HP Fortify Static Code Analyzer

Software Version 4.10

User Guide

Document Release Date: April 2014
Software Release Date: April 2014

Legal Notices

Warranty

The only warranties for HP 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. HP 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 HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notice

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:

http://h20230.www2.hp.com/selfsolve/manuals

This site requires that you register for an HP Passport and sign in. To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

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

Part Number: 1-16b3-2014-04-410-01

Preface

Contacting Fortify Software

If you have questions or comments about any part of this guide, contact Fortify Software at:

Technical Support

650.735.2215

fortifytechsupport@hp.com

Corporate Headquarters

Moffett Towers
1140 Enterprise Way
Sunnyvale, CA 94089

650.358.5600

contact@fortify.com

Website

http://www.hpenterprisesecurity.com

About the HP Fortify Software Security Center Documentation Set

The HP Fortify Software Security Center documentation set contains installation, user, and deployment guides for all HP Fortify Software Security Center products and components. It also includes technical notes and release notes that describe new features, known issues, and last-minute updates. The latest versions of these documents are available on the HP Software Product Manuals site:

http://h20230.www2.hp.com/selfsolve/manuals

Change Log

The following table tracks changes made to this guide.

Software Release-version	Date	Change
3.90-01	4/12/2013	Added new introductory chapter and this Change Log.
4.00-01	6/10/2013	Added Maven Integration appendix.
4.00-01	7/8/2013	Added Parallel Analysis Mode appendix and introductory paragraph about feature to the Introduction chapter.
4.00-01	7/15/2013	Added Issue Tuning appendix.
4.10-01	3/22/2014	Updated iOS section.

Contents

Chapter 1: Introduction

This document provides instructions for using HP Fortify Static Code Analyzer.

About the Intended Audience

This guide is intended for people responsible for security audits and secure coding. HP Fortify Static Code Analyzer provides a suite of analyzers and application components.This guide provides instructions on scanning code on most of the major programming platforms.

About the HP Fortify Software Security Center Components

HP Fortify Static Code Analyzer is component of an HP Fortify Software Security Center installation. The installation consists of one or more of the following *analyzers*:

- HP Fortify Static Code Analyzer: Analyzes your build code according to a set of rules specifically tailored to provide the information necessary for the type of analysis performed.
- HP Fortify Runtime Application Protection: Monitors and protects deployed applications from common attacks, unintended use, and targeted hacking. In addition, best security practices, such as input verification and proper exception handling, can be consistently applied to deployed applications.
- HP Fortify SecurityScope: Identifies vulnerabilities in pre-deployment applications during the QA phase, preventing exposure to security flaws before they are exploited.

An HP Fortify Software Security Center installation may also include one or more of the following application tools:

- HP Fortify Audit Workbench: provides a graphical user interface for HP Fortify Static Code Analyzer that helps you organize, investigate, and prioritize analysis results so that security flaws can be fixed quickly.
- HP Fortify Plugin for Eclipse: integrates with the Eclipse development environment and adds the ability to scan and analyze the entire code base of a project and apply hundreds of software security rules that identify the vulnerabilities in your Java code. The results are displayed within the IDE, along with descriptions of each of the security issues and suggestions for their elimination.
- HP Fortify Eclipse Remediation Plug-in: integrates with the Eclipse development environment. The Eclipse Remediation Plug-in is a lightweight plug-in option for developers who need remediation functionality but do not need the scanning and auditing capabilities of Audit Workbench or the full Eclipse Plugin.
- HP Fortify Package for Microsoft Visual Studio: integrates with Visual Studio Premium and Visual Studio Professional to locate security vulnerabilities in your solutions and packages and displays the scan results in Visual Studio. The results include a list of issues uncovered, descriptions of the type of vulnerability each issue represents, and suggestions on how to fix them.
- HP Fortify Remediation Package for Visual Studio: integrates with Microsoft Visual Studio Premium and Visual Studio Professional integrated development environments (IDEs). The HP Fortify Remediation Package for Visual Studio is a lightweight plug-in option for developers who need remediation functionality but do not need the scanning and auditing capabilities of Audit Workbench or the full Visual Studio package.
- HP Fortify Extension for JDeveloper: integrates with the JDeveloper integrated development environment (IDE) and adds the ability to scan and analyze the entire code base of a project and apply hundreds of software security rules that identify the vulnerabilities in your code.
- HP Fortify Remediation Plugin for IntelliJ: integrates with the IntelliJ Integrated Development Environment (IDE) and adds the ability to scan and analyze the entire code base of a project and apply hundreds of software security rules that identify the vulnerabilities in your code.

Related Documents

The following documents provide additional information about HP Fortify Static Code Analyzer:

- *HP Fortify Static Code Analyzer User Guide*

  This document provides instructions on using the analyzers to identify vulnerabilities in your code.
- *HP Fortify Static Code Analyzer Utilities User Guide*

  This document provides information on the command-line tools that provide additional management and access to the functions provided by SCA.

Chapter 2: HP Fortify Static Code Analyzer

This chapter covers the following topics:

- About HP Fortify Static Code Analyzer
- About Analyzers
- About the Analysis Process

About HP Fortify Static Code Analyzer

HP Fortify Static Code Analyzer (SCA) is a set of software security analyzers that search for violations of security-specific coding rules and guidelines in a variety of languages. The rich data provided by SCA language technology enables the analyzers to pinpoint and prioritize violations so that fixes can be fast and accurate. The analysis information produced by SCA helps you deliver more secure software, as well as making security code reviews more efficient, consistent, and complete. This is especially advantageous when large code bases are involved. The modular architecture of SCA allows you to quickly upload new third-party and customer-specific security rules.

At the highest level, using SCA involves:

1. Choosing to run SCA as a stand-alone process or integrating SCA as part of the 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 transferring the resulting FPR file to HP Fortify Audit Workbench or HP Fortify Software Security Center for analysis, or directly with the results displayed on screen

Note: For information on transferring results to HP Fortify Audit Workbench and creating customer-specific security rules, see the *HP Fortify Audit Workbench User's Guide*.

About Parallel Analysis

Beginning with version 4.00, SCA supports parallel processing for large projects. If your project scan takes longer than an hour or two to complete, you can dramatically decrease the time necessary to complete the scan by enabling parallel processing. Parallel processing allows you to take advantage of multiple CPUs and cores within a single machine and automatic memory tuning.

For information on enabling parallel analysis for your projects, see Appendix B: Parallel Analysis Mode.

About Analyzers

SCA comprises six distinct analyzers: Dataflow, Control flow, Semantic, Structural, Configuration, and Buffer. 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 may result in security vulnerabilities or are otherwise unsafe.

Rules are organized according to the analyzer that uses them, resulting in rules that are specific to the Dataflow, Control flow, Semantic, Structural, and Configuration analyzers. These rule categories are further divided to reflect the category of the issue or type of information represented by the rule.

The installation process downloads and updates the set of rules used by SCA on your system. HP updates the specific rules contained within the HP Fortify Secure Coding Rulepacks on a regular basis. The Customer Portal offers updated Rulepacks.

The following table lists and describes each SCA analyzer.

Table 1: HP Fortify Static Code Analyzer

Analyzer	Description
Dataflow	The Dataflow Analyzer detects potential vulnerabilities that involve tainted data (user-controlled input) put to potentially dangerous use. The Dataflow Analyzer uses global, inter-procedural 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 being copied into a statically sized buffer, and detects whether a user controlled string is being used to construct SQL query text.
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.
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. A call to any potentially dangerous function can be flagged by the Semantic Analyzer. For example, the Semantic Analyzer detects deprecated functions in Java and unsafe functions in `C/C++`, such as `gets()`.
Structural	The Structural Analyzer detects potentially dangerous flaws in the structure or definition of the program. By understanding the way programs are structured, the Structural Analyzer identifies violations of secure programming practices and techniques that are often difficult to detect through inspection because they encompass a wide scope involving both the declaration and use of variables and functions. For example, the Structural Analyzer detects assignment to member variables in Java servlets, identifies the use of loggers that are not declared static final, and flags instances of dead code that will never be executed because of a predicate that is always false.
Configuration	The Configuration Analyzer searches for mistakes, weaknesses, and policy violations in an application's deployment configuration files. For example, the Configuration Analyzer checks for reasonable timeouts in user sessions in a web application.
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 inter-procedural analysis to determine whether or not there is a condition that causes the buffer to overflow. If all execution paths to a buffer lead to a buffer overflow, SCA reports it as buffer overflow vulnerability and points out the variables that could cause the overflow. If some, but not all, execution paths to a buffer lead to a buffer overflow and the value of the variable causing the buffer overflow is tainted (user-controlled), then SCA will report it as well and display the data flow trace to show how the variable is tainted.

About the Analysis Process

There are four distinct stages that make up the SCA source code analysis process:

- **Build Integration:** The first stage in the process involves deciding whether to integrate SCA into the build compiler system.

- **Translation:** Next, source code is gathered using a series of commands and then it is translated into an intermediate format associated with a build ID. The build ID is usually the name of the project being scanned.
- **Analysis:** Source files identified during the translation phase are scanned and an analysis results file, typically in the HP Fortify project (FPR) format, is generated. FPR files are indicated by the `.fpr` file extension.
- **Verification of the translation and analysis**: Ensure that the source files were scanned using the correct Rulepacks and that no significant errors were reported.

About Analysis Commands

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

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

To analyze more than one build at a time, add the additional builds as parameters:

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

About Memory Considerations

When running SCA, the amount of physical RAM required is dependent on a number of factors. These factors, which include the size and complexity of the source file, make it impossible to quantify and provide guidance -- each customer situation is unique. If you do encounter a low memory error, increasing the amount of memory available to SCA may resolve the problem.

By default, SCA uses up to 600 MB of memory. If this is not sufficient to analyze a particular code base, you might have to provide more memory in the scan phase. This can be done by passing the `-Xmx` option to the `sourceanalyzer` command.

For example, to make 1000 MB available to SCA, include the option `-Xmx1000M`.

You can also use the `SCA_VM_OPTS` environment variable to set the memory allocation.

Note: Do not allocate more memory for SCA than the machine has available, because this will degrade performance. As a guideline, assuming that no other memory-intensive processes are running, do not allocate more than 2/3 of the available physical memory.

About the Translation Phase

The basic command line syntax for performing the first analysis phase, translating the files, is:

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

The translation phase consists of one or more invocations of SCA using the `sourceanalyzer` command. A build ID (`-b <build_id>`) is used to tie together the invocations.

Subsequent invocations of `sourceanalyzer` add any newly specified source or configuration files to the file list associated with the build ID.

At the end of translation, you can use `-show-build-warnings` to list all warnings and errors that were encountered during the translation process:

```
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

- Translating .NET Source Code
- Translating C/C++ Code
- Translating ABAP/4
- Translating Flex
- Translating Code for Mobile Platforms
- Translating Other Languages

About SCA Mobile Build Sessions

An SCA mobile build session allows a project to be translated on one machine and analyzed on another. When you create an SCA mobile build session, a `.mbs` file that includes the files needed for the analysis phase is created in the build session directory. The `.mbs` file is then moved to a different machine for analysis.

Creating a Mobile Build Session

On the machine where the translation was done, issue the following command to generate an SCA mobile build session:

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

where `<file.mbs>` is the file name you assign for the SCA mobile build session.

Importing a Mobile Build Session

Once you've moved the `.mbs` file to the machine where you want to run the analysis, issue the following command:

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

where `<file.mbs>` is the SCA mobile build session.

Once you have imported your SCA mobile build session, you are ready to move on to the analysis phase.

About the Analysis Phase

This topic describes the syntax for the analysis phase: scanning the intermediate files created during the translation and creating the analysis results file. The phase consists of one invocation of sourceanalyzer. You specify the build ID and include the `-scan` directive and any required analysis or output options.

Note: By default, SCA includes the source code in the FPR.

The basic command line syntax for the analysis phase is:

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

To run an analysis more than one build at a time, add the additional builds to the command line:

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

To run a silent analysis on more than one build at a time, add the additional builds to the command line:

```
sourceanalyzer -b <build-id1> -b <build-id2> -b <build-id3> -auth-silent -scan -f
results.fpr
```

About Verification of the Translation and Analysis Phase

The Result Certification feature of Audit Workbench verifies that the analysis is complete. Result certification shows specific information about the code scanned by SCA, including:

- List of files scanned, with file sizes and timestamps
- Java `CLASSPATH` used for the translation

- List of Rulepacks used for the analysis
- List of SCA runtime settings and command line arguments
- List of errors or warnings encountered during translation or analysis
- Machine/platform information

To view result certification information, open the FPR file in Audit Workbench and select **Tools - Project Summary - Certification**.

About the HP Fortify Scan Wizard

HP Fortify Scan Wizard is a utility that allows you to quickly and easily prepare and scan project code using SCA. The Scan Wizard allows you to run your scans locally, or, if you are using HP Fortify CloudScan, in a cloud of computers provisioned for taking care of the processor-intensive scanning phase of the analysis. For more information, see Appendix G: *HP Fortify Scan Wizard* on page 84.

About HP Fortify CloudScan

With HP Fortify CloudScan (CloudScan), users of HP Fortify Static Code Analyzer can better manage their resources by offloading the processor-intensive scanning phase of the analysis from their build machines to a cloud of machines provisioned for this purpose.

After the translation phase is completed on the build machine, an SCA mobile build session is generated and CloudScan moves it to an available machine for scanning. In addition to freeing up the build machines, this process makes it easy to grow the system by adding more resources to the cloud as needed, without having to interrupt your build process.

In addition, users of Software Security Center can direct CloudScan to output the FPR file directly to the server.

For more information on HP Fortify CloudScan, see the *HP Fortify CloudScan Installation, Configuration, and Usage Guide.*

Chapter 3: Translating Java Code

This chapter covers the following topics:

- About Java Command Line Syntax
- About Java Command Line Examples
- Integrating with Ant using the HP Fortify Ant Compiler Adapter
- Handling Resolution Warnings
- Using FindBugs
- Translating J2EE Applications

About Java Command Line Syntax

The basic command line syntax for Java is:

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

With Java code, SCA can either emulate the compiler, which may be convenient for build integration, or accept source files directly, which is more convenient for command line scans.

Note: For a description of all the options you can use with the `sourceanalyzer` command, see *Command Line Interface* on page 50.

To tell SCA to emulate the compiler, enter:

```
sourceanalyzer -b <build_id> javac [<translation options>]
```

To pass files directly to SCA, enter:

```
sourceanalyzer -b <build_id> -cp <classpath> [<translation options>]
<files>|<file specifiers>
```

where:

```
<translation options>
```

are options passed to the compiler.

```
-cp <classpath>
```

specifies the `CLASSPATH` to be used for the Java source code. A `CLASSPATH` is a list of build directories and jar files. The format is the same as expected by javac (colon or semicolon-separated list of paths). You can use SCA file specifiers.

```
-cp "build/classes:lib/*.jar"
```

Note: If you do not specify the classpath with this option, the `CLASSPATH` environment variable is used.

For more information, see *Java/J2EE Options* on page 53. For information about file specifiers, see *Specifying Files* on page 56.

About Java Command Line Examples

To translate a single file named `MyServlet.java` with `j2ee.jar` on the `CLASSPATH`, enter:

```
sourceanalyzer -b MyServlet -cp lib/j2ee.jar MyServlet.java
```

To translate all `.java` files in the `src` directory using all jar files in the `lib` directory as a `CLASSPATH`:

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

To translate and compile the `MyCode.java` file while using the javac compiler:

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

Integrating with Ant using the HP Fortify Ant Compiler Adapter

SCA provides an Ant Compiler Adapter that you can use as an easy way to translate Java source files if your project uses an Ant build file. This integration requires setting only two Ant properties, and can be done on the command line without modifying the Ant `build.xml` file. When the build runs, SCA intercepts all javac task invocations and translates the Java source files as they are compiled. Note that any JSP files, configuration files, or any other non-Java source files that are part of the application need to be translated in a separate step.

The following steps must be taken to use the Compiler Adapter:

- The `sourceanalyzer` executable must be on the system PATH.
- `sourceanalyzer.jar` (located in `Core/lib`) must be on Ant's classpath.
- The `build.compiler` property must be set to `com.fortify.dev.ant.SCACompiler`.
- The `sourceanalyzer.buildid` property must be set to the build ID.

The following examples show how to run an Ant build using the Compiler Adapter without modifying the build file:

```
ant -Dbuild.compiler=com.fortify.dev.ant.SCACompiler
  -Dsourceanalyzer.buildid=MyBuild
  -lib <install_dir>/Core/lib/sourceanalyzer.jar
```

The `-lib` option is only available in Ant version 1.6 or higher. In older versions you must set the CLASSPATH environment variable or copy `sourceanalyzer.jar` to Ant's lib directory.

Alternatively, with Ant 1.6 or newer, the following shorthand can be used to run Ant with the compiler adapter:

```
sourceanalyzer -b <build-id> ant [ant-options]
```

By default, 600 MB of memory is allocated to SCA for translation. Increase the memory allocation when using the Ant Compiler Adapter using the `-Dsourceanalyzer.maxHeap` option as follows:

```
ant -Dbuild.compiler=com.fortify.dev.ant.SCACompiler
  -Dsourceanalyzer.buildid=MyBuild
  -lib <install_directory>/Core/lib/sourceanalyzer.jar
  -Dsourceanalyzer.maxHeap=1000M
```

Handling Resolution Warnings

To see all warnings that were generated during your build, enter the following command before you start the scan phase:

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

Java Warnings

You may 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 have not been specified. To resolve the warnings, make sure that you have included 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 SCA and the results will be integrated into the analysis results file. Unlike SCA, which runs on Java source files, FindBugs runs on Java bytecode. Therefore, before running an analysis on your project, you should first compile the project and produce the class files.

To demonstrate how to run FindBugs automatically with SCA, compile the sample code, `Warning.java`, as follows:

1. Go to the following directory:
   ```
   <install_directory>/Samples/advanced/findbugs
   ```
2. Enter the following command to compile the sample:
   ```
   mkdir build
   javac -d build Warning.java
   ```
3. Scan the sample with FindBugs and SCA 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 SCA Structural Analyzer produced one issue and FindBugs produced eight. The `Object model violation` issue produced by SCA on line 25 is similar to the `Equal objects must have equal hash codes` issue produced by FindBugs. 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, apply the `filter.txt` filter file by using the `-filter` option during the

scan. Note that the filtering is not complete because each tool filters at a different level of granularity. To demonstrate 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 J2EE Applications

Translating J2EE applications involves processing Java source files and J2EE components such as JSP files, deployment descriptors, and configuration files. While you can process all the pertinent files in a J2EE application using a single-step process, your project may require that you break the procedure into its components for integration in a build process or to meet the needs of various stakeholders within your organization. The following sections provide information on each component, followed by an all-in-one process.

Translating the Java Files

Earlier in this chapter we provided the command line instructions for translating Java files. When translating J2EE applications, use the same procedure for translating the Java files within the application.

For examples, see *About Java Command Line Examples* on page 17.

Translating JSP Projects, Configuration Files, and Deployment Descriptors

In addition to translating the Java files in your J2EE application, you may also need to translate JSP files, configuration files, and deployment descriptors. You can scan JSP files created with version 2.0 and above. 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 this is not the case, you may 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.

J2EE Warnings

You may see the following warnings for J2EE 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 displays because 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, ensure 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. You should also verify that you have all of the `TLD` files for all of the tags that you have and the corresponding `.jar` files with their tag implementations.

Chapter 4: Translating .NET Source Code

The chapter covers the following topics:

- About the Visual Studio Command Prompt
- About Visual Studio .NET
- Translating Simple .NET Applications
- Translating ASP.NET 1.1 (Visual Studio Version 2003) Projects
- Handling Resolution Warnings

This chapter describes how to use SCA to translate Microsoft Visual Studio .NET and ASP.NET applications built with:

- .NET Versions 1.1 and 2.0
- Visual Studio .NET version 2003
- Visual Studio .NET version 2005
- Visual Studio .NET version 2008
- Visual Studio .NET version 2010
- Visual Studio .NET version 2012
- Visual Studio .NET version 2013

SCA works on the Common Intermediate Language (CIL), and therefore supports all of the .NET languages that compile to CIL, including C# and VB .NET.

Note: The easiest way to analyze a .NET application is to use the HP Fortify Package for Microsoft Visual Studio, which automates the process of gathering information about the project.

About the Visual Studio Command Prompt

Visual Studio 2005 and higher include the Visual Studio Command Prompt. The Visual Studio Command Prompt is located in the Visual Studio Tools directory of your Visual Studio installation. You should use this command prompt in the instructions that follow.

About Visual Studio .NET

If you perform command line builds with Visual Studio .NET, you can easily integrate static analysis by wrapping the build command line with an invocation of `sourceanalyzer`. For this to work, you must have the Secure Coding Package 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 by Visual Studio. Be sure to do a clean or a rebuild so that all files are included. You can then perform the analysis phase, as in the following example:

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

Note: If your classic ASP/VBScript application uses *virtual* includes, for example,

```
<!--include virtual="/myweb/foo.inc">
```

then you should specify the physical location of the `myweb` application by passing the following property value:

```
com.fortify.sca.ASPVirtualRoots=<semicoloon separated list of full paths to virtual
roots used>
```

For example, if the IIS virtual root `/myweb` is located at `C:\webapps\myweb-folder`, then your property value should be:

```
-Dcom.fortify.sca.ASPVirtualRoots=c:\webapps\myweb-folder
```

If you add this line to the fortify-sca.properties file, you must escape the `\` character, as in the following:

```
com.fortify.sca.ASPVirtualRoots=c:\\webapps\\myweb-folder
```

Translating Simple .NET Applications

You can also use SCA command line interface for processing .NET applications.

Prepare your application for analysis using one of the following methods:

- Perform a complete rebuild of your project with the "debug" configuration enabled. Compiling your project with debug enabled provides information that SCA uses for presenting the results.
- Obtain all of the third-party `dll` files, project output `dll` files, and corresponding `pdb` files for your projects. Note that SCA ignores any `dll` file passed as an input argument if the corresponding `pdb` file does not exist in the same folder. It is therefore imperative that you include all of the `pdb` files for all your project `dll` files.

  Note: `pdb` files are not required for third-party libraries.

Run SCA to analyze the .NET application from the command line as follows:

- For Visual Studio .NET Version 2010, enter:

  ```
  sourceanalyzer -vsversion 10.0 -b MyBuild
  -libdirs ProjOne/Lib;ProjTwo/Lib ProjOne/bin/Debug ProjTwo/bin/Debug
  ```

  where:

  - `MyBuild` is the build identifier
  - `ProjOne/Lib;ProjTwo/Lib` is a semicolon-separated list of paths to folders or DLLs with third-party DLLs
  - `ProjOne/bin/Debug ProjTwo/bin/Debug` are the output folders
- Use the following version numbers with the `-vsversion` parameter:

Table 2: Visual Studio .NET version numbers

Visual Studio .NET Release	Version Number
Visual Studio .NET 2003	7.1
Visual Studio .NET 2005	8.0
Visual Studio .NET 2008	9.0
Visual Studio .NET 2010	10.0
Visual Studio .NET 2012	11.0
Visual Studio .NET 2013	12.0

Note: Standard .NET DLLs used in your project are automatically picked up by SCA, so you do not need to include them in the command line.

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

```
sourceanalyzer -vsversion <version_number> -b  <build_id>
-libdirs <paths> <folder_1>
...
sourceanalyzer -vsversion <version_number> -b <build_id>
-libdirs <paths> <folder_n>
```

where:

- `<version_number>` is either 7.1, 8.0, 9.0, 10.0, 11.0 or 12.0
- `<build_id>` is the build ID
- `<paths>` is a semicolon-separated list of paths to folders or DLLs with third-party DLLs
- `<folder_1>` and `<folder_n>` are the output folders

Note: SCA requires the appropriate version of Visual Studio unless you are using MSBuild. For information of using SCA with MSBuild, see Appendix E: *MSBuild Integration* on page 74.

Translating ASP.NET 1.1 (Visual Studio Version 2003) Projects

As discussed previously, SCA works on CIL generated by the .NET compilers. For ASP.NET projects, web components such as `aspx` files need to be compiled before they can be analyzed. However, there is no standard compiler for `aspx` files. The .NET 1.1 runtime automatically compiles them when they are accessed from a browser.

To facilitate the `aspx` compilation phase, HP Fortify Software provides a simple tool that compiles all of the `aspx` files in your project. The tool is located in the HP Fortify installation directory at:

```
\Tools\fortify_aspnet_compiler\fortify_aspnet_compiler.exe
```

To analyze ASP.NET 1.1 solutions:

1. Perform a complete rebuild of the solution.
2. For each of the web projects in the solution, delete the following folder:

   ```
   %SYSTEMROOT%\Microsoft.NET\Framework\v1.1.4322\Temporary ASP.NET Files\<web_application_name>
   ```
3. For each of the web projects in the solution, run the following command:

   ```
   fortify_aspnet_compiler <url_to_the_web_site> <source_root_of_the_web_project>
   ```

   where:

   `<url_to_the_web_site>` is the URL for your website, such as `http://localhost/WebApp`

   `<source_root_of_the_web_project>` is the source location of your web project, such as `<VS_project_location>\WebApp`
4. Perform the translation phase for the DLLs built in Step 1. Enter the following command using the same build ID as in the following steps:

   ```
   sourceanalyzer -b <build_id>  "<VS_project_location>\**\*.dll"
   ```
5. Perform the translation phase for the web components. For each of the web projects in the solution, enter the following when you invoke `sourceanalyzer`:

   ```
   sourceanalyzer -b <build_id> %SYSTEMROOT%\Microsoft.NET\Framework\v1.1.4322\Temporary ASP.NET
   Files\<web_application_name>
   ```
6. Include the configuration files and any Microsoft T-SQL source files that you have:

   ```
   sourceanalyzer -b <build_id> "<solution_root>\**\*.config"
   <"t-sql_src>\**\*.sql">
   ```

   Note: These steps are all automated if you use the HP Fortify Package for Microsoft Visual Studio.

Handling Resolution Warnings

To see all warnings that were generated during your build, enter the following command before you start the scan phase:

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

About .NET Warnings

You may see the following warnings for .NET:

```
Cannot locate class... in the given search path and the Microsoft .NET Framework
libraries.
```

These warnings are typically caused by missing resources. For example, some of the `.DLL` files required to build the application have not been specified. To resolve the warnings, make sure that you have included all of the required files that your application uses. If you still see a warning and the classes it lists are empty interfaces with no members, you can ignore the warning. If the interface is not empty, contact Technical Support.

About ASP.NET Warnings

You may see the following warnings for ASP.NET applications:

```
Failed to parse the following aspx files:

<list of .aspx file names>
```

This warning displays because your web application is not deployed correctly or does not contain the full set of required libraries, or it uses the Global Access Cache (GAC). If your application is a .NET version 1.1 application, you may also have access issues from Microsoft IIS. Verify that you can access the application from a browser without authentication or access errors. If your web application uses the GAC, you must add the `DLL` files to the project separately to ensure a successful scan. SCA does not load `DLL` files from the GAC.

Chapter 5: Translating C/C++ Code

This chapter covers the following topics:

- About C and C++ Command Line Syntax
- C and C++ Command Line Examples
- About Integrating with Make
- About the HP Fortify Build Monitor
- About Command Line Builds in Visual Studio .NET
- About Command Line Builds in Visual Studio 6.0

About C and C++ Command Line Syntax

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

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

where:

`<compiler>` is the name of the compiler you want to use during a project build scan, such as `gcc` or `cl`.

`<compiler options>` are options passed to the compiler that are typically used to compile the file.

C and C++ Command Line Examples

The following is a simple usage example:

To translate a file named `helloworld.c` using the gcc compiler, enter:

```
sourceanalyzer -b my_buildid gcc helloworld.c
```

Note: This also compiles the file.

About Integrating with Make

You can use either of the following methods to integrate SCA with Make:

- HP Fortify Touchless Build Adapter
- Modify a Makefile to Invoke SCA

Using the HP Fortify Touchless Build Adapter

The following section describes the different methods for using the touchless build adapter.

Using the sourceanalyzer Build Adapter Command

For example, to use the HP Fortify touchless build adapter to integrate with a python build script:

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

SCA runs the entire command following the "touchless" keyword. When the command creates a new process that SCA determines is a compiler, the command is processed by SCA.

For information about informing SCA about specially named compilers, see the com.fortify.sca.compilers.* property in “Using Properties to Control Runtime Options” on page 68. Any build command that executes a recognized compiler process can be used with this system; just replace the 'make' section of the above command with the command used to run a build.

Note: The HP Fortify touchless build adapter does not function correctly if:

- The build script invokes the compiler with an absolute path or overrides the executable search path.
- The build script does not create a new process to run the compiler. Many Java build tools, including Ant, operate this way.

Modifying a Makefile to Invoke SCA

To modify a makefile to invoke SCA, replace any calls to the compiler, archiver, or linker in the makefile with calls to SCA. These tools are typically specified in a special variable in the makefile, as in the following example:

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

The step can be as simple as prepending these tool references in the makefile with SCA and the appropriate options:

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

About the HP Fortify Build Monitor

This section describes how to use HP Fortify Build Monitor to scan C/C++ projects automatically during a build on Windows and view the results. It includes examples that use sample projects provided with SCA.

The following options are available from the HP Fortify Build Monitor menu:

Table 3: HP Fortify Build Monitor Options

Option	Description
Monitor	Enables the monitoring. Build Monitor intercepts and translate the next build on the machine.
Build Done	Stops the monitor after the build is complete.
Scan	Scans the code that was monitored during the build.
Scan Settings	Controls the Rulepacks and memory settings.
Set Results Folder	Controls where SCA outputs the results.
Stay on Top	Keeps the HP Fortify Build Monitor window on top of other windows.
Minimize to Tray	Shows the HP Fortify Build Monitor as an icon in the task bar.
Exit	Closes the HP Fortify Build Monitor.
Show Messages	Shows or hides the messages in the lower area of the window. Messages include Scan Messages, Errors, and Monitor Driver information. You can click **Detailed Messages** at the bottom of the window.
Help	Displays online help.
Reset	Resets the HP Fortify Build Monitor to its beginning state.

Configuring HP Fortify Build Monitor

This section covers setting up the results folder and setting SCA scan options.

Setting Up the Results Folder

HP Fortify Build Monitor outputs results in FPR format to a local folder. You can change the output folder. HP Fortify Build Monitor replaces the results each time a scan is performed. Results are not archived.

To change the results folder:

1. Select **Action > Set Results Folder**.

   The Browse for Folder dialog displays.
2. Select a folder and click **OK**.

HP Fortify Build Monitor will output the results to the selected folder.

Setting SCA Scan Options

HP Fortify Build Monitor scans the project using SCA. You can adjust the following scan settings:

- **Allocate memory:** Increase or decrease the amount of memory allocated to SCA
- **Secure Coding Rulepacks and custom Rulepacks:** Change which Rulepacks SCA uses to analyze the source code
- **User:** Only monitor builds run by the current user

To change the scan options:

1. Select **Action > Scan Settings**.

   The HP Fortify Build Monitor: Scan Settings dialog displays.
2. To change the memory allocation, select a value.

   Note: Entering an invalid option sets the memory to unlimited.
3. To add or remove Rulepacks, click **Rulepacks**.
4. To view the SCA command line options, click **Preview**.
5. Click **Done**.

The SCA scan options are changed.

Monitoring Builds

For C/C++ projects and solutions on Windows, SCA includes the HP Fortify Build Monitor, which is a graphical user interface tool that automates analysis during builds.

To analyze C/C++ source code builds on Windows:

1. Select **Start > Program Files > HP Fortify Software > HP Fortify v3.80 > HP Fortify SCA - Build Monitor**.
2. Click **Monitor**.

   After the monitor initiates a green light icon displays.
3. Create a complete build of your project in your build environment.
4. Check that the build has finished successfully.
5. Return to the HP Fortify Build Monitor window and click **Build Done**.
6. SCA outputs the results to a subfolder, specify a name for the folder for the output. If the folder already exists, SCA cleans the folder before starting the scan.
7. Click **Scan**.

SCA displays the results and saves an FPR file in the folder you specified.

Note: To view the results, open the FPR file in Audit Workbench or using the Secure Coding Package for Microsoft Visual Studio.

Monitoring a Project Example

This example for Windows users analyzes the sample C++ code project named `qwik-smtpd`. It uses Microsoft Visual Studio and the HP Fortify Build Monitor.

To analyze the `qwik-smtpd` project:

1. Using Microsoft Visual Studio, open and build the `qwik-smtpd` project located in the `Tutorial/C/source` directory.
2. Select **Start > Program Files > HP Fortify Software > HP Fortify v3.80 > HP Fortify SCA > Build Monitor**.
3. Click **Monitor**.
4. Minimize the window.
5. In Microsoft Visual Studio, rebuild the project.

   Note: Since nothing in the project changed, you must use the rebuild option.
6. Check that build has finished successfully.
7. Return to the HP Fortify Build Monitor window and click **Build Done**.
8. Specify the location of the build output.
9. Click **Scan**.

SCA saves an FPR file in the folder you specified.

Note: To view the results, open the FPR file in Audit Workbench or using the Secure Coding Package for Microsoft Visual Studio.

About Command Line Builds in Visual Studio .NET

If you perform command line builds with Visual Studio .NET, you can easily integrate static analysis by simply wrapping the build command line with an invocation of `sourceanalyzer`. For this to work, you must have the HP Fortify Package for Microsoft Visual Studio for your version of Visual Studio installed.

Consider the following example

```
sourceanalyzer -b my_buildid devenv MyProject.sln /REBUILD
```

This performs the translation phase on all files built by Visual Studio. Be sure to do a clean or a rebuild so that all files are included.

About Command Line Builds in Visual Studio 6.0

If you perform command line builds with Visual Studio 6.0, you can integrate static analysis by wrapping the build command line with an invocation of `sourceanalyzer`.

Consider the following example:

```
sourceanalyzer -b my_buildid msdev MyProject.dsp /MAKE "MyProject DEBUG" /REBUILD
```

This performs the translation phase on all files built by the Visual Studio. Be sure to do a clean or a rebuild so that all files are included, as described in your Visual Studio documentation.

Chapter 6: Translating ABAP/4

This chapter covers the following topics:

- About Translating ABAP/4 Code
- About Scanning ABAP Code
- Overview of the Process
- About the Transport Request
- Create a Transaction Object
- Adding Fortify SCA to Your Favorites List (optional)
- Running the HP Fortify ABAP Extractor

About Translating ABAP/4 Code

Translating ABAP/4 code is similar to translating other operating language code, but requires additional steps in order to extract the code from the SAP database and prepare it for scanning. This chapter assumes you have SCA running and have a basic understanding of SCA, SAP, and ABAP/4.

About Scanning ABAP Code

When a PACKAGE is extracted from ABAP, the HP 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 may optionally provide.

Programs are extracted from `TRDIR` by matching the name field against either the program name given by the user in the selection screen or by comparing with 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 finally 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 sourceanalyzer will translate and scan, producing an `.fpr` file which can be viewed with HP Fortify Audit Workbench.

About INCLUDE Processing

As source code is downloaded, the HP 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.

Overview of the Process

There are two main steps required prior to translating your ABAP/4 code:

- Install a Transport Request on your SAP server.

  In this step, you will install the HP Fortify Extractor program.
- Create a transaction object.

In this step you will assign a transaction to the ABAP object created when you installed the Transport Request.

In addition, you may want to add the transaction object to your Favorites list to make HP Fortify SCA easily accessible.

Note: The following procedure is based on the use of the Windows SAP client-based interface. Screen shots and interface locations may vary if you are using a different SAP client.

About the Transport Request

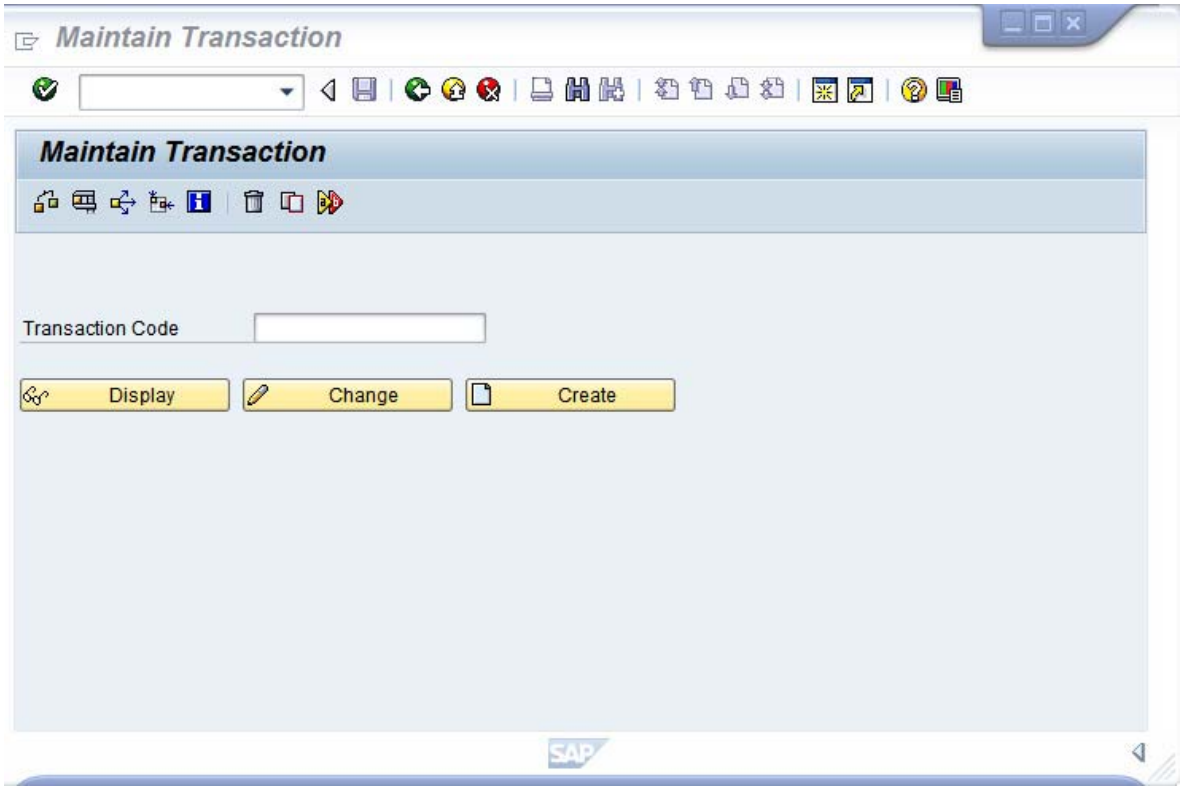
ABAP scanning is available as a premium component of SCA. If you purchased a license that includes this capability, you will need to install the HP Fortify Transport Request on your SAP Server. Contact HP Fortify Support for a copy of the HP Transport Request or information on adding this functionality to your licensed copy of SCA.

Create a Transaction Object

You will need to create a transaction object in order to launch an SCA scan. Follow the steps below to create a transaction object.

1. Press **ESC** until you reach the **SAP Easy Access** screen and type `se93`.

   The **Maintain Transaction** screen appears.

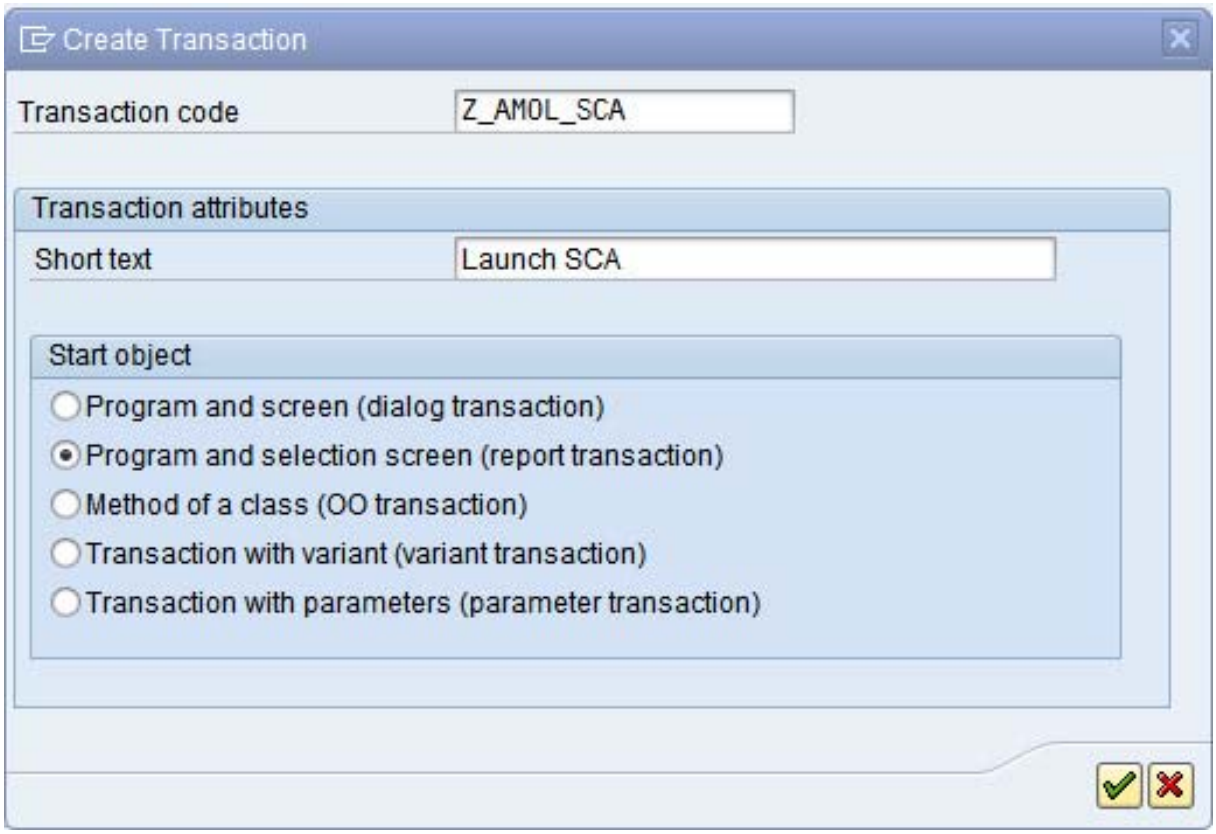


2. Type `Z_AMOL_SCA` in the **Transaction Code** box and click the **Create** button.

   The **Create Transaction** box appears.

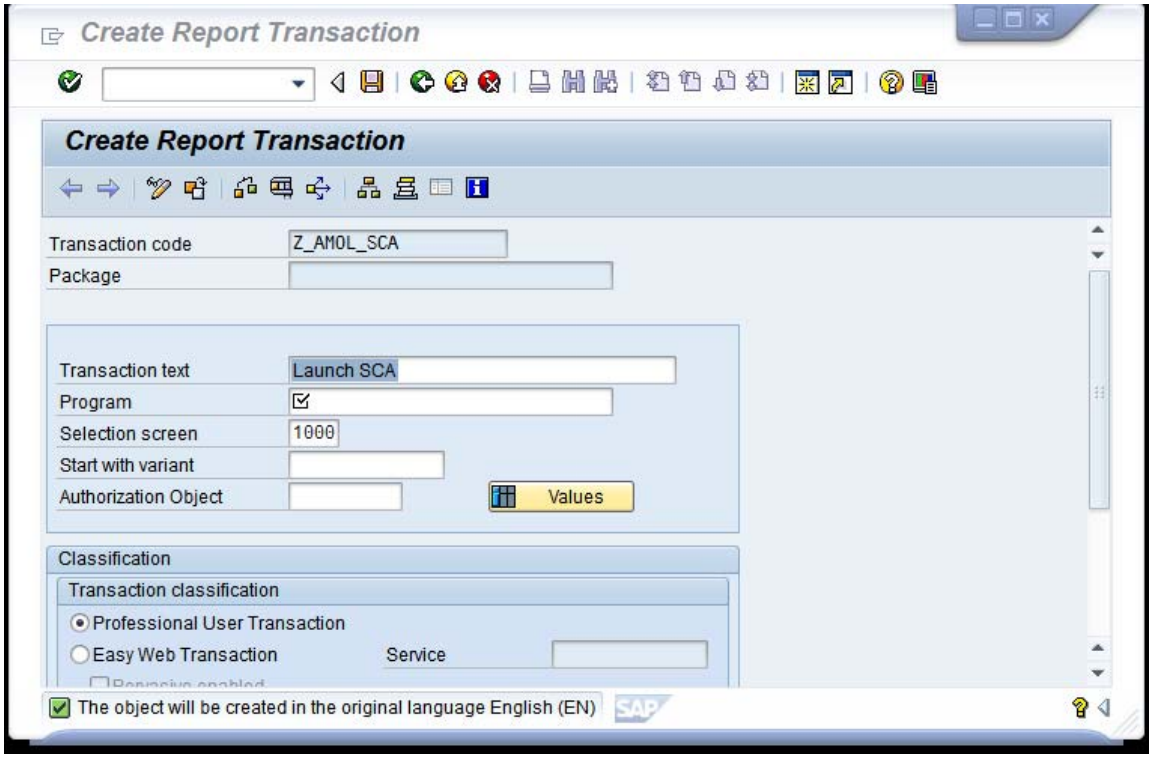
3. In the **Transaction attributes** section, type in a short description of the purpose of the transaction (for example, Launch SCA) in the **Short text** box.

4. In the **Start object** section, select the **Program and selection screen (report transaction)** radio button.



5. Click the green checkmark button  .

The **Create Report Transaction** screen appears.

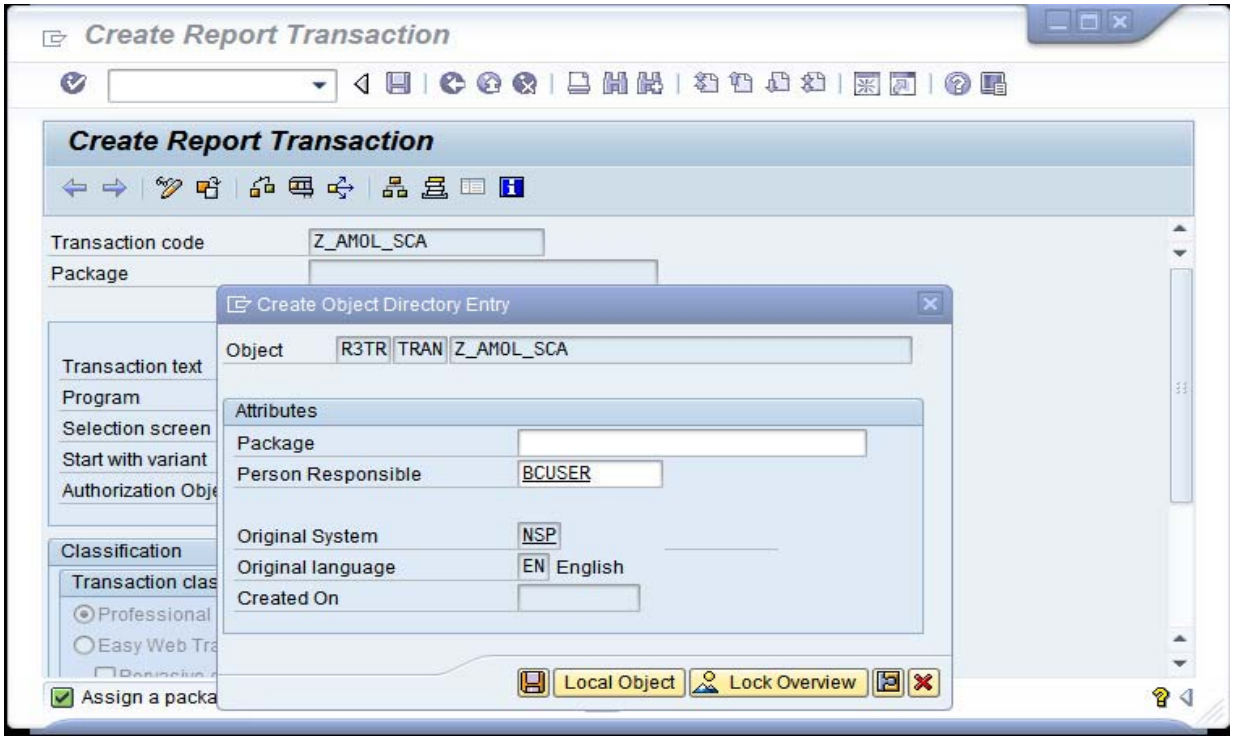


6. In the **Package** text box, type `$TMP`

   Note: If the **Package** box is not editable, leave it empty. On the following screen you should be able to enter the package details.

7. Type the program name (for example, `Z_AMOL_SCA`) into the **Program** box.
8. In the **GUI support** section, select all three check boxes.
9. Click **Save** on the toolbar.

The **Create Object Directory Entry** screen appears.



Note: If you weren't able to enter package details on the previous screen, type `$TMP` in the Package field.

10. Click the **Save** button.

Adding Fortify SCA to Your Favorites List (optional)

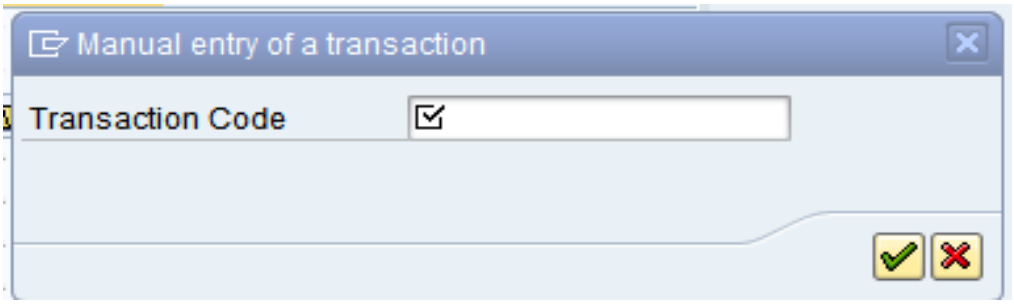
Adding Fortify SCA to your Favorites list is optional, but doing so may make it easier to access and launch Fortify SCA scans. The following steps assume that you use the User menu in your day-to-day work. If your work is done from a different menu, add the Favorites link to the menu that you use. Before you create the Fortify SCA entry, the SAP server should be running and you should be in the SAP Easy Access area of your web-based client.

1. From the **SAP Easy Access** menu, type `SOOO` in the transaction box.

   The **SAP Menu** appears.

2. Right-click the **Favorites** folder and select **Insert transaction**.

   The **Manual entry of a transaction** box appears.

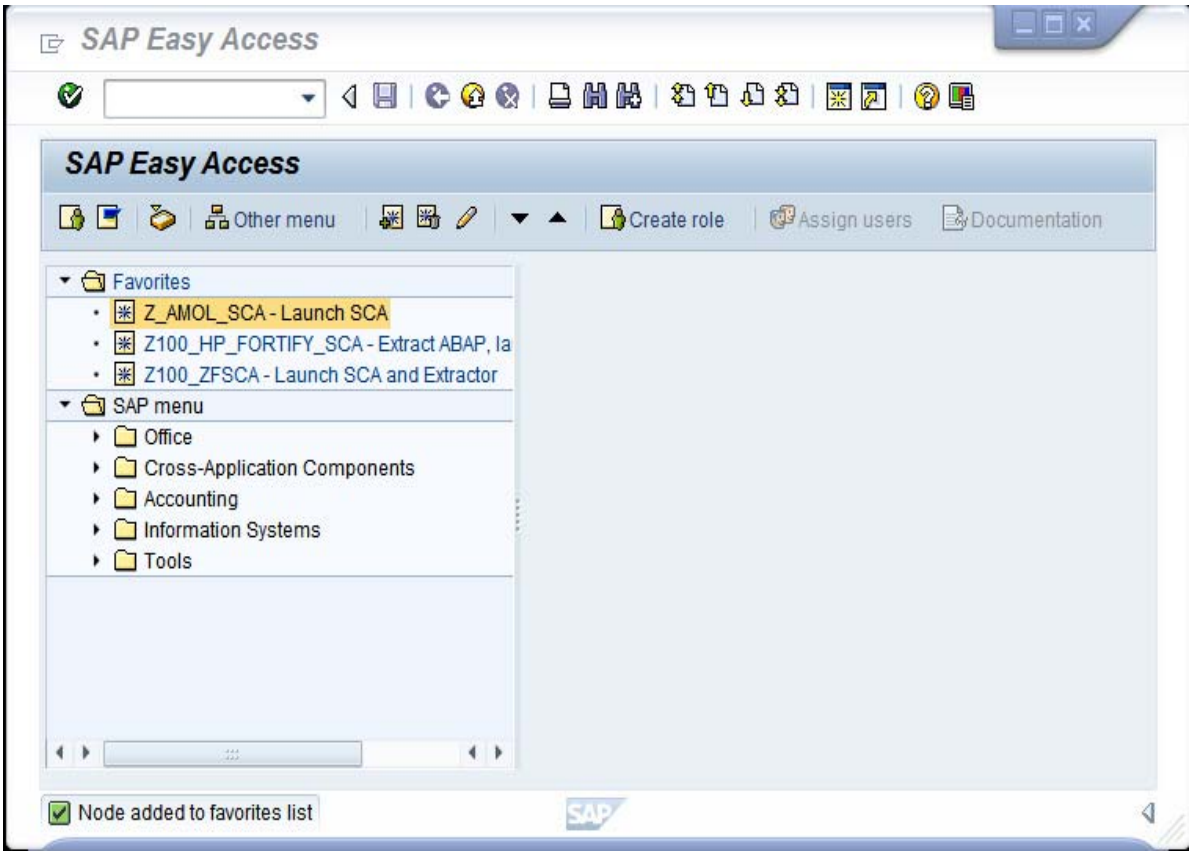


3. Type `Z_AMOL_SCA` in the **Transaction code** box.

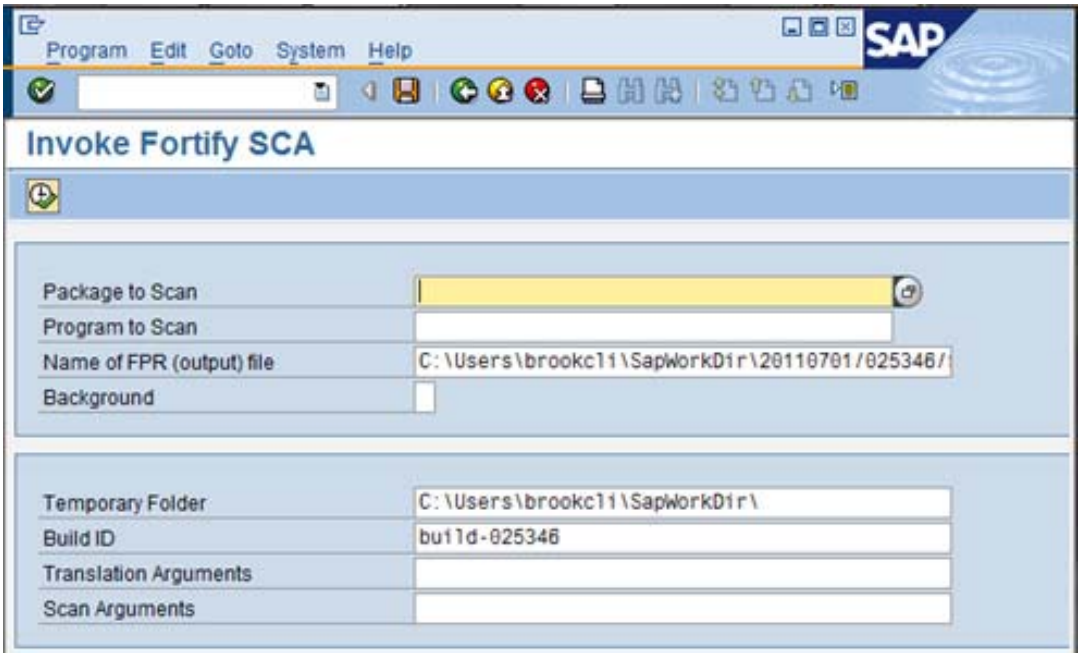
   Note: If you chose a different name when creating your transaction code, use that instead.

4. Click the green checkmark button  .

   The **Launch SCA** item should appear in the **Favorites** list.

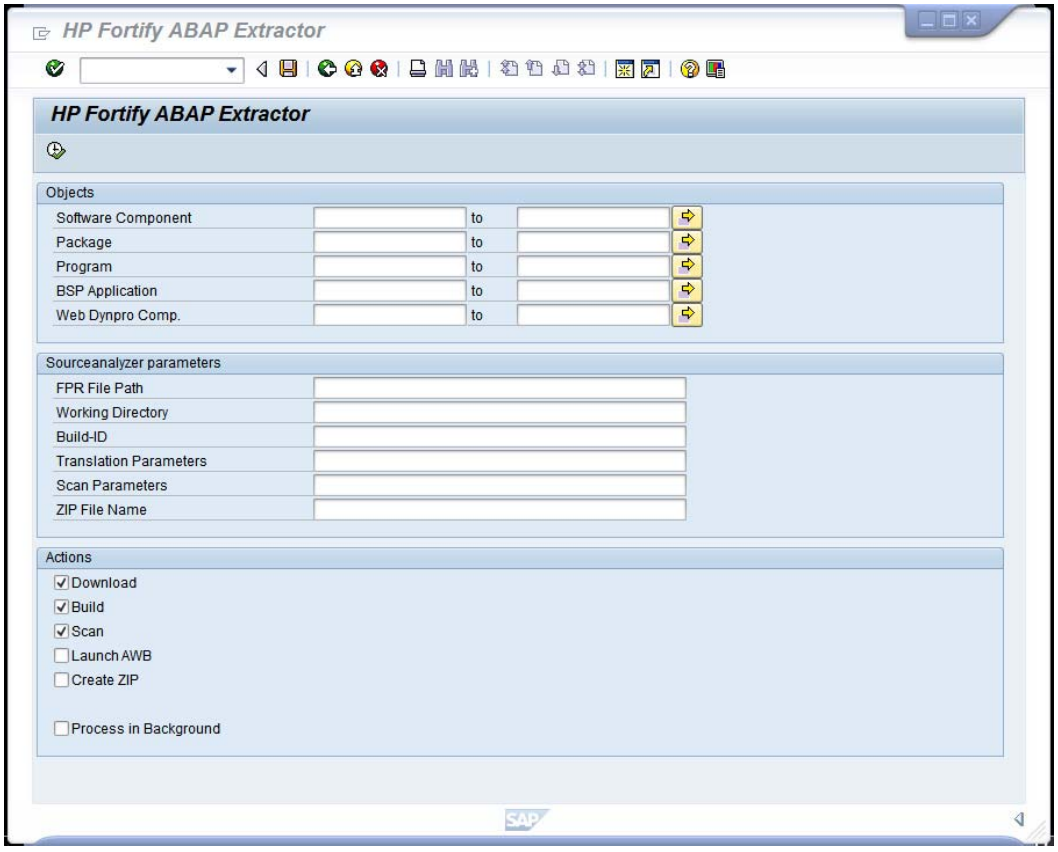


5. Click the link to launch SCA.



Running the HP Fortify ABAP Extractor

1. Launch the program from the **Favorites** link, the transaction code, or by manually launching the `Z_AMOL_SCA` object.



2. Fill in the requested information:

Section	Data
Objects	Enter the name of the **Software Component**, **Package**, **Program**, **BSP** or W**eb Dynpro Component** you want to scan.
Sourceanalyzer parameters	**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. **Working Directory**: Type the directory where the extracted source code should be copied. **Build-ID:** Type the build ID for the scan. **Translations Parameters:** List any optional sourceanalyzer translation arguments. **Scan Parameters**: List any optional sourceanalyzer scan arguments. **ZIP File Name:** Type a ZIP file name if you would like your output provided in a compressed package.
Actions	**Download**: Check this box to instruct SCA to download the source code extracted from your SAP database. **Build**: Check this box to instruct SCA to <> **Scan**: Check this box to request a scan. **Launch AWB**: Check this box to launch Audit Workbench and load the FPR. **Create ZIP**: Check this box to request the output be compressed. **Process in Background:** Check this box to request that processing occur in the background.

3. Click the **Execute** button.

Chapter 7: Translating Flex

This chapter covers the following topics:

- About the Command-Line Options
- About ActionScript Command Line Syntax
- ActionScript Command Line Examples
- About Handling Resolution Warnings

About the Command-Line Options

The following command-line options (with corresponding *properties* that can be used instead, for convenience) are using when translating Flex files:

- **`-flex-sdk-root`** `(com.fortify.sca.FlexSdkRoot)` should point to 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.

  You can set this property in your `fortify-sca.properties` file.

- **`-flex-libraries`** `(com.fortify.sca.FlexLibraries)` contains a `:` or `;` separated list (`:` on most platforms, `;` 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/` under your Flex SDK root).

  You can set this property in your `fortify-sca.properties` file to use the same set of SWCs.

Note: You can specify SWC or SWF files as Flex libraries, but we do not currently support SWZ.

- `-flex-source-roots` (`com.fortify.sca.FlexSourceRoots`) contains a `:` or `;` separated list of root directories in which MXML sources can be found. Normally, these will contain a subfolder named `com`. For instance, if a Flex source root is given that is pointing at `foo/bar/src`, then `foo/bar/src/com/fortify/manager/util/Foo.mxml` will get transformed into an object named `com.fortify.manager.util.Foo`, (an object named Foo in the package `com.fortify.manager.util`).
- `-flex-sdk-root` and `-flex-source-roots` are primarily for MXML translation, and are optional if you are scanning pure ActionScript. `-flex-libraries` is used for resolving all ActionScript

Note: MXML files are translated into ActionScript and then run through the ActionScript parser. The ActionScript that is generated is intended to be simple to analyze; not rigorously correct like the Flex run-time model. As a consequence of this, you may 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, please notify HP Fortify Support.

About ActionScript Command Line Syntax

The basic command line syntax for ActionScript is:

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

To pass files directly to Fortify SCA, enter:

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

where:

`<listOfLibraries>`

is a semicolon-separated list (Windows) or a colon separated-list (non-Windows systems) of library names that you want to "link" to.

ActionScript Command Line Examples

The following examples illustrate command-line structure for typical scenarios you may encounter.

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 your MXML application's being translated 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
```

In this example, we locate the Flex SDK. SCA file specifiers are used to include the `.as` and `mxml` files under the `src` folder. It is not necessary to explicitly specify the `.SWC` files found under the `-flex-sdk-root`, although this example does so for the purposes of illustration. SCA will automatically locate all `.SWC` files under the specified Flex SDK root, and it assumes that these are libraries intended for use translating ActionScript or mxml files.

Example 3

In this example, the Flex SDK root and Flex libraries are specified in a properties file since typing in the data is time consuming and it tends to be constant. The application may be divided into two sections and stored in folders: a main section folder and a modules folder. Each folder contains an `src` folder where the paths should be begun. Wildcards are used in file specifiers to pick up all the `.mxml` and `.as` files in both of the src folders. An `MXML` file in `main/src/com/foo/util/Foo.mxml` will be 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
```

About Handling Resolution Warnings

To see all warnings that were generated during your build, enter the following command before you start the scan phase:

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

About ActionScript Warnings

You may receive a message similar to:

```
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 SCA cannot find all of the libraries it needs. You may need to specify additional SWC or SWF Flex libraries (-flex-libraries option, or com.fortify.sca.FlexLibraries property) so that SCA can complete the analysis.

Chapter 8: Translating Code for Mobile Platforms

This chapter covers the following topics:

- About Translating Objective-C Code
- About Objective-C on iPhone
- About Translating Google Android Code

About Translating Objective-C Code

This section describes how to translate Objective-C source code for iOS applications.

Prerequisites

- Xcode command-line tools must be installed in the path
- Projects must use the non-fragile Objective-C runtime (ABI version 2 or 3)
- Use Apple's *xcode-select* command-line utility to set your Xcode path. SCA uses the systems's global Xcode configuration to find the Xcode toolchain and headers.

About Objective-C Command Line Syntax

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

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

where:

`<compiler options>` are options passed to xcode.

Objective-C Command Line Example

The following simple examples illustrate usage patterns for the supported compilers. The following command samples should be run from the directory where the project files are located.

To translate an Xcode Objective-C project, enter:

```
sourceanalyzer -b my_buildid -clean
sourceanalyzer -b my_buildid xcodebuild -sdk iphonesimulator
```

Note: If you have an Apple Developer Certificate, pass `-sdk iphoneos` instead of `-sdk iphonesimulator`.

To scan the application artifact files:

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

Note: The source code will be compiled when running these commands.

Xcode Compiler Errors

If you receive Xcode compiler errors, this may be due to the inclusion of Clang options added after your version of SCA was released. To eradicate the errors, type the following after xcodebuild:

```
ARCHS=i386
GCC_TREAT_WARNINGS_AS_ERRORS=NO
```

where `ARCHS=i386` represents the architectures (ABIs, processor models) to which the binary is targeted.

About Objective-C on iPhone

When building code for the iPhone, you will need to pass an SDK option based on whether or not you have an Apple Developer Certificate.

If you have an Apple Developer Certificate, pass `-sdk iphoneos`

If you do not have an Apple Developer Certificate, pass `-sdk iphonesimulator`

About Translating Google Android Code

SSR provides rules support for programs that run on the Google Android platform. These rules

- identify insecure data storage
- categorize applications by their security permissions and detect overprivileged uses
- send and receive intents, identify database, file system, web, private information and Android inter-component sources

Translating Google Android code is similar to translating Java code. For instructions on translating Java code, see Chapter 2, *Translating Java Code* on page 17.

Migration Issues

If you have migrated from a previous version of SCA and receive an error when running SCA, it may be due to a deprecated property key in your fortify-sca.properties file. Check the fortify-sca.prorties file (located in the `install directory>/SCA/Core/config/` directory for any of the following, deprecated property keys:

```
com.fortify.sca.xcodebuild.CompilerPath
com.fortify.sca.xcodebuild.SupportedVersion
com.fortify.sca.xcodebuild45.llvmgcc
com.fortify.sca.xcodebuild43.CompilerPath
com.fortify.sca.clang.includes
com.fortify.sca.clang.CaptureWarnings
com.fortify.sca.llvmtonst.CaptureWarnings
com.fortify.sca.llvmtonst.FailOnError
com.fortify.sca.llvmtonst.command
com.fortify.sca.llvmtonst.options
com.fortify.sca.pretranslate.command
```

If found, remove the key from your properties file.

Chapter 9: Translating Other Languages

This chapter covers the following topics:

- About Command Line Syntax for Other Languages
- Configuration Considerations
- Translating COBOL Code

About Command Line Syntax for Other Languages

This topic describes the SCA command syntax for translating other languages.

The basic command line syntax for other languages is:

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

Enter the following to select the `sql` type being translated on Windows platforms:

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

or

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

SQL Note: By default, files with the extension `sql` 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, you can configure SCA to treat them as PL/SQL rather than explicitly specify it each time your run sourceanalyzer

To change the default behavior, set the `com.fortify.sca.fileextensions.sql` property in `fortify-sca.properties to "TSQL" or "PLSQL."`

Enter the following to perform translation on 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

ColdFusion Note: SCA calculates the relative path to each CFML source file by using the `-source-base-dir` directory as the starting point, then uses these relative paths when generating instance IDs. If the entire application source tree is moved to a different directory, the instance IDs generated by a security analysis should remain the same if you specify an appropriate value for `-source-base-dir`.

For a description of all the options you can use with the `sourceanalyzer` command, see *Command Line Interface* on page 50.

File specifiers are shown in the following table:

Table 4: File Specifiers

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

Table 4: File Specifiers

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

Note: Windows and many Unix shells automatically try to expand arguments containing the '`*`' character, so file-specifier expressions should be quoted. Also, on Windows, enter the backslash (\) instead of the forward slash (/).

Configuration Considerations

This section provides information on configuring Python, configuring ColdFusion, configuring the SQL extension, and configuring ASP/VSScript virtual root.

Configuring Python

SCA translates Python applications, and treats files with the extension `py` as Python source code. In order for SCA to translate Python applications and prepare the application for a scan, SCA searches any import files for the application. SCA does not respect the PYTHONPATH environment variable which the Python runtime system uses to find imported files, so this information should be given directly to SCA using the `-python-path` argument. In addition, some applications add additional import directories during runtime initialization.

To add paths for additional import directories, use the sourceanalyzer command line option:

```
-python-path pathname
```

Note: SCA translates Python applications using all import files located in the directory path defined by the `-python-path pathname` option. Subsequently, translation may take a significant amount of time to complete.

Configuring ColdFusion

In order 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
```

Doing so serves as a hint to the data flow analyzer to watch out for register-globals-style vulnerabilities. However, enabling this property interferes with Dataflow findings in which a variable in an included page is initialized to a tainted value in an earlier-occurring included page.

Configuring the SQL Extension

By default, files with the extension `sql` 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, you should configure SCA 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."

Note: Fortify 360 v2.5 updated the PL/SQL parser to improve translation of PL/SQL source code. However, the existence of two different parsers can make merging results from pre-v2.5 and post-v2.5 difficult.

To revert to the older version of the PL/SQL parser, add the following property to the `fortify-sca.properties` file:

```
com.fortify.sca.UseOldPlsql=true
```

Configuring ASP/VBScript Virtual Roots

SCA allows you to handle ASP virtual roots. For web servers that use virtual directories as aliases that map to physical directories, SCA allows you to use alias.

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

As an example, the ASP/VBScript code for an application using *virtual* includes, as follows:

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

The above ASP code refers to the actual directory, as follows:

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

The real directory replaces the virtual directory name `Include` in that instance.

Accommodating Virtual Roots

In order to indicate to SCA what each virtual directory is an alias for, you must set a property of the form `com.fortify.sca.ASPVirtualRoots.name_of_virtual_directory` as part of your commandline invocation of SCA in the following manner:

```
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 include the property setting in double-quotes:

```
sourceanalyzer "-Dcom.fortify.sca.ASPVirtualRoots.name_of_virtual_directory=<full path
to corresponding *physical* directory>"
```

To expand upon the example in the previous section, the property value that you must pass along should be:

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

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

Doing so causes the mapping of `Include` to its directory and `Library` to its directory.

When SCA encounters the include directive:

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

SCA will first check to see if your project contains a physical directory named Include. If there is no such physical directory, SCA looks through its own run-time properties and sees that:

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

This tells SCA that virtual directory `Include` is actually the directory:

```
C:\WebServer\CustomerOne\inc
```

This will cause SCA to look for the file:

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

Alternately, if you choose to set this property in the `fortify-sca.properties` file, which is located in `<sca_install_dir>\Core\config`, you must escape the \ character, as well as any spaces that appear in the path of the physical directory:

```
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, which you may use on the SCA command line as follows:

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

This prompts SCA to search through the listed directories in the order specified when it is resolving a virtual include directive.

Example: Using Virtual Roots

You have a file as follows:

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

You can specify this file by using the following include:

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

Then you should set the virtual root as:

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

This will strip the `/foo` from the front of the virtual root. If you do not specify `foo` in the `ASPVirtualRoots` property, SCA will look in `C:\files\bar.asp`, and will fail.

The sequence for specifying virtual roots are 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.

Other Language Command Line Examples

This section includes examples of translating PL/SQL, T-SQL, PHP, Classic ASP written with VBScript, JavaScript, VB Script Files.

Translating PL/SQL Example

The following example demonstrates syntax for translating two PL/SQL files:

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

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

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

Translating T-SQL Example

The following example demonstrates syntax for translating two T-SQL files:

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

The following example demonstrates how to translate all T-SQL files under 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 PHP Example

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

```
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 will need to set the PHP source root:

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

where `<path>` should be the absolute or relative path to the project root directory. The relative path name will expand from the php project root directory.

Translating Classic ASP written with VBScript Example

To translate a single file named `MyASP.asp`, enter:

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

Translating JavaScript Example

To translate all JavaScript files under the `scripts` directory, enter:

```
sourceanalyzer -b mybuild "scripts/*.js"
```

Translating VB Script File Example

To translate a VB file named `myApp.vb`, enter:

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

Translating COBOL Code

This section provides information on supported technologies, preparing COBOL source files for translation, COBOL command line syntax, and auditing a COBOL scan.

Note: In order to use SCA to scan COBOL, you must have a specialized HP Fortify license specific for COBOL scanning capabilities. Contact HP Fortify for more information about scanning COBOL and the necessary license required.

Supported Technologies

SCA supports IBM Enterprise COBOL for IBM z/OS and is compatible with the following systems:

- CICS
- IMS
- DB/2 embedded SQL
- IBM WebSphere MQ

Preparing COBOL Source Files for Translation

SCA runs only on the supported systems listed in the HP Fortify System Requirements document, not on mainframe computers. This means that before you can scan a COBOL program, you must copy the following program components to the system running SCA:

- The COBOL source code
- All copybook files used by the COBOL source code
- All SQL INCLUDE files referenced by the COBOL source code

Preparing COBOL Source Code Files

If you are retrieving COBOL source files from a mainframe without COB or CBL file extensions (which is usually the case for COBOL filenames), then you must use the following command line:

```
-noextension-type COBOL <directory-file-path>
```

Specify the directory and folder with all COBOL files as the argument to SCA, and SCA will process all the files in that directory and folder without any need for COBOL file extensions.

Preparing COBOL Copybook Files

SCA does not identify copybooks by extension. All copybook files should therefore retain the names used in the COBOL source code COPY statements.

About COBOL Command Line Syntax

Free-format COBOL is the default translation and scanning mode for SCA. The basic syntax for translating a single free-format COBOL source code file is:

```
sourceanalyzer -b <build-id>
```

The basic syntax for scanning a translated free-format COBOL program is:

```
sourceanalyzer -b <build-id> -scan -f <FPR file name>
```

Working with Fixed-Format COBOL

SCA also supports fixed-format COBOL. When translating and scanning fixed-format COBOL, both the translation and scanning command lines must include the `-fixed-format` command line option. For example, the translation line syntax would look like:

```
sourceanalyzer -b <build-id> -fixed-format
```

And the scanning line syntax would look like:

```
sourceanalyzer -b <build-id> -scan -fixed-format -f <FPR file name>
```

If your COBOL code is IBM Enterprise COBOL, then it is most likely fixed format. If the COBOL translation command appears to hang indefinitely, terminate the translation by typing Ctrl-C several times, and repeat the translation command with the "-fixed-format" parameter.

Searching for COBOL Copybooks

Use the `copydirs` command line option to direct SCA to search a list of paths for copybooks and `SQL INCLUDE` files. For example, the command line syntax would look like the following:

```
sourceanalyzer -b coboltest -copydirs c:\cobol\copybooks
```

About Auditing COBOL Scans

After using the command line to scan the application, you can upload the resulting FPR file to HP Fortify Audit Workbench or HP Fortify Software Security Center and audit the application's issues.

SCA does not currently support custom rules for COBOL applications.

Chapter 10: Troubleshooting and Support

This chapter covers the following topics:

- Using the Log File to Debug Problems
- About the Translation Failed Message
- About JSP Translation Problems
- About ASPX Translation Problems
- About C/C++ Precompiled Header Files
- About Reporting Bugs and Requesting Enhancements

Using the Log File to Debug Problems

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

- On Windows: `C:\Documents and Settings\<username>\Local Settings\Application Data\Fortify\scax.xx\log`
- On other platforms: `$HOME/.fortify/scax.xx/log`

where x.xx is the version of SCA you are using.

Email the `sca.log` file as a zip file to techsupport@fortify.com for further investigation.

About the Translation Failed Message

If your C/C++ application builds successfully but you see one or more “translation failed” messages when building with SCA, edit the `<install_directory>/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 encountered by the translator. If the output indicates an incompatibility between your compiler and the HP Fortify translator, send your output to Fortify Technical Support for further investigation.

About JSP Translation Problems

SCA uses either the built-in or your specific application server's JSP compiler to translate JSP files into Java files for analysis.

If the JSP parser encounters problems when SCA is converting JSP files to Java files for analysis, 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 due to 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
- Some tag libraries or their definitions (TLD) are missing from your application

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

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

These errors are generated by the JSP parser that was used. Resolve the errors and rerun SCA.

For more information about scanning J2EE applications, see *Translating J2EE Applications* on page 20.

About ASPX Translation Problems

SCA compiles ASPX files to DLLs for analysis as follows:

- If you are using .NET 2.0 or later and Visual Studio 2005, using the Microsoft aspnet_compile compiler
- If you are using .NET 1.1 and Visual Studio 2003, trying to fetch ASPX files one at a time from the website

The compilation step can fail if:

- You have access or authentication problems with accessing the web application
- You are missing some required DLLs

In either case, you will see a message similar to the following:

```
Failed to translate the following aspx files into analysis model. Please see the log
file for any errors from the aspx precompiler and the user manual for hints on fixing
those.
<List of ASPX file names>
```

If you are using the plug-in, enable plug-in debugging and examine the plug-in log file for any errors generated by the ASPX precompiler.

If you are using the command line tool, `fortify_aspnet_compiler`, you should see the error messages on the console.

If you still cannot determine the cause of the problem, try to access some of the failed ASPX files from your browser and see what kind of errors display. If you see messages such as `cannot locate assembly`, ensure that you have the missing DLLs and rerun SCA.

If you can access the failed ASPX files from the browser, but SCA still fails to scan it, contact HP Fortify Technical Support for additional help.

For more information about scanning ASP.NET applications, see *Translating ASP.NET 1.1 (Visual Studio Version 2003) Projects* on page 23.

About 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 may accept erroneous source code without warnings or errors. This can result in a discrepancy where SCA reports translation errors even when your compiler does not.

If you use the precompiled header feature of your compiler, make sure your source code compiles cleanly by disabling precompiled headers and doing a full build.

About Reporting Bugs and Requesting Enhancements

Feedback is critical to the success of this product. To request enhancements or patches, or to report bugs, send an email to Technical Support at:

`techsupport@fortify.com`

Be sure to include the following information in the email body:

- Product: SCA
- Version Number: To determine the version number, run the following:

  ```
  sourceanalyzer -version
  ```
- Platform: (such as PC)
- OS: (such as Windows 2000)

When requesting enhancements, include a description of the feature enhancement.

When reporting bugs, provide enough details for the issue to be duplicated. The more descriptive you are, the faster we can analyze and fix the issue. Also include the log files, or the relevant portions of them, from when the issue occurred.

Appendix A: Command Line Interface

This appendix covers the command line options:

- Output Options
- Analysis Options
- Python Option
- ColdFusion Options
- Java/J2EE Options
- .NET Options
- Build Integration Options
- Runtime Options
- Other Options

Output Options

The following table describes the output options.

Table 5: Output Options

Output Option	Description
`-append`	Appends results to the file specified with `-f`. If this option is not specified, SCA adds the new findings to the FPR file, and labels the older result as `previous` findings. To use this option, the output file format must be `.fpr` or `.fvdl`. For information on the `-format` output option, see the description in this table. **Note:** When `-append` is passed to SCA and the output file specified with the `-f` option contains the results of an earlier scan, 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. The engine data section, which includes rule pack information, command line options, system properties, warnings and errors, and other information about the execution of `sourceanalyzer` (as opposed to information about the program being analyzed), is not merged, in part because there is no way to meaningfully merge this data from multiple scans. Because engine data is not merged with `-append`, HP Fortify does not certify results generated with `-append`. In general, `-append` should only be used when it is not possible to analyze an entire application at once.
`-build-label <label>`	The label of the project being scanned. The label is not used by SCA but is included in the analysis results.
`-build-project <project>`	The name of the project being scanned. The name is not used by SCA but is included in the analysis results.
`-build-version <version>`	The version of the project being scanned. The version is not used by SCA but is included in the analysis results.
`-f <file>`	The file to which results are written. If you do not specify an output file, the output is written to the terminal.

Table 5: Output Options (Continued)

Output Option	Description
`-format <format>`	Controls the output format. Valid options are `fpr`, `fvdl`, `text`, and `auto`. The default is `auto`, which selects the output format based on the file extension. **Note:** If you are using result certification, you must specify the `fpr` format. See the *Audit Workbench User's Guide* for information on result certification.

Analysis Options

The following table describes the analysis options.

Table 6: Analysis Options

Analysis Option	Description
`-disable-default-rule-type <type>`	Disables all rules of the specified type in the default Rulepacks.Can be used multiple times to specify multiple rule types. Where the value of type is the XML tag minus the suffix "Rule." For example, use DataflowSource for DataflowSourceRule elements. You can also specify specific sections of characterization rules, such as Characterization:Control flow, Characterization:Issue, and Characterization:Generic. Type is case-insensitive.
`-encoding`	Specifies the encoding. SCA allows scanning a project that contains different encoded source files. To work with a multi-encoded project, you must specify the `-encoding` option at the translation step, when SCA first reads the source code file. This encoding is remembered in the build session, and is propagated into the FVDL file.
`-filter <file_name>`	Specifies a results filter file.
`-findbugs`	Enables FindBugs analysis for Java code. The Java class directories must have been specified with the `-java-build-dir` option, described in "Java/J2EE Options" on page 53.
`-no-default-issue-rules`	Disables rules in default Rulepacks that lead directly to issues. Still loads rules that characterize the behavior of functions. **Note:** This equivalent to disabling the following rule types: DataflowSink, Semantic, Control flow, Structural, Configuration, Content, Statistical, Internal, and Characterization:Issue.
`-no-default-rules`	Specifies not to load rules from the default Rulepacks. SCA processes the Rulepacks for description elements and language libraries, but no rules are processed.
`-no-default-source-rules`	Disables source rules in the default Rulepacks. **Note:** Characterization source rules are not disabled.
`-no-default-sink-rules`	Disables sink rules in the default Rulepacks. **Note:** Characterization sink rules are not disabled.
`-disable-source-rendering`	Source files are not included in the FPR file.

Table 6: Analysis Options (Continued)

Analysis Option	Description
`-quick`	Scans the project in Quick Scan Mode, using the `fortify-sca-quickscan.properties` file. By default, this scan searches for high-confidence, high-severity issues. For more information about Quick Scan Mode, see the *Audit Workbench User's Guide*.
`-rules [<file>|<directory>]`	Specifies a custom Rulepack or directory. Can be used multiple times to specify multiple Rulepack files. If you specify a directory, all of the files in the directory with the `.bin` and `.xml` extensions are included.
`-scan`	Causes SCA to perform analysis for the specified build ID.

Python Option

The following table describes the ColdFusion option.

Table 7: Python Options

Python Option	Description
`-python-path <path name>`	Specifies the path for additional import directories. SCA does not respect the PYTHONPATH environment variable that the Python runtime system uses to find imported files. Use the `-python-path` argument to specify additional import directories.

ColdFusion Options

The following table describes the ColdFusion option.

Table 8: ColdFusion Options

ColdFusion Option	Description
`-source-base-dir`	The web application's root directory.
`-source-archive`	The application's source archive repository. You must include the -scan and -f options to use this option.

Java/J2EE Options

The following table describes the Java/J2EE options.

Table 9: Java/J2EE Options

Java/J2EE Options	Description
`-appserver`	Specifies the application server for processing JSP files: `weblogic` or `websphere`.
`-appserver-home`	Specifies the application server's home. For Weblogic, this is the path to the directory containing the `server/lib directory`. For WebSphere, this is the path to the directory containing the `JspBatchCompiler` script.
`-appserver-version`	Specifies the version of the application server. For Weblogic, valid values are `7`, `8`, `9`, and `10`. For WebSphere, the valid value is `6`.
`-cp <classpath>`, `-classpath <classpath>`	Specifies the classpath to use for analyzing Java source code. The format is same as javac: a colon or semicolon-separated list of paths. You can use SCA file specifiers. **Note:** If you do not specify the classpath with this option, the CLASSPATH environment variable is used.
`-extdirs <dirs>`	Similar to the javac `extdirs` option, accepts a colon or semicolon-separated list of directories. Any jar files found in these directories are included implicitly on the classpath.
`-java-build-dir`	Specifies one or more directories to which Java sources have been compiled. Must be specified for FindBugs results, as described in "Analysis Options" on page 51.
`-source <version>`	Indicates which version of the JDK the Java code is written for. Valid values for `version` are `1.3`, `1.4`, `1.5`, `1.6` and `1.7`. The default is `1.4`.
`-sourcepath`	Specifies the location of source files which will not be included in the scan but will be used for name resolution. The sourcepath is like classpath, except it uses source files rather than class files for resolution.

.NET Options

The following table describes the .NET options.

Table 10: .NET Options

.NET Options	Description
`-libdirs <dirs>`	Accepts a colon or semicolon-separated list of directories where system DLLs are located.

Table 10: .NET Options (Continued)

.NET Options	Description
`-dotnet-sources <directory name>`	Specifies where to look for source files for additional information. This option is automatically passed from the SCA plug-ins and Audit Workbench but when you are running SCA manually, you must provide it yourself. This option causes SCA to attempt to find any .NET classes, enums, or interfaces that are not explicitly declared in the compiled project.
`-vsversion <version>`	Specifies Visual Studio version. Valid values for `version` are `7.1` for Visual Studio Version 2003, `8.0` for Visual Studio Version 2005, 9.0 for Visual Studio 2008, 10.0 for Visual Studio 2010 and 11.0 for Visual Studio 2012. The default value is `7.1`.

Build Integration Options

The following table describes the build integration options.

Table 11: Build Integration Options

Build Integration Options	Description
`-b <build_id>`	Specifies the build ID. The build ID is used to track which files are compiled and combined to be part of a build and later to scan those files.
`-bin <binary>`	Used with `-scan` to specify 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. Can be used multiple times to specify the inclusion of multiple binaries in the scan.
`-exclude <file_pattern>`	Removes files from the list of files to translate. For example: `sourceanalyzer -cp "**/*.jar" "**/*" -exclude "**/Test.java"` **Note:** The `-exclude` option works when input files are specified on the command line; it does not work with compiler integration.
`-nc`	When specified before a compiler command line, SCA processes the source file but does not run the compiler.

Directives

The following directives can be used to list information about translation steps that have been taken. Only one directive can be used at a time and cannot be used in conjunction with normal translation or analysis steps.

Table 12: Directives

Directives	Description
`-clean`	Deletes all SCA intermediate files and build records. When a build ID is also specified, only files and build records relating to that build ID are deleted.
`-show-binaries`	Displays all objects that were created but not used in the production of any other binaries. If fully integrated into the build, it lists all of the binaries produced.

Table 12: Directives (Continued)

Directives	Description
`-show-build-ids`	Displays a list of all known build IDs. **Note:** This option may erase build IDs generated by previous versions of SCA.
`-show-build-tree`	Displays all files used to create `binary` and all files used to create those files in a tree layout. If the `-bin binary` option is not present, the tree is displayed for each binary. **Note:** This option can generate an extensive amount of information.
`-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-build-warnings`	Use with `-b <build_id>` to show all errors and warnings from the translation phase on the console. **Note**: These errors and warnings display in the results certification panel of Audit Workbench.

Runtime Options

The following table describes the runtime options.

Table 13: Runtime Options

Runtime Options	Description
`-64`	Runs SCA under the 64-bit JRE. If no 64-bit JRE is available, SCA fails.
`-logfile <file_name>`	Specifies the log file that is produced by SCA.
`-quiet`	Disables the command line progress bar.
`-verbose`	Sends verbose status messages to the console.
`-Xmx <size>`	Specifies the maximum amount of memory used by SCA. By default, it uses up to 600 MB of memory (`-Xmx600M`), which can be insufficient for large code bases. When specifying this option, ensure 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.

Other Options

The following table describes other options.

Table 14: Other Options

Other Options	Description
`@<filename>`	Reads command line options from the specified file.
`-encoding <encoding_name>`	Specifies the source file encoding type. This option is the same as the javac encoding option.
`-h, -?, -help`	Prints this summary of command line options.
`-version`	Displays the version number.
`-debug`	Enables debug mode which is useful during troubleshooting.
`-build-migration-map <old_fpr_file>`	Runs the InstanceID mapper at the end of a scan.

Specifying Files

File specifiers are expressions that allow you to easily pass a long list of files to SCA using wildcard characters. SCA recognizes two types of wildcard characters: '`*`' matches part of a filename, and '`**`' 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>
```

File specifiers can take the following forms:

Table 15: File Specifiers

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

Note: Windows and many Unix shells automatically try to expand arguments containing the '`*`' character, so file-specifier expressions should be quoted. Also, on Windows, the backslash character (\) may be used as the directory separator instead of the forward slash (/).

File specifiers do not apply to C or C++ languages.

Appendix B: Parallel Analysis Mode

This appendix covers the following topics:

- About Parallel Analysis Mode
- Hardware Requirements
- Configuring Parallel Analysis Mode
- Running in Parallel Analysis Mode

About Parallel Analysis Mode

Parallel processing allows you to reduce scan times by harnessing the multiple cores, memory, and processing power in your machine. Depending on the nature of your project and your hardware, parallel processing can reduce scan time as much as 90 percent.

While parallel processing can be enabled for all scans, scans that complete in less than 2 hours may not warrant the higher processing power requirements. For this reason, parallel processing is not the default mode of operation. You must enable parallel processing on your system and initiate it on the command line.

Hardware Requirements

Please refer to the *HP Fortify Software Security Center System Requirements* document for the latest hardware and software requirements for running SCA in parallel.

Configuring Parallel Analysis Mode

After installing SCA and completing the post-installation steps, you will need to add a couple properties to your SCA configuration file to enable parallel processing.

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

Table 16: Parallel Analysis Mode Properties

Property	Description
`com.fortify.sca.RmiWorkerMaxHeap` (default: heap size of master JVM)	Sets the heap size for the workers. The amount of memory required varies from project to project, but you don't have to allocate as much memory for the workers as you do for the master JVM. You may need to experiment with this property if you experience low memory warnings, crashes, or don't achieve a significant speed increase. The `RmiWorkerMaxHeap` property accepts values in kilobytes (K), megabytes (M), or Gigabytes (G). For example, to set the property to 500 kilobytes: `-Dcom.fortify.sca.Rmi WorkerMaxHeap = 500K`
`com.fortify.sca.ThreadCount` (default: If unchanged, SCA will use all available threads.)	You will only need to add this parameter if you need to lower the number of threads used because of a resource constraint. If you experience slow-downs or problems with your scan, reducing the number of threads used may solve the problem.

Running in Parallel Analysis Mode

To run sourceanalyzer in parallel analysis mode, add the following parameter to your command string:

`-j <# 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 would be 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 using the `com.fortify.sca.RmiWorkerMaxHeapMB` in the `fortify-sca.properties` file). You may need to balance the `-Xmx` and `-j` options to insure you don't allocate more memory than is physically available.

To figure out the maximum number of workers for your installation:

$$\frac{\textit{Total Physical Memory}}{\textit{Physical Memory Per Java Process x Number of processes}}$$

Example of translating a single file named MyServlet.java:

```
sourceanalyzer -b MyServlet -cp lib/j2ee.jar
MyServlet.java -j 6
```

The minimum value for `-j` is 2, but 3 or higher is recommended. A value of 3 is usually faster than when not running in parallel, but 4 or more should provide you with the best overall speed increases.

Appendix C: Using the sourceanalyzer Ant Task

This appendix covers the following topics:

- Using the Ant Sourceanalyzer Task
- Ant properties
- Sourceanalyzer Task Options

About the sourceanalyzer Ant Task

The `sourceanalyzer` Ant task provides a convenient way to integrate SCA into your Ant build. As discussed in Translating Java Code, translation of Java source files that are part of an Ant build is most easily accomplished using the SCA Compiler Adapter, which automatically captures input to javac task invocations. The `sourceanalyzer` task provides a convenient and flexible way to accomplish other translation tasks and to run analysis.

This section describes how to use the `sourceanalyzer` Ant task and provides an example of a sample build file with a self-contained `analysis target.rs`.

Using the Ant Sourceanalyzer Task

As with the SCA Compiler Adapter, using the `sourceanalyzer` task requires `sourceanalyzer.jar` to be on Ant's classpath, and the `sourceanalyzer` executable to be on the PATH.

The first step to using the `sourceanalyzer` task is to include a `typedef` in the `build.xml` file as follows:

```
<typedef name="sourceanalyzer" classname="com.fortify.dev.ant.SourceanalyzerTask"/>
```

Note: Only Ant 1.6 and higher supports top-level `typedef` of the sourceanalyzer task. For Ant 1.5 and lower, include the `typedef` in the target where the sourceanalyzer task is used.

Once this `typedef` is included, targets can be defined that invoke the `sourceanalyzer` task to perform translation and analysis operations exactly as if running `sourceanalyzer` from the command line. The `sourceanalyzer` task syntax is similar to that of the command line interface, but Ant fileset and path primitives can be leveraged.

The following is an example of a snippet from an Ant `build.xml` file which provides a target users can call to generate SCA results for the project. This snippet assumes that the targets `clean` and `compile` and the path `jsp.classpath` are defined elsewhere in the file. It also uses verbose and log to create a separate SCA log file for the build.

```
<available classname="com.fortify.dev.ant.SourceanalyzerTask"
          property="fortify.present"/>
<property name="sourceanalyzer.buildid" value="mybuild"/>
<!-- For debugging in a separate HP Fortify SCA log file -->
<property name="fortify.debug" value="false" />
<property name="fortify.verbose" value="false" />
 <mkdir dir="${code.build}/log" />
 <mkdir dir="${code.build}/audit" />
<tstamp/>
<property=”com.fortify.sca.PPSSilent” value=”true” />
<target name="fortify" if="fortify.present">
          <typedef name="sourceanalyzer"
          classname="com.fortify.dev.ant.SourceanalyzerTask"/>
          <!-- call clean to ensure that all source files are recompiled -->
```

```
            <antcall target="clean"/>
            <!-- call the compile target using the SCA Compiler Adapter to -->
            <!-- translate all source files-->
            <antcall target="compile">
            <!-- Log SCA in separate file -->
            <param name="com.fortify.sca.Debug" value="${fortify.debug}" />
            <param name="com.fortify.sca.Verbose" value="${fortify.verbose}" />
            <param name="com.fortify.sca.LogFile"
            value="${code.build}/log/${sourceanalyzer.buildid}-${DSTAMP}-
            ${TSTAMP}.log" />
            <param name="build.compiler"
            value="com.fortify.dev.ant.SCACompiler" />

            </antcall>
            <!-- capture all configuration files in WEB-INF directory -->
            <echo>sourceanalyzer ${web-inf}</echo>
            <sourceanalyzer buildid="${sourceanalyzer.buildid}">
            <fileset dir="${web-inf}">
                                    <include name="**/*.properties"/>
                                    <include name="**/*.xml"/>
            </fileset>
            </sourceanalyzer>
            <!-- translate all jsp files-->
            <echo>sourceanalyzer ${basedir} jsp</echo>
            <sourceanalyzer buildid="${sourceanalyzer.buildid}">
            <fileset dir="${basedir}">
                                    <include name="**/*.jsp"/>
            </fileset>
            <classpath refid="jsp.classpath"/>
            </sourceanalyzer>
            <!-- run analysis -->
            <echo>sourceanalyzer scan</echo>
            <sourceanalyzer buildid="${sourceanalyzer.buildid}"
            scan="true"
            resultsfile="issues.fpr"
            / >
  </target>
```

Ant properties

Any Ant property that begins with `com.fortify` is relayed to the `sourceanalyzer` task via `-D`. For example, setting the `com.fortify.sca.ProjectRoot` property results in `-Dcom.fortify.sca.ProjectRoot=<value>` being passed to the `sourceanalyzer` task. This is also used for the SCACompiler adapter. These properties can be set either in the build file, using the `<property>` task for example, or on the Ant command line using the `-D<property=<value>` syntax.

When using the SCACompiler adapter via the `build.compiler` setting, the `sourceanalyzer.build` Ant property is equivalent to the `buildID` attribute of the `sourceanalyzer` task, and the `sourceanalyzer.maxHeap` is equivalent to `maxHeap`. You can use either the command line or your build script to set these properties.

Sourceanalyzer Task Options

The following table contains the command line options for the `sourceanalyzer` task. Path values use colon (`:`) or semi-colon (`;`) delimited lists of file names.

Table 17: Sourceanalyzer Task Command Line Options

Attribute	Command Line Option	Description
`append`	`-append`	Appends results to the file specified with the `-f` option. If this option is not specified, SCA overwrites the file. **Note:** To use this option, the output file format must be `.fpr` or `.fvdl`. For information on the `-format` output option, see the description in this table.
`appserver`	`-appserver <appserver>`	Specifies the application server: Valid options are `weblogic` or `websphere`
`appserverHome`	`-apperserver-home <directory>`	Specifies the application server's home directory. For Weblogic, this is the path to the directory containing `server/lib` directory. For WebSphere, this is the path to the directory containing the `bin/JspBatchCompiler` script.
`appserverVersion`	`-apperserver-version <version_number>`	Specifies the version of the application server. For Weblogic: versions `7`, `8`, `9`, and `10` For WebSphere: version `6`
`bootclasspath`	`-bootclasspath <classpath>`	Specifies the JDK bootclasspath.
`buildID`	`-b <build_ID>`	Specifies the build ID. The build ID is used to track which files are compiled and linked as part of a build and later to scan those files.
`buildLabel`	`-build-label <build_label>`	Specifies the label of the project being scanned. The label is not used by SCA but is included in the analysis results.
`buildProject`	`-build-project <project_name>`	Specifies the name of the project being scanned. The name is not used by SCA but is included in the analysis results.
`buildVersion`	`-build-version <version>`	The version of the project being scanned. The version is not used by SCA but is included in the analysis results.
`classpath`	`-cp <classpath>`	Specifies the classpath to be used for Java source code. Format is same as javac (colon or semicolon-separated list of paths).
`clean`	`-clean`	This option resets the build ID. The default value is false.

Table 17: Sourceanalyzer Task Command Line Options (Continued)

Attribute	Command Line Option	Description
debug	-debug	This option enables the debug mode, which is useful during troubleshooting.
disableAnalyzers	-disable-analyzer <list_of_analyzers>	This option takes a colon-delimited list of analyzers so that you can disable multiple analyzers at once if necessary.
enableAnalyzers	-enable-analyzer <list_of_analyzers>	This option takes a colon-delimited list of analyzers so that you can enable multiple analyzers at once if necessary.
encoding	-encoding <encoding_type>	Specifies the source file encoding type. This option is the same as the javac encoding option.
extdirs	-extdirs <list_of_dirs>	Similar to the javac `extdirs` option, accepts a colon or semicolon separated list of directories. Any jar files found in these directories are included implicitly on the classpath.
filter	-filter <file_name>	Specifies the filter file.
findbugs	-findbugs	Setting this to true enables FindBugs analysis. The default value is false.
format	-format <format_type>	Controls the output format. Valid options are `fpr`, `fvdl`, `text`, and `auto`. The default is `auto`, which selects the output format based on the file extension. **Note:** If you are using results certification, you must specify the `fpr` format. See the *Audit Workbench User's Guide* for information on results certification.
javaBuildDir	-java-build-dir <directory>	Specifies one or more directors to which Java sources have been compiled. Must be specified for the `findbugs` option, as described above.
jdk	-source <value>	Indicates which version of the JDK the Java code is written for. Valid values for this option are `1.3`, `1.4`, `1.5`, `1.6` and `1.7`. The default is `1.4`. **Note:** The `source` and `JDK` options are the same. If both options are specified, the option that is specified last will take precedence.
jdkBootclasspath	-jdk-bootclasspath <classpath>	Specifies the JDK bootclasspath.
logfile	-logfile <file_name>	Specifies the log file that is produced by SCA.

Table 17: Sourceanalyzer Task Command Line Options (Continued)

Attribute	Command Line Option	Description
maxHeap	-Xmx <size>	Specifies the maximum amount of memory used by SCA. By default, it uses up to 600 MB of memory (600M), which can be insufficient for large code bases. When specifying this option, ensure that you do not allocate more memory than is physically available, because this will degrade performance. As a guideline, assuming no other memory intensive processes are running, do not allocate more than 2/3 of the available memory.
noDefaultRules	-no-default-rules	Setting this option specifies that SCA should not apply default rules when scanning.
quick	-quick-scan	Launches an SCA quick scan instead of a regular scan. Set value to true to launch a quick scan.
resultsfile	-f <absolute_path_file name>	The file to which the results are written.
rules	-rules <delimited_rules_list>	The rules option takes a list of rules files, delimited by the path separator. This is a semi-colon (;) on Windows, and a colon (:) on other platforms. For each element in this list, SCA is passed the -rules <file> command.
scan	-scan	Setting this option determines whether SCA should perform analysis on the provided build ID. The default value is false.
source	-source <value>	Indicates which version of the JDK the Java code is written for. Valid values for this option are 1.3, 1.4, 1.5, and 1.6. The default is 1.4. **Note:** The source and JDK options are the same. If both options are specified, the option that is specified last will take precedence.
sourcepath	-sourcepath <directory>	Specifies the location of source files which will not be included in the scan but will be used for resolution.
use64bit	-64	Runs SCA under the 64-bit JRE. If no 64-bit JRE is available, SCA fails.
verbose	-verbose	Setting this option sends verbose status messages to the console.

The bootclasspath, classpath, extdirs, and options may also be specified as nested elements, as with the Ant javac task. Source files can be specified via nested <fileset> elements.

The following table includes sourceanalyzer elements.

Table 18: Sourceanalyzer Task Nested Elements

Element	Ant Type	Description
`fileset`	Fileset	Specifies the files to pass to SCA.
`classpath`	Path	Specifies the classpath to be used for Java source code.
`bootclasspath`	Path	Specifies the JDK bootclasspath.
`extdirs`	Path	Similar to the javac `extdirs` option. Any jar files found in these directories are included implicitly on the classpath.
`sourcepath`	Path	Specifies the location of source files which will not be included in the scan but will be used for resolution.

Appendix D: Advanced Options

This chapter describes the following advanced options:

- About Filter Files
- Using Properties to Control Runtime Options

About Filter Files

You can create a text file for filtering out particular vulnerability instances, rules, and vulnerability categories when you run the `sourceanalyzer` command. The file is specified by the `-filter` analysis option.

Note: HP Fortify Software 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 be able to see and evaluate all issues found by SCA.

A filter file is a flat text file that can be created with any text editor. The file functions as a blacklist, such that only the filter items you do not want are specified one per line. The following filter types can be entered on a line:

- Category
- Instance ID
- Rule ID

The filters are applied at different times in the analysis process, according to the type of filter. Category and rule ID filters are applied during the initialization phase before any scans have taken place, whereas an instance ID filter is applied after the analysis phase.

Filter File Creation Example

As an example, the following output resulted from a scan of the `EightBall.java`, located in the `/Samples/basic/eightball` directory in your HP Fortify installation directory.

The following command is executed to produce the analysis results:

```
>sourceanalyzer -b eightball Eightball.java
>sourceanalyzer -b eightball -scan
```

The following result set displays, showing six detected issues.

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

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

[60AC727CCEEDE041DE984E7CE6836177 : medium : Unreleased Resource : Streams : con
trolflow ]

    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 Syste
m Output Stream : structural ]
    EightBall.java(13)
```

The sample filter file, `test_filter.txt` does the following:

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

The `test_filter.txt` file used in this example 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
```

You can create a file to test the filtered output by copying the above text into a file.

The following command is executed using the `-filter` option to specify the `test_filter.txt`:

```
[C:\Program Files\Fortify Software\HP Fortify vX.XX\Fortify SCA X.XX\Samples\basic\
eightball]>sourceanalyzer -b eightball -scan -filter test_filter.txt
```

The following result set displays:

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

[BB9F74FFA0FF75C9921D0093A0665BEB : low : J2EE Bad Practices : Leftover Debug Code :
structural]

EightBall.java(4)
```

Using Properties to Control Runtime Options

You can edit properties to define runtime options for SCA, including analysis, output, and performance tuning options. These properties can be set in four different places:

- Global configuration file (`fortify-sca.properties`): used to define global settings.
- User configuration file -- (`fortify-sca.properties` (Windows) or `.fortify-sca.properties` (non-Windows): used to define user-specified settings.
- Quick Scan configuration file (`fortify-sca-quickscan.properties`): used to define settings used when SCA is run in Quick Scan mode.
- Command line: you can define property settings on the command line

  `-D<property_name>=<property_value>`

The `fortify-sca.properties` global settings file and the `fortify-sca-quickscan.properties` file are located in the `<install_directory>/Core/config` directory. The user-specific properties files -- `fortify-sca.properties` on Windows installations and `.fortify-sca.properties` on non-Windows installations -- are located in either your Windows user directory or your Unix home directory.

You can edit all properties files directly.

Specifying the Order of Properties

SCA processes properties in a specific order, using this order to override any previously set properties with the values that you specify. You should keep this processing order in mind when making changes to the properties files.

Property definitions are processed in the following order:

- Properties specified on the command line have the highest precedence and can be specified during any scan.
- Properties specified in the Quick Scan configuration file (`fortify-sca-quickscan.properties`) are processed second, but only when the `-quick` option is used to operate in Quick Scan mode. If Quick Scan is not invoked, this file is ignored.
- Properties specified in the Global configuration file (`fortify-sca.properties`) are processed last. You should edit this file if you want to change the property values on a more permanent basis for all scans.

SCA also relies on some properties that have internally defined default values.

Table 19: HP Fortify Properties lists properties that can be defined. The default values are listed. If you want to use Quick Scan Mode, or you want to tune your application, you can make the changes as described in Table 20: on page 72.

Table 19: HP Fortify Properties

Property Name	
Default Value	**Description**
`com.fortify.sca.AbortedScanOverwritesOutput`	
false	By default, if a scan is interrupted, the partial results are written to a different output file: `<output>.partial.fpr` instead of `<output>.fpr`. If this property is set to true, the interrupted result are written to the normal outfile (`<output>.fpr`), which overwrites any full-scan results that may be present in that file.

Table 19: HP Fortify Properties (Continued)

Property Name	
Default Value	**Description**
`com.fortify.sca.Appserver`	
(none)	Specifies the application server for processing JSP files: `weblogic` or `websphere`
`com.fortify.sca.Appserver.Home`	
(none)	Specifies the application server's home. For Weblogic, this is the path to the directory containing `server/lib` directory. For WebSphere, this is the path to the directory containing the `bin/JspBatchCompiler` script.
`com.fortify.sca.Appserver.Version`	
(none)	Specifies the version of the application server. For Weblogic, valid values are `7`, `8`, `9`, and `10`. For WebSphere, the valid value is `6`.
`com.fortify.sca.fileextensions.*`	
(none)	Controls how SCA handles files with given extensions. See `fortify-sca.properties` for examples.
`com.fortify.sca.FPRDisableSrcHtml`	
(none)	If true, disables source code rendering into the FPR file.
`com.fortify.sca.NoDefaultRules`	
(none)	If true, rules from the default Rulepacks are not loaded. SCA processes the Rulepacks for description elements and language libraries, but no rules are processed.
`com.fortify.sca.NoDefaultIssueRules`	
(none)	If true, disables rules in default Rulepacks that lead directly to issues. Still loads rules that characterize the behavior of functions. **Note:** This equivalent to disabling the following rule types: DataflowSink, Semantic, Controlflow, Structural, Configuration, Content, Statistical, Internal, and Characterization:Issue.
`com.fortify.sca.DisableDefaultRuleTypes`	
(none)	Disables the specified type of rule in the default Rulepacks; where type is the XML tag minus the suffix "Rule." For example, use DataflowSource for DataflowSourceRule elements. You can also specify specific sections of characterization rules, such as Characterization:Controlflow, Characterization:Issue, and Characterization:Generic. Type is case-insensitive. Use a colon delimited list to specify multiple types of rules.
`com.fortify.sca.NoDefaultSinkRules`	
(none)	If true, disables sink rules in the default Rulepacks. **Note:** Characterization sink rules are not disabled.

Table 19: HP Fortify Properties (Continued)

Property Name	
Default Value	**Description**
com.fortify.sca.NoDefaultSourceRules	
(none)	If true, disables source rules in the default Rulepacks. **Note:** Characterization source rules are not disabled.
com.fortify.sca.ProjectRoot	
(platform dependent)	Directory used by SCA to store intermediate files generated during scans.
com.fortify.sca.ASPVirtualRoots.<virtual path>=<physical path>	
false	If true, enables support for virtual roots. This property associates virtual path names with physical path names.
com.fortify.sca.DefaultFileTypes	
java,jsp,sql,pks,pkh,pkb,xml,properties,config,dll,exe	Comma-separated list of file extensions that are picked up by default by SCA.
com.fortify.sca.compilers.*	
(none)	Can be used to inform SCA about specially named compilers. See `fortify-sca.properties` for examples.
com.fortify.sca.CfmlUndefinedVariablesAreTainted	
false	If true, treats undefined variables in a CFML page as tainted. Doing so serves as a hint to the data flow analyzer to watch out for register-globals-style vulnerabilities. However, enabling this property interferes with data flow findings in which a variable in an included page is initialized to a tainted value in an earlier-occurring included page.
com.fortify.sca.FVDLDisableProgramData	
false	If true, causes the `ProgramData` section to be excluded from the analysis results (FVDL output).
com.fortify.sca.FVDLDisableSnippets	
false	If true, code snippets are not included in the analysis results (FVDL output).
com.fortify.sca.LogFile	
${com.fortify.sca.ProjectRoot}/log/sca.log	The default location for the SCA log file.
com.fortify.sca.LogMaxSize	
(none)	When this property is set, it enables log rotation for the SCA log. The value is the number bytes that can be written to the log file before it is rotated. Must be used with `com.fortify.sca.LogMaxFiles`.
com.fortify.sca.LogMaxFiles	

Table 19: HP Fortify Properties (Continued)

Property Name	
Default Value	**Description**
(none)	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. Must be used with `com.fortify.sca.LogMaxSize`.
`com.fortify.sca.Debug`	
false	Produces a debug log file. This log file is for Technical Support purposes.
`com.fortify.sca.PPSSilent`	
false	Prompts the user with the number of lines the scan requires to analyze the source code. Set to true to suppress the prompt and automatically deduct the lines. **Note**: If the scan requires more lines than are available, the scan fails with an error indicating how many additional lines are required.
`com.fortify.sca.UnicodeInputFile`	
(none)	When set to true, this property indicates that the input file is UTF-8 based and begins with a byte-order mark (BOM). Typically, you should only set this property if you see a lexical error at Line 1, Column 1, indicating that the BOM is present.
`com.fortify.rules.SkipRulePacks`	
(none)	Semicolon-delimited list of Rulepacks to exclude from the default set. This property controls which Rulepacks are used by SCA by default. All Rulepacks installed in `<install_directory>/Core/config/rules` are used by default unless they are on this list.
`com.fortify.sca.limiters.MaxChainDepth`	
5	Controls the maximum call depth through which the data flow analyzer tracks tainted data. Increasing this value increases the coverage of data flow analysis, and results in longer analysis times. This property can be changed if you are using Quick Scan Mode: see the following table for the suggested value to use. **Note:** In this case, call depth refers to the maximum call depth on a data flow path between a taint source and sink, rather than call depth from the program entry point, such as `main()`.
`com.fortify.sca.limiters.MaxFieldDepth`	
4	Controls the maximum granularity of taint tracking through data structure member fields. This value is the number of nested fields through which taint will be tracked before the entire structure is considered tainted. Increasing this value improves the accuracy of analysis by reducing false positives, and normally increases analysis time.
`com.fortify.sca.limiters.MaxPaths`	
5	Controls the maximum number of paths to report for a single data flow vulnerability. Changing this value does not change the results that are found, only the number of data flow paths displayed for an individual result.

Table 19: HP Fortify Properties (Continued)

Property Name	
Default Value	**Description**
com.fortify.sca.limiters.MaxIndirectResolutionsForCall	
128	Controls the maximum number of virtual functions that are followed at a given call site.
com.fortify.sca.jspparserusesclasspath	
false	Allows the user to specify the classpath to the Weblogic parser. This is for Weblogic 9 and 10 only.

The following table describes the properties that can be used to tune default scanning performance. They have different defaults for Quick Scan mode, which can be adjusted by editing the `fortify-sca-quickscan.properties` file. If you want to use the recommended tuning parameters, you do not need to edit this file; however, you may find that you want to experiment with other settings to fine-tune your specific application.

Remember that properties in this file are processed only if you specify the `-quick` option on the command line when invoking your scan.

Table 20: Performance Tuning Properties

Property Name	
Values	**Description**
com.fortify.sca.FilterSet	
Default value is not set. Quick Scan value: `Critical Exposure`.	When set to `targeted`, this property runs rules only for the targeted filter set. Running only a subset of the defined rules allows the SCA scan to complete more quickly. This causes SCA to run only those rules that can cause issues identified in the named filter set, as defined by the default project template for your application. For more information about project templates, see the *Audit Workbench User's Guide.*
com.fortify.sca.FPRDisableSrcHtml	
Default value: `False`. Quick Scan value: `True`.	When set to `true`, this property prevents the generation of marked-up source files. If you plan to upload FPRs that are generated as a result of a quick scan, you must set this property to false.
com.fortify.sca.limiters.ConstraintPredicateSize	
Default value: `50000`. Quick Scan value: `10000`.	Skips calculations defined as very complex in the buffer analyzer to improve scanning time.
com.fortify.sca.limiters.BufferConfidenceInconclusiveOnTimeout	
Default value: `true`. Quick Scan value: `false`.	Skips calculations defined as very complex in the buffer analyzer to improve scanning time.
com.fortify.sca.limiters.MaxChainDepth	

Table 20: Performance Tuning Properties (Continued)

Property Name	
Values	**Description**
Default value: `5`. Quick Scan value: `4`.	Controls the maximum call depth through which the data flow analyzer tracks tainted data. Increasing this value increases the coverage of data flow analysis, and results in longer analysis times. **Note:** In this case, call depth refers to the maximum call depth on a data flow path between a taint source and sink, rather than call depth from the program entry point, such as `main()`.
`com.fortify.sca.limiters.MaxTaintDefForVar`	
Default value: `1000`. Quick Scan value: `500`.	This property sets the complexity limit for data flow precision backoff. Data flow incrementally decreases precision of analysis for functions that exceed this complexity metric for a given precision level.
`com.fortify.sca.limiters.MaxTaintDefForVarAbort`	
Default value: `4000`. Quick Scan value: `1000`.	This property sets a hard limit for function complexity. If complexity of a function exceeds this limit at the lowest precision level, the analyzer will not analyze that function.
`com.fortify.sca.DisableGlobals`	
Default value: `false`. Quick Scan value: `false`.	This property prevents the tracking of tainted data through global variables to allow faster scanning.
`com.fortify.sca.CtrlflowSkipJSPs`	
Default value: `false`. Quick Scan value: `false`.	This property skips control flow analysis of JSPs in your project.
`com.fortify.sca.NullPtrMaxFunctionTime`	
Default value: `300000`. Quick Scan value: `30000`.	This property sets a time limit, in milliseconds, for Null Pointer analysis for a single function. The default is five minutes. Setting it to a shorter limit decreases overall scanning time.
`com.fortify.sca.CtrlflowMaxFunctionTime`	
Default value: `600000`. Quick Scan value: `30000`.	This property sets a time limit, in milliseconds, for control flow analysis for a single function. The default is 10 minutes.
`com.fortify.sca.TrackPaths`	
By default, this property is not set. Quick Scan value: `NoJSP`.	This property disables path tracking for control flow analysis. Path tracking provides more detailed reporting for issues, but requires more scanning time. You can disable this for JSP only by setting it to `NoJSP`, or for all functions by setting it to `None`.
`com.fortify.sca.JdkVersion`	
Default value: 1.4	This property specifies the JDK version.

Appendix E: MSBuild Integration

This appendix covers:

- Installation
- Setting Windows Environment Variables for Touchless Integration of SCA
- Adding Custom Tasks to your MSBuild Project
- Adding Custom Tasks to Your Project

About MSBuild Integration

SCA provides the ability to translate your .NET source code as part of your MSBuild build process. With SCA's MSBuild integration, you can translate files on machines where the Visual Studio IDE has not been installed. MSBuild integration is compatible with version 2.0, 3.5, and 4.X of the MSBuild executable, allowing for the translation of the following project/source code types:

- C/C++ Console Applications (Visual Studio 2010 and above only)
- C/C++ Libraries (Visual Studio 2010 and above only)
- 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

This section describes how to launch an SCA analysis as part of your MSBuild project.

Installation

There are no installation steps required unless the machine you run MSBuild on does not include a copy of Microsoft Visual Studio. If your build machine doesn't include a copy of Microsoft Visual Studio, you will need to add `com.fortify.sca.IldasmPath=<Path_to_ildasm.exe>` to your `<SCA_Installation_Directory>\core\config\fortify-sca.properties` file.

Setting Windows Environment Variables for Touchless Integration of SCA

When integrating SCA into your MSBuild process, there are a number of Windows environment variables that you can set. If you don't set these variables, SCA will assume a default set of variables and use those. Once you've set the appropriate Windows environment variables, successive builds will use the same set until you make a change to the environment variables. The environment variables that can be set are listed in Table 21.

Table 21: Windows Environment Variables

Environment Variable	Definition	Default
`FORTIFY_MSBUILD_BUILDID`	Used to set the SCA build ID.	The build ID passed on the command line.
`FORTIFY_MSBUILD_DEBUG`	Used to put the logger and HP Fortify Static Code Analyzer in debug mode.	False
`FORTIFY_MSBUILD_MEM`	Used to set the memory used to invoke HP Fortify Static Code Analyzer (i.e., `-Xmx1200M`)	600 MB

Table 21: Windows Environment Variables (Continued)

Environment Variable	Definition	Default
FORTIFY_MSBUILD_LOG	Used to set the location for the log.	${win32.LocalAppdata}/Fortify/MSBuildPlugin
FORTIFY_MSBUILD_SCALOG	Used to set the location for the SCA log. Use an absolute path when changing.	
FORTIFY_MSBUILD_LOGALL	Use to set the plug in so that it will log every message passed to it. This will create a very large amount of information.	False

Touchless integration requires the **FortifyMSBuildTouchless.dll** located in the \Core\lib directory. It must be run from a Visual Studio command prompt.

The following is an example of the command used to run the build and launch an SCA analysis using the default environment variables, or those you have previously set:

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

Alternatively, you can call MSBuild to launch a build and SCA analysis:

```
Msbuild <solution_file> /logger:"C:\Program Files\Fortify Software\HP Fortify
vX.XX\Core\lib\FortifyMSBuildTouchless.dll" <msbuild_options>
```

Adding Custom Tasks to your MSBuild Project

Rather than using the Touchless Integration method, you can add a number of custom tasks to your MSBuild project in order to invoke SCA. These tasks must be added to an MSBuild project, not a solution. A solution file is not a valid MSBuild script. Solutions are parsed by MSBuild and a corresponding project file is created.

Table 22 lists the custom tasks you can add to your MSBuild project:

Table 22: Custom Tasks

Custom Task	Required Parameters	Optional Parameters
Fortify.TranslateTask	BuildID - the build ID for the translation BinariesFolder - the directory where the files to be translated reside VSVersion - the version of the .NET dlls being used.	References - list of dlls to be passed via the libdirs command JVMSettings - memory settings to pass to SCA LogFile - location for the SCA log file Debug - sets task and SCA to debug mode
Fortify.ScanTask	BuildID - the build ID for the scan Output - the name of the FPR file to be generated	JVMSettings - memory settings to pass to SCA LogFile - location for the SCA log file Debug - sets task and SCA to debug mode
Fortify.CleanTask	BuildID - the build ID for the scan	Debug - sets task and SCA to debug mode

Table 22: Custom Tasks (Continued)

Custom Task	Required Parameters	Optional Parameters
`Fortify.SSCTask`	AuthToken - should be defined or use username and password Project - project name ProjectVersion - project version. If undefined, a ProjectID and ProjectVersionID must be defined FPRFile - name of the file to upload to Software Security Center SSCURL - the URL for the SSC	Debug - specifies task and SCA should be invoked with debug Username - necessary if AuthToken isn't defined Password - necessary if AuthToken isn't defined ProjectID - used with ProjectVersionID if Project and ProjectVersion aren't used ProjectVersionID - used with ProjectID if Project and ProjectVersion aren't used Proxy - necessary if a proxy is required
`Fortify.CloudScanTask`	BuildID - the build ID for the translation SSCUpload - set this boolean to true to upload your output to SSC. CloudURL - the CloudURL or SSCUrl - the SSC URL	Debug - set this boolean to true for debug mode The following parameters are only used when SSCUpload is set to true: SSCToken - the SSC token Project - the project to upload to VersionName - the version you want to upload to The following parameter are only used when SSCUpload is set to false: FPRName - the name for the FPR file

Adding Custom Tasks to Your Project

You can add any of the following tasks to your project script:

- Fortify.TranslateTask
- Fortify.ScanTask
- Fortify.CleanTask
- Fortify.SSCTask
- Fortify.CloudScanTask

Adding Fortify.TranslateTask

To add Fortify.TranslateTask to your project script:

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

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

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

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

<TranslateTask BinariesFolder="$(OutDir)"

VSVersion="<Visual Studio Version>"

BuildID="TempTask"

JVMSettings="-Xmx1000M"
```

```
LogFile=”trans_task.log”

Debug=”true”/>

</Target>
```

The FortifyBuild target will be invoked after the AfterBuild target is run. The AfterBuild target is one of several default targets defined in the MSBuild target file. If one of the required parameters isn’t defined, the MSBuild will fail.

Note: If adding a new target when running MSBuild 2.0 or 3.5, you will need to remove the string `AfterTargets=”AfterBuild”` and replace `FortifyBuild` with `AfterBuild`.

Adding Fortify.ScanTask

The following code adds Fortify.ScanTask to the MSBuild project. New content is in bold text.

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

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

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

         <TranslateTask BinariesFolder="$(OutDir)"

             VSVersion="<Visual Studio Version>"

             BuildID="TempTask"

           JVMSettings="-Xmx1000M"

           LogFile="trans_task.log"

           Debug="true" />

        <ScanTask BuildID="TempTask"

             JVMSettings="-Xmx1000M"

             LogFile="scan_task.log"

             Debug="true"

             Output="Scan.fpr" />

   </Target>
```

Adding Fortify.CleanTask

The following example adds the **Fortify.CleanTask** to the MSBuild project.

```
<UsingTask TaskName="Fortify.CleanTask" AssemblyFile="<Install
Directory>\Core\lib\FortifyMSBuildTasks.dll" />
<Target Name="FortifyBuild" AfterTargets="AfterBuild" Outputs="dummy.out">

      <CleanTask BuildID="TempTask" />

</Target>
```

Adding Fortify.SSCTask

The following example adds the **Fortify.SSCTask** to the MSBuild project. New content is in bold:

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

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

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

       <TranslateTask BinariesFolder="$(OutDir)"

               VSVersion="<Visual Studio Version>"

               BuildID="TempTask"

               JVMSettings="-Xmx1000M"

               LogFile="trans_task.log"

               Debug="true" />

        <ScanTask BuildID="TempTask"

               JVMSettings="-Xmx1000M"

               LogFile="scan_task.log"

               Debug="true"

               Output="Scan.fpr" />

         <SSCTask Username="admin"

            Password="admin"
            Project="Test Project"
            ProjectVersion="Test Version 1"
            FPRFile="SSC.fpr"
            SSCURL="http://localhost:8180/SSC7"/>
    </Target>
```

Adding Fortify.CloudScanTask

If you are using CloudScan to process your scans, you can send the translated output to your cloud-based resource. The following example adds the **Fortify.CloudScanTask** to the MSBuild project:

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

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

   <CloudScanTask BuildID="TempTask"

   SSCUpload="false"

   FPRName="Scan.fpr"

   CloudURL="http://localhost:8080/cloud-ctrl" />

</Target>
```

Appendix F: Maven Integration

This appendix covers the following topics:

- About the Maven Plugin
- Installing the Maven Plugin
- Updating the Maven Plugin
- Using the Maven Plugin
- Excluding Files from the Scan
- Uninstalling the Maven Plugin
- Additional Documentation

About the Maven Plugin

SCA includes a Maven plugin which provides a means for you to add SCA clean, translation, scan, and .fpr upload capabilities to your Maven project builds. You can use the plugin directly or integrate its functionality into your build process.

The Maven plugin is located:

```
<HP_Fortify_Install_Directory>\Samples\advanced\maven-plugin
```

Inside the directory, you will find the files listed in Table 23.

Table 23: Contents of maven-plugin directory

File	Description
`pom.xml`	Project object model file.
`README.TXT`	The README text provides installation and usage instructions.
`samples` (directory)	Includes two sample project directories: `EightBall` and `MyEnterpriseApp`.
`settings.xml`	XML file that establishes the namespace to be used as it relates to Maven settings.
`src` (directory)	Location of source code, assemblies, etc.

The plugin is compatible with Maven 2.0.9 to 3.0.5, inclusive. Maven 3.X.X is recommended.

Installing the Maven Plugin

When installing the Maven plugin, we assume that the path to the SCA `bin` folder is in your PATH environment variable.

Note: If you are using SCA version 3.60, 3,70, or 3.80, you will need to edit the EOL format of the plugin source files before installing the plugin. See **Editing the Plugin Source Files** before installing the plugin. If you have a previous version of the Maven Plugin installed, see

To install the Maven plugin:

1. Open a terminal or command prompt window and navigate to the maven-plugin directory.

   ```
   <HP_Fortify_Install_Directory>\Samples\advanced\maven-plugin
   ```

2. At the prompt, type:

   ```
   mvn clean package install
   ```

Updating the Maven Plugin

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

To update the Maven Plugin on your system:

1. Open a terminal or command prompt window and navigate to the maven-plugin directory.

   ```
   <HP_Fortify_Install_Directory>\Samples\advanced\maven-plugin
   ```

2. At the prompt, type:

   ```
   mvn install
   ```

Editing the Plugin Source Files

If you are using SCA version 3.60, 3.70, or 3.80, you will need to edit the end-of-line (EOL) format of the plugin source files before installing the plugin. If you are using a version of SCA post 3.80, you do not need to edit the source files. The following instructions are based on the OS you are using.

To edit the files on a Windows system:

1. Open a Command Prompt window and Navigate to:

   ```
   Samples\advanced\maven-plugin\src\main\java\com\fortify\ps\maven\plugin\sca
   ```

2. Open the following files in a text editor:
   - `CleanMojo.java`
   - `DeleteGeneratedSourcesMojo.java`
   - `StringHelper.java`
   - `Util.java`
3. Change the EOL format for each of these files to CR+LF (DOS/Windows).

   For example, if using Notepad++ as your text editor, click **Edit-->EOL Conversion --> Windows Format**. If you make these changes after installing the plugin, you will need to uninstall it and then reinstall it after the changes have been made.

To edit the files on a Linux or Unix system:

1. Open a terminal window and navigate to the following directory:
   `<HP_Fortify_Install_Directory>/Samples/advanced/maven-plugin/src/main/java/com/fortify/ps/maven/plugin/sca`
2. Run the following commands:

   ```
   sudo dos2unix -o *.java
   sudo mac2unix -o *.java
   ```

Note: If you make these changes after installing the plugin, you will need to uninstall it and then reinstall it after the changes have been made.

To edit the files on a Macintosh system:

1. Navigate to:

   `<HP_Fortify_Install_Directory>/Samples/advanced/maven-plugin/src/main/java/com/fortify/ps/maven/plugin/sca`
2. In a text editor of your choice, change the line endings of all Java source files to LF (OS X / Unix format). If Xcode is available on your system, you can open the source files in Xcode and change the line endings to LF.

Note: If you make these changes after installing the plugin, you will need to uninstall it and then reinstall it after the changes have been made.

Testing the Plugin

After installing the maven plugin, use one of the included sample files to ensure your installation is working properly.

To test the Maven plugin using the EightBall sample file:

1. Add the directory containing the sourceanalyzer executable to the path environment variable.

   For example:

   ```
   export set PATH=$PATH:/path/to/f360/bin
   ```

   or

   ```
   set PATH=%PATH%;path\to\f360\bin
   ```

2. Type `sourceanalyzer -h` to test the PATH setting.

   It should return sourceanalyzer help.

3. Navigate to the Eightball directory:

   ```
   <HP_Fortify_Install_Directory>\Samples\advanced\maven-plugin\samples\EightBall
   ```

4. Open the `pom.xml` file in a text editor and locate the `<version>` tag. This is the version of the Maven plugin.

   Do not mistake the `<modelVersion>` tag for the `<version>` tag.

Using the Maven Plugin

You can run the package locally or integrate it as part of your build process. During the translation phase, the SCA Maven Plugin will search your jar file from the local repository and try to resolve classes in your application.

Note: In the following steps, you are provide three versions of the SCA commands. The **Short Goal Name** version of each command can only be used if you are using the latest version of the Maven plugin and you have placed a copy of the `setting.xml` file in the local repository.

To manually scan your code using the Maven Plugin:

1. Install the target application in the local repository:

   ```
   mvn install
   ```

2. Clean out the previous build using one of the following commands:

Complete	`mvn com.fortify.ps.maven.plugin:sca-maven-plugin:<ver>:clean`
Without Version ID	`mvn com.fortify.ps.maven.plugin:sca-maven-plugin:clean`
Short Goal Name	`mvn sca:clean`

3. Translate the code:

Complete	`mvn com.fortify.ps.maven.plugin:sca-maven-plugin:<ver>:translate`
Without Version ID	`mvn com.fortify.ps.maven.plugin:sca-maven-plugin:translate`
Short Goal Name	`mvn sca:translate`

4. Scan the code::

Complete	`mvn com.fortify.ps.maven.plugin:sca-maven-plugin:<ver>:scan`
Without Version ID	`mvn com.fortify.ps.maven.plugin:sca-maven-plugin:scan`
Short Goal Name	`mvn sca:scan`

where `<ver>` is the version of the Maven Plugin you're using.

Note: If you don't specify the version, Maven will call the latest version of the sca-maven-plugin in the local repository.

To scan your files as part of your build system

1. Install the target application in the local repository:

   ```
   mvn install
   ```

2. Clean out the previous build:

   ```
   mvn com.fortify.ps.maven.plugin:sca-maven-plugin:<version>:clean
   ```

3. Translate the code using one of the following options:

Translation Code Options
`sourceanalyzer -b <build id> [sca build options] mvn`
`sourceanalyzer -b <build id> [sca build options] mvn com.fortify.ps.maven.plugin:sca-maven-plugin:<ver>:translate`
`sourceanalyzer -b <build id> [sca build options] mvn com.fortify.ps.maven.plugin:sca-maven-plugin:translate`
`sourceanalyzer -b <build id> [sca build options] mvn sca:translate` **Note**: To use this version of the command, you must have placed a copy of the `setting.xml` 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 don't want to include all of the files in your project or solution, you can direct SCA to exclude selected files from your scan:

1. Create an exclusion file in a text editor.
2. Add the following line to the file you just created:

   ```
   com.fortify.sca.exclude="fileA;fileB;fileC"
   ```

   Note: File names must be separated with a semicolon. Wild cards are supported; asingle asterisk (*) can be used to match part of a file name while two asterisks (**) can be used to recursively match directories. For more information on wild cards, see

3. Add the following code to the translation step:

   ```
   -Dfortify.sca.properties.file=my.exclusions
   ```

For example, for the sample EightBall project, you would issue the following command to translate the source code:

```
mvn com.fortify.ps.maven.plugin:sca-maven-plugin:4.00:translate
-Dfortify.sca.source.version=1.6 -Dfortify.sca.properties.file=my.properties
```

Uninstalling the Maven Plugin

To uninstall the Maven Plugin, manually delete `sca-maven-plugin` from the local repository.

Additional Documentation

After the plugin has been properly installed, a new directory will be included in the following location:

```
Samples\advanced\maven-plugin\target\site
```

Open the file `index.html` to start reading the documentation. There are sections on the available options, basic usage guide, uploading scans to SSC Server, and FAQs.

Appendix G: HP Fortify Scan Wizard

This appendix covers the Fortify Scan Wizard.

About the Fortify Scan Wizard

HP Fortify Scan Wizard is a utility that enables you to quickly and easily prepare and scan project code using HP Fortify Static Code Analyzer. The Scan Wizard enables you to run your scans locally, or, if you are using HP Fortify CloudScan, in the cloud.

The wizard steps you through a number of screens, making it simple to initiate a scan. If you use HP Fortify Software Security Center, you can direct Scan Wizard to send the resultant FPR file directly to Software Security Center.

Note: Because of the nature of the script generated by Scan Wizard, a script generated on a Windows machine cannot be run on a non-Windows machine; nor can a Windows machine process a script generated on a non-Windows machine.

To use Scan Wizard, you will need:

- The location and access to the build directory or directories of the project(s) to be scanned
- The version of the Java JDK used in development, if you are scanning Java code
- The location of custom rule files (optional)

If you plan to scan your code in the cloud using HP Fortify CloudScan, you will also need:

- The URL of the cloud server

If you plan to send your scan results to Software Security Center, you will also need:

- Your Software Security Center credentials
- The server's URL
- An upload token

Note: If you don't have an upload token, the Scan Wizard will allow you to generate a multi-use token. You must have Software Security Center credentials in order to generate a multi-use token.

If you don't have Software Security Center credentials, you will also need:

- The project name
- The project version name

Launching HP Fortify Scan Wizard

1. Download and uncompress the Scan Wizard package for your OS.
2. Launch the `ScanWizard.cmd` (Windows) or `ScanWizard` (Linux or Macintosh) file.
   The file is located in the `HP-Fortify-X.XX-Scan-Wizard/bin` directory.
3. Follow the onscreen prompts.

Appendix H: Sample Files

This appendix covers the following topics:

- About the Sample Files
- Basic Samples
- Advanced Samples

About the Sample Files

Your HP Fortify software installation includes a number of sample files that you can use when testing or learning to use SCA. The sample files are located in the following directory:

```
<HP_Fortify_Install_Directory>/Samples
```

Inside the Samples directory are two sub-directories: basic and advanced. Each code sample includes a `README.txt` file that provides instructions on scanning the code in SCA and viewing the output 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 SCA with your bug tracking system.

Basic Samples

Table 24 provides a list of the sample files in the basic sub-directory (`<HP_Fortify_Install_Directory>\Samples\basic`), a brief description of the sample file, and a list of the vulnerabilities identified. Each sample includes a README.txt file that provides further details and instructions on its use.

Table 24: Basic Samples

Sample File Folder	Contents	Vulnerabilities
cpp	Includes a C++ sample file and instructions for testing a simple data flow vulnerability. It requires a gcc or cl compiler.	Command Injection Memory Leak
database	Includes a `database.pks` sample file. This SQL sample includes issues that can be found in SQL code.	Access Control: Database
eightball	Includes `EightBall.java`, a Java application that exhibits bad error handling. It requires an integer as an argument. If you supply a filename instead of an integer, it will display the contents of the file.	Path Manipulation Unreleased Resource: Streams J2EE Bad Practices: Leftover Debug Code
formatstring	Includes `formatstring.c` file. It requires a gcc or cl compiler.	Format String
javascript	Includes `sample.js`, a JavaScript file.	Cross Site Scripting (XSS) Open Redirect

Table 24: Basic Samples (Continued)

Sample File Folder	Contents	Vulnerabilities
nullpointer	Includes `NullPointerSample.java` file.	Null Dereference
php	Includes both `sink.php` and `source.php` files. Analyzing `source.php` surfaces simple Dataflow vulnerabilities and a dangerous function.	Cross Site Scripting SQL Injection
sampleOutput	Includes a sample output file (`WebGoat5.0.fpr`) from the WebGoat project located in the `Samples/advanced/webgoat` directory.	Example input for Audit Workbench.
stackbuffer	Includes `stackbuffer.c`. A gcc or cl compiler is required.	Buffer Overflow
toctou	Includes `toctou.c` file.	Time-of-Check/Time-of-Use (Race Condition)
vb6	Includes `command-injection.bas` file.	Command Injection SQL Injection
vbscript	Includes `source.asp` and `sink.asp` files.	SQL Injection

Advanced Samples

Table 25 provides a list of the sample files in the advanced subdirectory (`<HP_Fortify_Install_Directory>\Samples\advanced`). Each sample includes a README.txt file that provides further details and instructions on its use.

Table 25: Advanced Samples

Sample File Folder	Description
Bugzilla	Includes a `Build.xml` file built using the Audit Workbench bugtracker plugin framework. The plugin includes the same functionality as the built-in Bugzilla plugin so that it can be used as a guide to creating your own plugin.

Table 25: Advanced Samples (Continued)

Sample File Folder	Description
c++	Includes a sample Visual Studio 2005 solution: `Sample.sln`, `Sample1.cpp`, `Sample.vcproj`, `stafx.cpp`, `stdafx.h`. You need to have Microsoft Visual Studio Visual C/C++ 2005 (or newer) installed. You should also have the Fortify Analyzers installed, with the plugin for the Visual Studio version you are using. The code includes a Command Injection issue and an Unchecked Return Value issue.
configuration	This is a sample J2EE application that has vulnerabilities in its web module deployment descriptor -`web.xml`.
crosstier	This is a sample that has vulnerabilities spanning multiple application technologies (Java, PL/SQL, JSP, struts). The output should contain several issues of different types, including two Access Control vulnerabilities. One of these is a cross-tier result. It has a data flow trace from user input in Java code that can affect a SELECT statement in PL/SQL.
csharp	This is a simple C# program that has SQL injection vulnerabilities. Versions are included for VS2003, VS2005, VS2010 and VS2012. Upon successful completion of the scan, you should see the SQL Injection vulnerabilities and one Unreleased Resource vulnerability. Other categories may also be present, depending on the rule packs used in the scan.
customrules	Several simple source code samples and Rulepack files that illustrate rules interpreted by four different analyzers: Semantic, Dataflow, control flow, and Configuration. This directory also includes several miscellaneous real-world rules samples that may be used for scanning real applications.
ejb	A sample J2EE cross-tier application with Servlets and EJBs.
filters	A sample that uses sourceanalyzer's –filter option.
findbugs	A sample that demonstrates how to run FindBugs static analysis tool together with the Fortify Source Code Analysis Engine (Fortify SCA Engine) and filters out results that overlap.
HPQC	A sample that demonstrates the Audit Workbench bugtracker plugin framework by implementing a plugin to HP Quality Center. This plugin communicates with an HPQC server instance through the HPQC client-side addin. The bug tracker talks to the addin through a COM interface, and the addin handles the communication to the server.
java1.5	Includes `ResourceInjection.java`. The result file should have a Path Manipulation result and a J2EE Bad Practices result.

Table 25: Advanced Samples (Continued)

Sample File Folder	Description
javaAnnotations	Includes a sample application that illustrates problems that may arise from its use and how to fix the problems using the Fortify Java Annotations. The goal of this example is to illustrate how the use of Fortify Annotations can result in increased accuracy in the reported vulnerabilities. The accompanying README file illustrate the potential problems and solutions associated with vulnerability results.
JavaDoc	JavaDoc directory for the bugtrackers, public-api, and WSClient.
maven-plugin	Tests can be run on any projects that use Maven (for instance those included in the samples directory, or WebGoat 5.3: http://code.google.com/p/webgoat/)
webgoat	WebGoat test J2EE web application provided by the Open Web Application Security Project (http://www.owasp.org). This directory contains the WebGoat 5.0 sources. WebGoat java sources can be used directly for java vulnerability scanning via Fortify Source Code Analysis Engine.

Appendix I: Issue Tuning

This appendix covers the following topics:

- About Issue Tuning
- About Interprocedural Constant Propagation

About Issue Tuning

This appendix lists properties that impact the number of issues that SCA reports. The default settings have been designed to provide optimal results and should not be altered unless you are experiencing a reporting issue or have been instructed to do so by support.

Issue tuning allows you to fine tune the number and quality of result you receive from an SCA scan. This is an advanced topic and should not be necessary for the majority of users.

If you feel that SCA is reporting too many or too few issues of a particular type, you may need to adjust a property to exclude or include additional issues. Before making any changes, you may want to contact support and discuss the issue you are experiencing.

The following areas can be tuned:

- Wrapper detection
- Interprocedural constant propagation
- Selective map tracking

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

About Wrapper Detection

Wrapper detection identifies methods that wrap map operations. In SCA, 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 may get propagated through the map.

The HP Fortify Software Security Research team (SSR) provides rules describing how various APIs implement map insertion and retrieval. Taint propagation occurs when methods matching those specified in the rules are invoked. If SCA 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 identified by an SSR rule as a map operation or already identified by SCA as a wrapper.
- Directly passes its parameters to the map operation.
- Directly passes the map operation's 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 properties listed in Table 26 control the behavior of wrapper detection:

Table 26: Wrapper Detection Analysis Keys

Analysis Property	Description
None	Executes wrapper detection, including detection of nested wrappers
com.fortify.sca.EnableNestedWrappers	A value of false disables all nested wrapper detection.
com.fortify.sca.EnableWrapperDetection	A value of false disables all wrapper detection
com.fortify.sca.WrapperHeuristic	By default, the heuristic used is "moderate". You can also set this value to "strict", which will not identify any methods containing multiple callsites as wrappers.

About Interprocedural Constant Propagation

Programming languages provide keywords indicating 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 have unchanging values, and it then propagates those constant values throughout all functions in the program.

The properties listed in Table 27 control Interprocedural Constant Propagation.

Table 27: Interprocedural Constant Propagation Keys

Analysis Property	Description
`com.fortify.sca.EnableInterproceduralConstantResolution`	Enables or disables propagation of constant values across function boundaries.
`com.fortify.sca.DisableInferredConstants`	If set to "true", disables identification of constant variables without explicit `const` or `final` keywords.
`com.fortify.sca.DisableInferredConstants.NonStatic`	If set to "true", disables identification of non-static constants.

About Selective Map Operation Tracking

Selective Map Operation Tracking analysis greatly reduces the prevalence of unresolved map keys. This analysis allows SCA to find true positives in global classes without introducing an increase in the number of false positives. This algorithm is configurable via a property key that accepts any of four values. The default value, `classrule`, is appropriate in most situations. If you find that too many issues are being suppressed, you can change the value and compare the results received.

The values listed in Table 3 control Selective Map Operation Tracking.

Table 28: Selective Map Operator Tracking Values

Analysis Property	Value	Description
`com.fortify.sca.RequireMapKeys`	classrule	This is the default value of the property and does not need to be set. SCA will analyze data flow operations on maps global by classrule only when it can determine keys.
	never	Set this property equal to "never" to disable Selective Map Operation Tracking analysis. All map operations will be analyzed.
	globals	Set this property equal to "globals" to increase the aggressiveness of the analysis. SCA will analyze data flow operations on all global maps only when it can determine keys.
	always	Set this property equal to "always" for maximum aggressiveness. SCA will process data flow operations on all maps only when it can determine keys.