Fortify Static Code Analyzer with your bug tracking system.

This section contains the following topics:

Basic Samples	106
Advanced Samples	107

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
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

Folder Name	Description	Vulnerabilities
javascript	The <code>sample.js</code> JavaScript file.	Cross Site Scripting (XSS) Open Redirect
nullpointer	The <code>NullPointerExceptionSample.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 are revealed.	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 `<scq_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< <i>bugtracker</i> >	Includes source code for the supported bug tracking plugin.
c++	A sample solution for different supported versions of Visual Studio. To use this sample, you must have the following installed: <ul style="list-style-type: none">• A supported version of Visual Studio Visual C/C++• Fortify Static Code Analyzer and the package for your Visual Studio version

Folder Name	Description
	The code includes a Command Injection issue and an Unchecked Return Value issue.
configuration	A sample Java EE application that has vulnerabilities in its web module deployment descriptor <code>web.xml</code> .
crosstier	<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 SELECT 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. After successful completion of the scan, you should see the SQL Injection vulnerabilities and one 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.
findbugs	A sample that demonstrates how to run the FindBugs static analysis tool (http://findbugs.sourceforge.net) 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 and a J2EE Bad Practices 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 accompanying Java Annotations <code>Sample.txt</code> file describes the potential problems and solutions associated with vulnerability results.</p>
JavaDoc	JavaDoc directory for the <code>public-api</code> and <code>WSCClient</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.

Folder Name	Description
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 E: Issue Tuning

This appendix lists properties that impact the number of issues that Fortify Static Code Analyzer reports. The default settings are designed to provide optimal results. Do not alter them unless you experience an issue or HPE Security Fortify Technical Support has instructed you to do so.

Issue tuning enables you to fine tune the number and quality of results from a Fortify Static Code Analyzer scan. This is an advanced topic and is not necessary for the majority of users.

If you determine that Fortify Static Code Analyzer reports too many or too few issues of a particular type, you might need to adjust a property to exclude or include additional issues. Before making any changes, you might want to contact HPE Security Fortify Technical Support to discuss the issue you are experiencing.

You can tune the following areas:

Wrapper Detection	110
Interprocedural Constant Propagation	111
Selective Map Operation Tracking	111

If you need to turn one or more of these analysis features off, edit the `fortify-sca.properties` file, located in the `<sca_install_dir>/Core/config` directory.

Wrapper Detection

Wrapper detection identifies methods that wrap map operations. In Fortify Static Code Analyzer, map operations insert `<key, value>` pairs to, or retrieve `<key, value>` pairs from, an associative map. When a tainted value is inserted or retrieved, its taint might get propagated through the map.

The HPE Security Fortify Software Security Research team (SSR) provides rules that describe how various APIs implement map insertion and retrieval. Taint propagation occurs when methods that match those specified in the rules are invoked. If Fortify Static Code Analyzer cannot compute the map keys used at those methods, then it promotes taint from a single value to all values in the map. This introduces false positives.

A function is treated as a wrapper method when it:

- Contains a callsite to a method that an SSR rule identified as a map operation or Fortify Static Code Analyzer already identified as a wrapper
- Directly passes its parameters to the map operation
- Directly passes the map operation return value to its own return

The effects of successful wrapper identification include:

- Reduction of false issue reports from the Dataflow Analyzer by reducing the number of issues reported with mismatched map insertions and retrievals

- Improved readability of dataflow issue reports by replacing unknown map keys, shown as '?', with explicit key values

The following property controls the behavior of wrapper detection:

`com.fortify.sca.EnableWrapperDetection`. If you set this property to true, it enables wrapper detection.

Interprocedural Constant Propagation

Programming languages provide keywords to indicate that a variable is a constant, unchanging value throughout an entire program. However, some software fails to consistently apply these keywords to constant variables. Interprocedural Constant Propagation identifies explicit constants and variables that are not defined as constants but the values do not change. It then propagates those constant values throughout all functions in the program.

The properties listed in the following table control Interprocedural Constant Propagation.

Property Name	Description
<code>com.fortify.sca.EnableInterproceduralConstantResolution</code>	If set to true, enables propagation of constant values across function boundaries.
<code>com.fortify.sca.DisableInferredConstants</code>	If set to true, disables identification of constant variables without explicit <code>const</code> or <code>final</code> keywords.

Selective Map Operation Tracking

Selective Map Operation Tracking analysis reduces the prevalence of unresolved map keys. This analysis enables Fortify Static Code Analyzer to find true positives in global classes without introducing an increase in the number of false positives. You configure this algorithm with a property key that accepts four values. The default value, `classrule`, is appropriate in most situations. If you find that too many issues are suppressed, you can change the value and compare the results received.

The properties listed in the following table control Selective Map Operation Tracking.

Property Name	Value	Description
<code>com.fortify.sca.RequireMapKeys</code>	<code>classrule</code>	This is the default value of the property. Fortify Static Code Analyzer analyzes dataflow operations on maps global by <code>classrule</code> only when it can determine keys.
	<code>never</code>	Set this property to <code>never</code> to disable Selective Map Operation Tracking analysis. All map operations are analyzed.

Property Name	Value	Description
	globals	Set this property to <code>globals</code> to increase the aggressiveness of the analysis. Fortify Static Code Analyzer analyzes dataflow operations on all global maps only when it can determine keys.
	always	Set this property to <code>always</code> for maximum aggressiveness. Fortify Static Code Analyzer processes dataflow operations on all maps only when it can determine keys.

Appendix F: Configuration Options

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

This section contains the following topics:

Fortify Static Code Analyzer Properties Files	113
fortify-sca.properties	114
fortify-sca-quickscan.properties	135

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. HPE 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 *HPE Security Fortify Static Code Analyzer Tools Properties Reference Guide*.

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

Properties File Format

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

```
com.fortify.sca.SqlLanguage = TSQL
```

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

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

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

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

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

Precedence of Setting Properties

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

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

Order	Property Specification	Description
1	Command line with the -D option	Properties specified on the command line have the highest priority and you can specify them in any scan.
2	Fortify Static Code Analyzer Quick Scan configuration file	Properties specified in the Quick Scan configuration file (<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 if you want 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 135](#) 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.AbortedScanOverwritesOutput</code>	By default, if a scan is interrupted, Fortify Static Code Analyzer writes the partial results to a different output file: <code><output>.partial.fpr</code> instead of <code><output>.fpr</code> . If this property is set to true, Fortify Static Code Analyzer writes the partial results to the normal output file (<code><output>.fpr</code>). This overwrites any full scan results that might exist in that file.

Property Name	Description
	<p>Value type: Boolean</p> <p>Default: false</p>
com.fortify.sca.Appserver	<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>
com.fortify.sca.AppserverHome	<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>
com.fortify.sca.AppserverVersion	<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>
com.fortify.sca.ASPVirtualRoots. <virtual_path>	<p>Specifies a semicolon delimited list of full paths to virtual roots used.</p> <p>Value type: String</p> <p>Default: (none)</p> <p>Example: com.fortify.sca.ASPVirtualRoots.Library=c:\\WebServer\\CustomerTwo\\Stuff com.fortify.sca.ASPVirtualRoots.Include=c:\\WebServer\\CustomerOne\\inc</p>
com.fortify.sca.NoDefaultRules	<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: -no-default-rules</p>

Property Name	Description
<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.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.NoDefaultSourceRules</code>	<p>If set to true, disables source rules in the default Rulepacks. This can be helpful when creating custom source rules.</p> <p>Note: Characterization source rules are not disabled.</p> <p>Value type: Boolean</p> <p>Default: (none)</p> <p>Command-line Option: <code>-no-default-source-rules</code></p>
<code>com.fortify.sca.ProjectRoot</code>	<p>Specifies the folder to store intermediate files generated in the translation and scan phases.</p> <p>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>

Property Name	Description
	Example: <code>com.fortify.sca.ProjectRoot=C:\Users\<user>\AppData\Local\</code>
<code>com.fortify.sca.DefaultAnalyzers</code>	<p>Specifies a comma- or colon-separated list of the types of analysis to perform. 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>Default: This property is commented out and all analysis types are used in scans.</p>
<code>com.fortify.sca.EnableAnalyzer</code>	<p>Specifies a comma- or colon-separated list of analyzers you want to use for a scan in addition to the default analyzers. 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>Default: (none)</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: <code>\${com.fortify.Core}/config/customrules</code></p>
<code>com.fortify.sca.ExitCodeLevel</code>	<p>Extends the default exit code options. See "Exit Codes" on page 93 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>Command-line Option: <code>-exit-code-level</code></p>
<code>com.fortify.sca.fileextensions.java</code> <code>com.fortify.sca.fileextensions.cs</code> <code>com.fortify.sca.fileextensions.js</code>	<p>Specifies how to translate specific file extensions for languages that do not require build integration. The valid types are: <code>ABAP</code>, <code>ACTIONSCRIPT</code>, <code>ASPNET</code>, <code>ARCHIVE</code>, <code>ASP</code>, <code>ASPX</code>, <code>BITCODE</code>, <code>BYTECODE</code>, <code>CFML</code>, <code>COBOL</code>, <code>CSHARP</code>, <code>HTML</code>, <code>JAVA</code>, <code>JAVA_PROPERTIES</code>, <code>JAVASCRIPT</code>, <code>JSP</code>, <code>JSPX</code>, <code>MSIL</code>, <code>MXML</code>, <code>PHP</code>, <code>PLSQL</code>, <code>PYTHON</code>, <code>RUBY</code>, <code>RUBY_ERB</code>, <code>SWIFT</code>, <code>TLD</code>, <code>SQL</code>, <code>TSQL</code>, <code>VB</code>, <code>VB6</code>, <code>VBSCRIPT</code>, and <code>XML</code>.</p>

Property Name	Description
<code>com.fortify.sca.fileextensions.py</code> <code>com.fortify.sca.fileextensions.rb</code> <code>com.fortify.sca.fileextensions.aspx</code> <code>com.fortify.sca.fileextensions.php</code>	<p>Value type: String (valid language type)</p> <p>Default: See the <code>fortify-sca.properties</code> 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>Note: This is a partial list. For the complete list, see the properties file.</p> <p>You can also specify a value of <code>oracle:<PathToScript></code> 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: <code>com.fortify.sca.fileextensions.jsp=oracle:<PathToScript></code></p>
<code>com.fortify.sca.jsp.UseSecurityManager</code>	<p>If set to true, the JSP parser uses JSP security manager.</p> <p>Value type: Boolean</p> <p>Default: true</p>
<code>com.fortify.sca.jsp.DefaultEncoding</code>	<p>Specifies the encoding for JSPs.</p> <p>Value type: String (encoding)</p> <p>Default: ISO-8859-1</p>
<code>com.fortify.sca.SqlLanguage</code>	<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: <code>-sql-language</code></p>

Property Name	Description
<code>com.fortify.sca.compilers.javac=</code> <code>com.fortify.sca.util.compilers.JavacCompiler</code> <code>com.fortify.sca.compilers.c++=</code> <code>com.fortify.sca.util.compilers.GppCompiler</code> <code>com.fortify.sca.compilers.make=</code> <code>com.fortify.sca.util.compilers.TouchlessCompiler</code> <code>com.fortify.sca.compilers.mvn=</code> <code>com.fortify.sca.util.compilers.MavenAdapter</code>	<p>Specifies custom-named compilers.</p> <p>Value type: String (compiler)</p> <p>Default: See the Compilers section in the <code>fortify-sca.properties</code> 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> <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: <pre>com.fortify.sca.compilers.g++= com.fortify.sca.util.compilers.GppCompiler</pre> <p>Note: This is a partial list. For the complete list, see the properties file.</p>
<code>com.fortify.sca.EnableDOMModeling</code>	<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 JavaScript-based code or any HTML files that include JavaScript. Note that enabling this property can increase the translation and scan time.</p> <p>Value type: Boolean</p> <p>Default: false</p>
<code>com.fortify.sca.DOMModeling.tags</code>	<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: A limited set of DOM element tags such as <code>html</code>, <code>head</code>, <code>input</code>, <code>button</code>, and <code>iframe</code>.</p> <p>Example: <code>com.fortify.sca.DOMModeling.tags=div,p</code></p>

Property Name	Description
<code>com.fortify.sca.JavaScript.src.domain.whitelist</code>	<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: <code>com.fortify.sca.JavaScript.src.domain.whitelist=http://www.xyz.com http://www.123.org</code></p>
<code>com.fortify.sca.DisableJavascriptExtraction</code>	<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>
<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 on which to run phase 0 higher order analysis.</p> <p>Value type: String (comma separated list of languages)</p> <p>Default: <code>python,ruby,swift</code></p>
<code>com.fortify.sca.TypeInferenceLanguages</code>	<p>Comma- or colon-separated list of languages using type inference.</p> <p>Value type: String</p> <p>Default: <code>javascript,python,ruby</code></p>
<code>com.fortify.sca.TypeInferencePhase0Timeout</code>	<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>
<code>com.fortify.sca.TypeInferenceFunctionTimeout</code>	<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>
<code>com.fortify.sca.</code>	<p>If set to true, disables function pointers during the scan.</p>

Property Name	Description
DisableFunctionPointers	<p>Value type: Boolean</p> <p>Default: false</p>
com.fortify.sca. EnableInterprocedural ConstantResolution	<p>If set to true, enables propagation of constant values across function boundaries.</p> <p>Value type: Boolean</p> <p>Default: true</p>
com.fortify.sca. EnableWrapperDetection	<p>If set to false, disables all wrapper detection</p> <p>Value type: Boolean</p> <p>Default: true</p>
com.fortify.sca. RequireMapKeys	<p>This property enables Fortify Static Code Analyzer to compromise a small number of true positives in exchange for a tremendous reduction of false positives. See "Selective Map Operation Tracking" on page 111 for more information. The valid values are <code>classrule</code>, <code>never</code>, <code>globals</code>, and <code>always</code>.</p> <p>Value type: String</p> <p>Default: <code>classrule</code></p>
com.fortify.sca. DisableDeadCode Elimination	<p>Dead code is code that can never be executed, such as code inside the body of an <code>if</code> 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: false</p>
com.fortify.sca. DeadCodeFilter	<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: true</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: <code>-fvd1-no-description</code></p>

Property Name	Description
<code>com.fortify.sca.FVDLDisableProgramData</code>	<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>
<code>com.fortify.sca.FVDLDisableEngineData</code>	<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>
<code>com.fortify.sca.FVDLDisableSnippets</code>	<p>If set to true, excludes code snippets from the analysis results file (FVDL).</p> <p>Value type: Boolean</p> <p>Default: false</p> <p>Command-line Option: -fvd1-no-snippets</p>
<code>com.fortify.sca.FVDLDisableLabelEvidence</code>	<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>
<code>com.fortify.sca.FVDLStylesheet</code>	<p>Specifies location of the style sheet for the analysis results.</p> <p>Value type: String (path)</p> <p>Default: <code>\${com.fortify.Core}/resources/sca/fvd12html.xsl</code></p>
<code>com.fortify.sca.LogFile</code>	<p>Specifies the default log file location.</p> <p>Value type: String (path)</p> <p>Default: <code>\${com.fortify.sca.ProjectRoot}/log/sca.log</code></p> <p>Command-line Option: -logfile</p>
<code>com.fortify.sca.LogMaxSize</code>	<p>When this property is set, it enables log rotation for the Fortify Static Code Analyzer log. The value is the number bytes that Fortify Static Code Analyzer can write to the log file before it is rotated. You must use this property with <code>com.fortify.sca.LogMaxFiles</code>.</p>

Property Name	Description
	<p>Value type: Number</p> <p>Default: (none)</p>
com.fortify.sca. LogMaxFiles	<p>The number of log files to include in the log file rotation set. When all files are filled, the first file in the rotation is overwritten. The value must be at least 1. You must use this property with com.fortify.sca.LogMaxSize.</p> <p>Value type: Number</p> <p>Default: (none)</p>
com.fortify.sca. ClobberLogFile	<p>If set to true, Fortify Static Code Analyzer overwrites the log file for each new scan.</p> <p>Value type: Boolean</p> <p>Default: false</p> <p>Command-line Option: -clobber-log</p>
com.fortify.sca. SuppressLogPrefix	<p>If set to true, no prefix is added to the log file name.</p> <p>Value type: Boolean</p> <p>Default: true</p>
com.fortify.sca. PrintPerformanceDataAfter Scan	<p>If set to true, Fortify Static Code Analyzer writes performance-related data to the log file after the scan is complete. This value is automatically set to true when in debug mode.</p> <p>Value type: Boolean</p> <p>Default: false</p>
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>

Property Name	Description
	Default:
<code>com.fortify.sca.cpfe.441.command</code>	<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: <code>\${com.fortify.Core}/private-bin/sca/cpfe441.rfct</code></p>
<code>com.fortify.sca.cpfe.options</code>	<p>Adds options to the CPFE command line to use when translating C/C++ code.</p> <p>Value type: String</p> <p>Default: <code>--remove_unneeded_entities --suppress_vtbl -tused</code></p>
<code>com.fortify.sca.cpfe.file.option</code>	<p>Sets the name of CPFE option that specifies the output (for example NST) file name.</p> <p>Value type: String</p> <p>Default: <code>--gen_c_file_name</code></p> <p>Example: <code>com.fortify.sca.cpfe.file.option=--gen_c_file_name</code></p>
<code>com.fortify.sca.cpfe.multibyte</code>	<p>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).</p> <p>Value type: Boolean</p> <p>Default: <code>false</code></p>
<code>com.fortify.sca.cpfe.CaptureWarnings</code>	<p>If set to true, any CPFE warnings are included in the Fortify Static Code Analyzer log.</p> <p>Value type: Boolean</p> <p>Default: <code>false</code></p>
<code>com.fortify.sca.cpfe.FailOnError</code>	<p>If set to true, CPFE fails if there is an error.</p> <p>Value type: Boolean</p> <p>Default: <code>false</code></p>
<code>com.fortify.cpfe.IgnoreFileOpen Failures</code>	<p>If set to true, any failure to open a source file (including headers) is considered a warning instead of an error.</p> <p>Value type: Boolean</p>

Property Name	Description
	Default: false
com.fortify.sca. EnableFindbugs	<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: -findbugs</p>
com.fortify.sca. findbugs.maxheap	<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: com.fortify.sca.findbugs.maxheap=500m</p>
com.fortify.sca. CfmlUndefinedVariablesAre Tainted	<p>If set to true, Fortify Static Code Analyzer treats undefined variables in CFML pages as tainted. This serves as a hint to the Dataflow Analyzer to watch out for register-globals-style vulnerabilities. However, enabling this property interferes with dataflow findings where a variable in an included page is initialized to a tainted value in an earlier-occurring included page.</p> <p>Value type: Boolean</p> <p>Default: false</p>
com.fortify.sca. AddImpliedMethods	<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>
com.fortify.sca. JdkVersion	<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>
com.fortify.sca. SuppressLowSeverity	<p>If set to true, Fortify Static Code Analyzer ignores low severity issues found during a scan.</p> <p>Value type: Boolean</p> <p>Default: true</p>
com.fortify.sca. LowSeverityCutoff	<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>

Property Name	Description
	<p>Value type: Number</p> <p>Default: 1.0</p>
com.fortify.sca.DefaultJarsDirs	<p>Specifies the location of commonly used JAR files. The JAR files located in these directories are appended to the end of the class path option (-cp). You can supply a semicolon- or colon-separated list of directories.</p> <p>Value type: String</p> <p>Default: (none)</p>
com.fortify.sca.analyzer.controlflow.liveness.skip.rules	<p>Specifies a (colon-separated) list of rule IDs for the Control Flow Analyzer to skip.</p> <p>Value type: String</p> <p>Default: (none)</p> <p>Example: com.fortify.sca.analyzer.controlflow.liveness.skip.rules=B530C5D6-3C71-48C5-9512-72A7F4911822</p>
com.fortify.sca.analyzer.controlflow.EnableRefRuleOptimization	<p>If set to true, optimizes returning machine data in the Control Flow Analyzer.</p> <p>Value type: Boolean</p> <p>Default: false</p>
com.fortify.sca.analyzer.controlflow.EnableLivenessOptimization	<p>If set to true, enables liveness optimization in Control Flow Analyzer.</p> <p>Value type: Boolean</p> <p>Default: false</p>
com.fortify.sca.analyzer.controlflow.EnableTimeOut	<p>Specifies whether to enable Control Flow Analyzer timeouts.</p> <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=</p>

Property Name	Description
	C:\Windows\System32\reg.exe
WinForms. TransformDataBindings WinForms. TransformMessageLoops WinForms. TransformChange NotificationPattern WinForms. CollectionMutation Monitor.Label WinForms. ExtractEventHandlers	<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. 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> <p>Default: 2</p>
com.fortify.sca. exclude	<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> <div> <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> </div> <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>
com.fortify.sca. FilterFile	<p>Specifies the path to a filter file for the scan. See "Filter Files" on page 100 for more information.</p> <p>Value type: String (path)</p> <p>Default: (none)</p>

Property Name	Description
	Command-line Option: -filter
com.fortify.sca. Debug	<p>If set to true, Fortify Static Code Analyzer includes debug information in the log file.</p> <p>Value type: Boolean</p> <p>Default: false</p> <p>Command-line Option: -debug</p>
com.fortify.sca. DebugVerbose	<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>
com.fortify.sca. DebugTrackMem	<p>If set to true, enables additional debugging for performance information.</p> <p>Value type: Boolean</p> <p>Default: (not enabled)</p>
com.fortify.sca. CollectPerformanceData	<p>If set to true, enables additional timers to track performance.</p> <p>Value type: Boolean</p> <p>Default: (not enabled)</p>
com.fortify.sca. Verbose	<p>If set to true, includes verbose messages in the log file.</p> <p>Value type: Boolean</p> <p>Default: (not enabled)</p> <p>Command-line Option: -verbose</p>
com.fortify.sca. Quiet	<p>If set to true, disables the command-line progress information.</p> <p>Value type: Boolean</p> <p>Default: false</p> <p>Command-line Option: -quiet</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>

Property Name	Description
	Default: (not enabled)
<code>com.fortify.sca.ResultsFile</code>	<p>The file to which results are written.</p> <p>Value type: String</p> <p>Default: (none)</p> <p>Command-line Option: <code>-f</code></p> <p>Example: <code>com.fortify.sca.ResultsFile=results.fpr</code></p>
<code>com.fortify.sca.CaseInsensitiveFiles</code>	<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>
<code>com.fortify.sca.RulesFileExtensions</code>	<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 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.JavaClasspath</code>	<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: <code>-cp</code></p>
<code>com.fortify.sca.JavaExtdirs</code>	<p>Specifies directories to include implicitly on the class path for WebLogic and WebSphere application servers.</p> <p>Value type: String</p>

Property Name	Description
	<p>Default: (none)</p> <p>Command-line Option: -extdirs</p>
com.fortify.sca. JavaSourcepath	<p>Specifies the location of source files 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. 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>
com.fortify.sca. FlexSdkRoot	<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>
com.fortify.sca. FlexSourceRoots	<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>
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>

Property Name	Description
	Command-line Option: -libdir
com.fortify.sca. AbapDebug	If set to true, Fortify Static Code Analyzer adds ABAP statements to debug messages. Value type: String (statement) Default: (none)
com.fortify.sca. AbapIncludes	When Fortify Static Code Analyzer encounters an ABAP 'INCLUDE' directive, it looks in the named directory. Value type: String (path) Default: (none)
com.fortify.sca. BuildID	Specifies the build ID of the build. Value type: String Default: (none) Command-line Option: -b
com.fortify.sca. BuildProject	Specifies a name for the scanned project. Fortify Static Code Analyzer does not use this name but includes it in the results. Value type: String Default: (none) Command-line Option: -build-project
com.fortify.sca. BuildLabel	Specifies a label for the scanned project. Fortify Static Code Analyzer does not use this label but includes it in the results. Value type: String Default: (none) Command-line Option: -build-label
com.fortify.sca. BuildVersion	Specifies a version number for the scanned project. Fortify Static Code Analyzer does not use this version number but it is included in the results. Value type: String Default: (none) Command-line Option: -build-version
com.fortify.sca. MobileBuildSession	If set to true, Fortify Static Code Analyzer copies source files into the build session. Value type: Boolean

Property Name	Description
	Default: false
com.fortify.sca.ExtractMobileInfo	<p>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.</p> <p>Note: Fortify Static Code Analyzer does not extract the mobile build with this property.</p> <p>Value type: Boolean</p> <p>Default: false</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</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.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>

Property Name	Description
	<p>Value type: String</p> <p>Default: auto</p> <p>Command-line Option: -format</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>
com.fortify.sca.ResultsAsAvailable	<p>If set to true, Fortify Static Code Analyzer prints results as they become available. This is helpful if you do not specify the -f option (to specify an output file) and print to stdout.</p> <p>Value type: Boolean</p> <p>Default: false</p>
com.fortify.sca.ProjectTemplate	<p>Specifies the issue template file you want 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: -project-template</p> <p>Example: com.fortify.sca.ProjectTemplate=test_issuetemplate.xml</p>
com.fortify.sca.InputFileEncoding	<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 -encoding option in the translation phase, when Fortify Static Code Analyzer first reads the source code file. Fortify Static Code Analyzer remembers this encoding in the build session and propagates it into the FVDL file.</p> <p>Valid encoding names are from the java.nio.charset.Charset (https://docs.oracle.com/javase/8/docs/api/java/nio/charset/Charset.html)</p>

Property Name	Description
	<p>Typically, if you do not specify the encoding type, Fortify Static Code Analyzer uses <code>file.encoding</code> via the <code>java.io.InputStreamReader</code> constructor with no encoding argument. 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.alias.Enable</code>	<p>If set to true, enables alias analysis.</p> <p>Value type: Boolean</p> <p>Default: <code>true</code></p>
<code>com.fortify.sca.SourceBaseDir</code>	<p>Specifies the base directory for ColdFusion projects.</p> <p>Value type: String (path)</p> <p>Default: (none)</p> <p>Command-Line Option: <code>-source-base-dir</code></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.PHPSourceRoot</code>	<p>Set the PHP source root.</p> <p>Value type: Boolean</p> <p>Default: (none)</p> <p>Command-Line Option: <code>--php-source-root</code></p>
<code>com.fortify.sca.PythonPath</code>	<p>Specifies paths for 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 to specify the additional import directories.</p> <p>Value type: String (path)</p> <p>Default: (none)</p> <p>Command-line Option: <code>-python-path</code></p>
<code>com.fortify.sca.DjangoTemplateDirs</code>	<p>Specifies path to django templates. Fortify Static Code Analyzer does not use the TEMPLATE_DIRS setting from the Django</p>

Property Name	Description
	<p>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. RmiWorkerMaxHeap	<p>Sets the heap size for the workers. The amount of memory required varies from project to project, but you do not have to allocate as much memory for the workers as you do for the master JVM. This property accepts values in kilobytes (K), megabytes (M), or Gigabytes (G).</p> <p>You might need to experiment with this property if you experience low memory warnings, crashes, or do not achieve a speed increase.</p> <p>Value type: String</p> <p>Default: (heap size of master JVM)</p> <p>Example: com.fortify.sca.RmiWorkerMaxHeap=500M</p>
com.fortify.sca. ThreadCount	<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>
com.fortify.sca. UseAntListener	<p>If set to true, Fortify Static Code Analyzer includes com.fortify.dev.ant.SCAListener in the compiler arguments.</p> <p>Value type: Boolean</p> <p>Default: false</p>
com.fortify.sca. DisableASPEXternalEntries	<p>If set to true, disables ASP external entries in the analysis.</p> <p>Value type: Boolean</p> <p>Default: false</p>

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 *HPE Security 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 "com.fortify.sca.ProjectTemplate" on page 133 to specify an issue template that contains the filter set you want 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>HPE Security 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: HPE 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 (HPE Security Fortify Static Code Analyzer 16.20)

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 HPFortifyTechPubs@hpe.com.

We appreciate your feedback!