OpenText™ Static Application Security Testing

User Guide

Version : 25.4

PDF Generated on : 10/10/2025

Table of Contents

1. User Guide

This section provides instructions for using OpenText™ Static Application Security Testing (OpenText SAST) to scan code on most major programming platforms. This section is intended for people responsible for security audits and secure coding.

1.1. Support and documentation

When contacting Customer Support, provide the following product information:
Software Version: 25.4.0
Software Release Date: 25.4.0

Contacting Customer Support

Visit the **Customer Support** website to:

- Manage licenses and entitlements
- Create and manage technical assistance requests
- Browse documentation and knowledge articles
- Download software
- Explore the Community

For more information

For more information about OpenText Application Security Testing products, visit **Application Security**.

Product feature videos

You can find videos that highlight OpenText Application Security Software products and features on the **Fortify Unplugged YouTube™ channel**.

1.2. Change log

The following table lists changes made to this help/document. Revisions to this help/document are published between software releases only if the changes made affect product functionality.

Software release / Document version	Changes
25.4.0	Added: • Added new Xcode build and MSBuild versions (see Supported build tools) • Added new .NET (Core), C#, Java, Go, Kotlin, Scala, and Swift versions (see Language compatibility) • Added new compiler versions OpenJDK javac and Swiftc (see Supported compilers) • Added new `com.fortify.sca.rules.Islibrary` and `com.fortify.sca.rules.enablePQCRules` properties (fortify-rules.properties) • Added page on analyzing library code (Analyzing Library Code) • Added new `com.fortify.sca.EnableSubtraceFiltering` property (Translation and analysis phase properties) • Added section on Composite Filters to Excluding issues with filter files Updated: • Changed all mentions of *Translating <languages>* to *Analyzing <languages>* • Made all the language sections top level for easy identification • Simplified Analysis process overview • Build integration sections have now been moved to the respective language sections • Merged Java, Kotlin and Android sections. (see Analyzing Java, Kotlin and JSP projects) • Reorganized iOS section. (see Analyzing iOS and Xcode projects) • Moved the scan policy section out of the analysis overview, combined with filters and other ways to improve results into a new section. (see Optimizing Results) • Moved section about regex analysis under a top-level section for secret scanning Removed: • Removed ScanCentral SAST client from the OpenText SAST installer. • Removed Gradle version 6.5 and earlier versions (see Supported build tools)
25.3.0	Added: • Updated Xcode build versions (see System requirements) • Added new rule property to control FortifyRemove comments functionality (fortify-rules.properties) • Added Filtering comments using FortifyRemove • MacOS ARM installers (see Acquiring software) Updated: • Changed all mentions of *Fortify Sofware Security content* to OpenText Application Security Content Removed: • Removed xcodebuild versions 15, 15.0.1, 15.1, 15.2 (see Supported build tools)
25.2.0	Added: • System requirements • Instructions on how to create a custom scan policy (Applying a scan policy to the analysis) Updated: • Incorporated product name changes (see Product name changes) • Installer file names changed for product name change (see various topics in Installing OpenText SAST) • Test projects are excluded by default in translation of Visual Studio projects (see Visual Studio Project command-line syntax) • Added support for Jupyter notebooks (see Translating Python code) • Setting limiter properties is no longer required to translate code created using the Django or Flask framework Removed: • Properties `com.fortify.sca.SuppressLowSeverity` and `com.fortify.sca.LowSeverityCutoff` were removed because they reference metadata that is deprecated in the Rulepacks. • The `com.fortify.sca.hoa.Enable` property was removed from this helpdocument and will be removed from the product in a future release.

24.4.0	Updated: • Added installer for Linux on ARM (see Installing OpenText SAST) • Scan policies can exclude dataflow issues based on taint flags (see Applying a Scan Policy to the Analysis) • By default, NPM dependencies are excluded from the analysis phase (see Managing translation of NPM dependencies) • Support added for Flask and Jinja2 (see Translating Python code) • Added the `-gotags` option to include custom build tags in OpenText SAST translation of Go project (see Including Custom Go Build Tags and Go Properties) • Changes to the command-line options to analyze PL/SQL (see AnalyzingTranslating SQL) • Added an option to disable build tool name resolution and translate build script files as source files (see Translation Options and Translation and Analysis Phase Properties) • The `-exclude` option is supported in Ant, Bazel, Gradle, and Maven build integrations (see Integrating with Ant, Integrating with Bazel, Using Gradle Integration, Using the Fortify Maven Plugin, and Translation Options) Removed: • Modular analysis was removed from this help/document. This feature is deprecated and will be removed from the product in the next release.

1.3. Introduction

This section contains the following topics:

1.3.1. Product name changes

OpenText is in the process of changing the following product names:

Previous name	New name
Fortify Static Code Analyzer	OpenText™ Static Application Security Testing (OpenText SAST)
Fortify Software Security Center	OpenText™ Application Security
Fortify WebInspect	OpenText™ Dynamic Application Security Testing (OpenText DAST)
Fortify on Demand	OpenText™ Core Application Security
Debricked	OpenText™ Core Software Composition Analysis (OpenText Core SCA)
Fortify Applications and Tools	OpenText™ Application Security Tools

The product names have changed on product splash pages, mastheads, login pages, and other places where the product is identified. The name changes are intended to clarify product functionality and to better align the Fortify Software products with OpenText. In some cases, such as on the documentation title page, the old name might temporarily be included in parenthesis. You can expect to see more changes in future product releases.

1.3.2. OpenText SAST

OpenText SAST (Fortify Static Code Analyzer) is a set of software security analyzers that search for violations of security-specific coding rules and guidelines in a variety of languages. OpenText SAST produces analysis information to help you deliver more secure software, and make security code reviews more efficient, consistent, and complete. Its design enables you to incorporate customer-specific security rules.

For a list of supported languages, libraries, compilers, and build tools, see System requirements.

To analyze your application with OpenText SAST, you can:

- Perform the analysis directly from an IDE using one of the Secure Code Plugins: Fortify Extension for Visual Studio, Fortify Plugin for Eclipse, and Fortify Analysis Plugin for IntelliJ IDEA and Android Studio). You can also run the analysis from Fortify Audit Workbench.

  You can also view the security vulnerability analysis results in the IDE and Fortify Audit Workbench or upload the results to Application Security. For a description of the tools, see OpenText Application Security Tools.

- Integrate the analysis into your build system or run the analysis from the command line.

  This guide focuses primarily on this method of performing the analysis.

1.3.2.1. About the analyzers

OpenText SAST comprises eight vulnerability analyzers: Buffer, Configuration, Content, Control Flow, Dataflow, Null Pointer, Semantic, and Structural. Each analyzer accepts a different type of rule specifically tailored to provide the information necessary for the corresponding type of analysis performed. Rules are definitions that identify elements in the source code that might result in security vulnerabilities or are otherwise unsafe. The following table describes each analyzer.

Analyzer	Description
Dataflow	The Dataflow Analyzer detects potential vulnerabilities that involve tainted data (user-controlled input or private data) put to potentially dangerous use. The Dataflow Analyzer uses interprocedural taint propagation analysis to detect the flow of data between a site of user input (or private data) through the application to a dangerous function call or operation. For example, the Dataflow Analyzer detects whether a user-controlled input string dynamically generates HTML (Cross-Site Scripting) and detects whether a user-controlled string constructs SQL queries (SQL injection).
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 race conditions, and checks whether utilities, such as XML readers, are configured properly before being used.
Buffer	The Buffer Analyzer detects buffer overflow vulnerabilities that involve writing or reading more data than a buffer can hold. The buffer can be either stack-allocated or heap-allocated. The Buffer Analyzer uses limited interprocedural analysis to determine whether there is a condition that causes the buffer to overflow. If any execution path to a buffer leads to a buffer overflow, OpenText SAST reports it as a buffer overflow vulnerability and points out the variables that might cause the overflow. If the value of the variable causing the buffer overflow is tainted (user-controlled), then OpenText SAST reports it as well and displays the dataflow trace to show how the variable is tainted. The Buffer Analyzer also detects buffer under-read and buffer underflow conditions.
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 hard-coded secrets, cookie misconfiguration in code, and encryption weaknesses.
Configuration	The Configuration Analyzer searches for mistakes, weaknesses, and policy violations in application deployment configuration files. For example, the Configuration Analyzer checks for reasonable timeouts in user sessions in a web application. The Configuration Analyzer also performs regular expression analysis (see Regular Expression Analysis).
Semantic	The Semantic Analyzer detects potentially dangerous uses of functions and APIs at the intra-procedural level.
Content	The Content Analyzer searches for security issues and policy violations in HTML content. In addition to static HTML pages, the Content Analyzer performs these checks on files that contain dynamic HTML, such as PHP, JSP, and classic ASP files.
Null Pointer	The Null Pointer Analyzer detects dereferences of pointer variables that are assigned the null value. The Null Pointer Analyzer detection is performed at the intra-procedural level. Issues are detected only when the null assignment, the dereference, and all the paths between them occur within a single function.

1.3.3. Licensing

OpenText SAST requires a license to perform both the translation and analysis (scan) phases of security analysis (for more information about these phases, see Analysis Process).

You must download the Fortify license file for your product from the Software Licenses and Downloads (SLD) portal. Use the credentials that Customer Support has provided for access.

To install OpenText SAST, you must have a Fortify license file (`fortify.license`) and optionally you can use the Fortify License and Infrastructure Manager to manage concurrent licenses for OpenText SAST. With a LIM managed concurrent license, multiple installations of OpenText SAST can share a single license. For information about how to set up the LIM with licenses for OpenText SAST, see *OpenText™ Fortify License and Infrastructure Manager Installation and Usage Guide*. For more information about managing your LIM license from OpenText SAST, see LIM license directives.

Note

Using OpenText™ Fortify License and Infrastructure Manager (LIM) to manage concurrent licenses for OpenText SAST requires LIM version 21.2.0 or later.

1.3.4. Renewing an expired license

The license for OpenText SAST expires annually.

To update an expired license:

- Put the updated Fortify license file in the root directory where OpenText SAST is installed.

To update an expired LIM managed concurrent license, see the *OpenText™ Fortify License and Infrastructure Manager Installation and Usage Guide*.

1.3.5. OpenText Application Security Content

OpenText SAST uses a knowledge base of rules to enforce secure coding standards applicable to the codebase for static analysis. OpenText Application Security Content is required for both translation and analysis. You can download and install security content when you install OpenText SAST (see Installing OpenText SAST). Alternatively, you can download or import previously downloaded OpenText Application Security Content with the fortifyupdate command-line tool as a post-installation task (see Manually Installing OpenText Application Security Content).

OpenText Application Security Content consists of Fortify Secure Coding Rulepacks and external metadata:

- Fortify Secure Coding Rulepacks describe general secure coding idioms for popular languages and public APIs
- External metadata includes mappings from the Fortify categories to alternative categories (such as CWE, OWASP Top 10, and PCI)

OpenText provides the ability to write custom rules that add to the functionality of OpenText SAST and the Fortify Secure Coding Rulepacks. For example, you might need to enforce proprietary security guidelines or analyze a project that uses third-party libraries or other pre-compiled binaries that are not already covered by the Fortify Secure Coding Rulepacks. You can also customize the external metadata to map Fortify issues to different taxonomies, such as internal application security standards or additional compliance obligations. For instructions on how to create your own custom rules or custom external metadata, see the *OpenText™ Static Application Security Testing Custom Rules Guide*.

OpenText recommends that you periodically update the security content. You can use `fortifyupdate` to obtain the latest security content. For more information, see Updating Security Content.

1.3.6. Fortify ScanCentral SAST

You can use OpenText™ ScanCentral SAST to manage your resources by offloading the OpenText SAST analysis phase from build machines to a collection of machines provisioned for this purpose. For most languages, ScanCentral SAST can perform both the translation and the analysis (scan) phases. Users of Application Security can direct ScanCentral SAST to output the FPR file directly to the server. You have the option to install a ScanCentral SAST client when you install OpenText SAST.

You can analyze your code in one of two ways:

- If your application is written in a language supported for ScanCentral SAST translation, you can offload the translation and analysis (scan) phase of the analysis to ScanCentral SAST.
- Perform the translation phase on a local build machine and generate a mobile build session (MBS). Start the scan with ScanCentral SAST using the MBS file. In addition to freeing up the build machines, this process gives you the ability to expand the system by adding more resources as needed, without having to interrupt the build process. For more information about MBS, see Mobile build sessions.

For information about the specific supported languages for translation and how to configure and use ScanCentral SAST, see the *OpenText™ ScanCentral SAST Installation, Configuration, and Usage Guide*.

1.3.7. OpenText Application Security Tools

OpenText provides applications and tools (including Secure Code Plugins) that integrate with OpenText SAST, ScanCentral SAST, and Application Security. The following table describes the applications that are available for installation with the OpenText Application Security Tools installer. For instructions about installing the OpenText Application Security Tools, see the *OpenText™ Application Security Tools Guide*.

Application	Description
OpenText™ Fortify Audit Workbench	An application that provides a graphical user interface to help you organize, investigate, and prioritize analysis results so that developers can fix security flaws quickly.
OpenText™ Fortify Plugin for Eclipse	Adds the ability to scan and analyze the entire codebase of a project and apply software security rules that identify the vulnerabilities in your Java code from the Eclipse IDE. The results are displayed, along with descriptions of each of the security issues and suggestions for their elimination.
OpenText™ Fortify Analysis Plugin for IntelliJ IDEA and Android Studio	Adds the ability to run scans on the entire codebase of a project and apply software security rules that identify the vulnerabilities in your code from IntelliJ IDEA and Android Studio.
OpenText™ Fortify Extension for Visual Studio	Adds the ability to scan and locate security vulnerabilities in your solutions and projects and displays the scan results in Visual Studio. The results include a list of issues uncovered, descriptions of the type of vulnerability each issue represents, and suggestions on how to fix them. This extension also includes remediation functionality that works with audit results stored on a Application Security server.
OpenText™ Fortify Custom Rules Editor	An application to create and edit custom rules.
Fortify Scan Wizard	Provides a graphical user interface that enables you to prepare a script to scan your code (either locally or remotely using ScanCentral SAST) and then optionally upload the results to Application Security.
BIRTReportGenerator ReportGenerator	Command-line tools to generate issue reports (BIRT) and legacy reports from FPR files.

1.3.8. Sample projects

OpenText provides sample projects available as a separate download in the `OpenText_SAST_Fortify_Samples_<version>.zip` package.

The ZIP file contains two directories: `basic` and `advanced`. Each code sample includes a `README.txt` file that provides instructions on how to scan the code with OpenText SAST and view the results in Fortify Audit Workbench.

The `basic` directory includes an assortment of simple language-specific code samples. The `advanced` directory includes more advanced samples.

1.3.9. Related documents

This topic describes documents that provide information about OpenText Application Security Software products.

All products

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

Document / file name	Description
About OpenText Application Security Software Documentation appsec-docs-n-*<version>*.pdf	This paper provides information about how to access OpenText Application Security Software product documentation. **Note** This document is included only with the product download.
OpenText Application Security Software Release Notes appsec-rn-*<version>*.pdf	This document provides an overview of the changes made to OpenText Application Security Software for this release and important information not included elsewhere in the product documentation.

OpenText SAST

The following documents provide information about OpenText SAST (Fortify Static Code Analyzer). Unless otherwise noted, these documents are available on the Product Documentation website at www.microfocus.com/documentation/fortify-static-code-analyzer-and-tools/.

Document / file name	Description
OpenText™ Static Application Security Testing User Guide sast-ugd-*<version>*.pdf	This document describes how to install and use OpenText SAST to scan code on many of the major programming platforms. It is intended for people responsible for security audits and secure coding.
OpenText™ Static Application Security Testing Custom Rules Guide sast-cr-ugd-*<version>*.zip	This document provides the information that you need to create custom rules for OpenText SAST. This guide includes examples that apply rule-writing concepts to real-world security issues. **Note** This document is included only with the product download.
OpenText™ Fortify License and Infrastructure Manager Installation and Usage Guide lim-ugd-*<version>*.pdf	This document describes how to install, configure, and use the Fortify License and Infrastructure Manager (LIM), which is available for installation on a local Windows server and as a container image on the Docker platform.

OpenText Application Security Tools

The following documents provide information about OpenText Application Security Tools. These documents are available on the Product Documentation website at https://www.microfocus.com/documentation/fortify-static-code-analyzer-and-tools.

Document / file name	Description
OpenText™ Application Security Tools Guide sast-tgd-*<version>*.pdf	This document describes how to install application security tools. It provides an overview of the applications and command-line tools that enable you to scan your code with OpenText SAST, review analysis results, work with analysis results files, and more.
OpenText™ Fortify Audit Workbench User Guide awb-ugd-*<version>*.pdf	This document describes how to use Fortify Audit Workbench to scan software projects and audit analysis results. This guide also includes how to integrate with bug trackers, produce reports, and perform collaborative auditing.
OpenText™ Fortify Plugin for Eclipse User Guide ep-udg-*<version>*.pdf	This document provides information about how to install and use the Fortify Plugin for Eclipse to analyze and audit your code.

OpenText™ Fortify Analysis Plugin for IntelliJ IDEA and Android Studio User Guide iap-udg-<*version*>.pdf	This document describes how to install and use the Fortify Analysis Plugin for IntelliJ IDEA and Android Studio to analyze your code and optionally upload the results to Application Security.
OpenText™ Fortify Extension for Visual Studio User Guide vse-ugd-<*version*>.pdf	This document provides information about how to install and use the Fortify Extension for Visual Studio to analyze, audit, and remediate your code to resolve security-related issues in solutions and projects.

1.4. System requirements

This contentchapter describes the system requirements, supported languages, build tools, and compilers, and how to acquire the OpenText SAST software package.

This section contains the following topics:

1.4.1. Hardware requirements

System resources such as CPU, memory, and storage can drastically impact the overall analysis time for a project. It depends on many factors related to the target project codebase such as overall code size, composition, language, and code complexity. The following guidance provides some general starting points based on our experience scanning many different real-world applications.

Application size and complexity	CPU cores	RAM (GB)	Description
Small and simple	4	16	A small standalone system that runs on a server or desktop such as a batch job or a command-line tool and includes: • Less than 10,000 functions
Small and simple (dynamic language)	8	32	A standalone system that works with complex computer models such as a tax calculation system or a scheduling system and includes: • Less than 10,000 functions • Primarily a dynamic language such as JavaScript, TypeScript, Python, PHP, and Ruby
Medium	16	64–128	A three-tiered business system with transactional data processing such as a financial system or a commercial website and includes: • Less than 100,000 functions • Over one million lines of code
Large and complex	32	256	A system that delivers content such as an application server, database server, or content management system and includes: • Over 1 million functions • Several million lines of code

OpenText SAST takes advantage of all CPU cores available on your system to reduce the scan time of large projects. When you run OpenText SAST, avoid running other CPU intensive processes during the OpenText SAST execution because it expects to have the full resources of your hardware available for the scan.

Additional system resource tuning considerations:

- **Virtual systems**—Virtualization enables hardware resources to be scaled by identifying unused resources in a workload and reallocating them to other workloads. Because OpenText SAST analysis is generally a long running resource intensive process (especially in large and complex projects), OpenText recommends dedicated resources at the virtualization layer to reduce resource swapping.
- **CPU**—Overall processing power can have significant impact on the total time required for analysis. OpenText recommends a high end processor with a fast clock speed (GHz per core). It is important to note that there is a correlation between the number of cores available to the system and the amount of memory that might be needed.
- **Memory**—For more information on how to determine the amount of memory required for optimal performance, see Memory tuning. Note that analysis of dynamic languages such as JavaScript, TypeScript, Python, PHP, and Ruby require more memory during the scan phase that other languages.
- **Disk I/O**—Project translation and scan are I/O intensive activities that serialize large amounts of data and benefit from faster storage. OpenText recommends that you run analysis on faster SSD storage when possible.
- **Number of functions**—You can verify the number of functions modeled during the analysis by running a scan with the `-debug` option and looking for the last occurrence of the `NameTable.funs: ###` value in the Support log file.

See Also

Sample scans

1.4.1.1. Sample scans

These sample scans were performed using OpenText SAST version 25.4.0 on dedicated virtual machines. These scans were run with OpenText Application Security Content 25.4 Update. The following table shows the scan times you can expect for several common open-source projects.

Language	Project name	Translation time (mm:ss)	Analysis (scan) time (mm:ss)	Total issues	LOC	System configuration
.NET (C#)	SharpZipLib	01:27	14:05	606	31,863	Windows Server 2022 with 4 CPUs and 32 GB of RAM
ABAP	abap2UI5	00:13	00:52	11	59,111	Linux (AlmaLinux 9) with 4 CPUs and 32 GB of RAM
C/C++	nasm 0.98.38	00:36	04:49	738	35,997	Linux (Centos 7) with 8 CPUs and 32 GB of RAM
Java	WebGoat 8	00:17	00:59	252	23,662	Linux (AlmaLinux 9) with 4 CPUs and 32 GB of RAM
Java	WordPress for Android	00:10	01:48	534	35,276	Linux (AlmaLinux 9) with 4 CPUs and 32 GB of RAM
JavaScript	three.js	06:14	14:43	277	639,230	Linux (AlmaLinux 9) with 8 CPUs and 32 GB RAM,Java 17
PHP	CakePHP	00:22	00:03	4,182	136,594	Linux (AlmaLinux 9) with 4 CPUs and 32 GB of RAM
PHP	phpBB 3	00:34	02:05	1,305	206,873	Linux (AlmaLinux 9) with 4 CPUs and 32 GB of RAM
Python 3	numpy-1.13.3	02:24	09:28	217	563,457	Linux (AlmaLinux 9) with 4 CPUs and 32 GB RAM
Swift	MediaBrowser	00:16	01:26	9	17,699	macOS® with 4 CPUs and 16 GB of RAM
TypeScript	rxjs-7.8.1	02:19	07:24	59	204,006	Linux (AlmaLinux 9) with 8 CPUs and 32 GB RAM, Java 17

1.4.2. Supported platforms and architectures

OpenText SAST supports the platforms and architectures listed in the following table.

Operating system	Platforms	Distributions and versions	Notes
Microsoft Windows®	x64	Windows 10, 11 Windows Server 2019, 2022	
Linux®	x64 ARM	CentOS Linux 7.x (7.6 or later) Red Hat® Enterprise Linux® 7.x (7.2 or later), 8.x (8.2 or later), 9.x SUSE® Linux® Enterprise Server 15 Ubuntu® 20.04.1 LTS, 22.04.1 LTS	
macOS®	x64 M series ARM	14, 15	
IBM® AIX®	Power ISA	7.1, 7.2, 7.3	**Important** You must have the IBM XL C/C++ for AIX 16.1 Runtime environment package installed.

1.4.3. Software requirements

The OpenText SAST installation includes an embedded OpenJDK/JRE version 17.0.14, which the software requires. You do not need to install Java 17.

Note

OpenText does not recommend upgrading the embedded OpenJDK/JRE to a later version.

To use OpenText SAST, you must have Read and Write permissions for the OpenText SAST installation directory.

The following table lists software requirements for analysis of specific project types.

Language	Software	Operating systems
Visual Studio, MSBuild, or .NET projects	.NET Framework 4.8 or later (MSBuild only)	Windows
	.NET SDK 8.0	Windows, Linux
ABAP®/BSP	Fortify ABAP Extractor is supported on a system running ABAP Platform 2023 / ABAP Version 7.58.	All
Bicep	.NET SDK 8.0	Windows, Linux
COBOL	Microsoft Visual C++ 2017 Redistributable (x86) **Note** This is not a requirement for legacy COBOL analysis.	Windows
Scala	The Akka compiler plugin is available in the Maven Central Repository.	All

1.4.4. Language compatibility

OpenText SAST verifies compatibility with the language versions listed below. While these versions have been tested, OpenText SAST is designed with flexibility in mind and may successfully scan other versions not explicitly verified.

We encourage users to upgrade to the latest version of OpenText SAST and attempt scans to determine compatibility. If you encounter issues scanning a newer, unverified version or wish to scan a language not currently supported, please reach out to OpenText Support for assistance.

Language / framework	**Verified Compatibility**
.NET (Core)	2.0-10.x
.NET Framework	2.0–4.8
ABAP/BSP	6.x, 7.x
ActionScript	3.0
Apex	55–61
Bicep	0.12.x–0.15.31
C#	5–14
C	C11, C17, C23 (see Compilers)
C++	C++11, C++14, C++17, C++20 (see Compilers)
Classic ASP (with VBScript)	2.0, 3.0
COBOL	IBM Enterprise COBOL for z/OS 6.1–6.3 (CICS, IMS, DB2, and IBM MQ) Visual COBOL 6.0–8.0
ColdFusion	8–10
Dart™	2.12-3.8
Docker® (Dockerfiles)	any
Go™ programming language	1.12–1.25
HCL	2.0 **Note** HCL language support is specific to Terraform and supported cloud provider Infrastructure as Code (IaC) configurations.
HTML	5 or earlier
Java (including Android)	7–25
JavaScript	ECMAScript® 2015–2024
JSON	ECMA-404
JSP	1.2–2.1
Kotlin	1.3–2.1
MXML (Flex®)	4
Objective-C/C++	2.0 (see Compilers)
PHP	7.3–8.4
PL/SQL	8–23
Python®	2.6–3.13
Ruby	1.x
Scala	2.11–2.13, 3.3–3.6
Solidity	0.4.12–0.8.21
Swift®	5.10, 6.0 - 6.2. (see Compilers for supported swiftc versions)

T-SQL	SQL Server 2005, 2008, 2012
TypeScript	3.6–5.4
VBScript	2.0, 5.0
Visual Basic (VB.NET)	15.0–16.9
Visual Basic	6.0
XML	1.0, 1.1
YAML	1.2

1.4.4.1. Libraries, frameworks, and technologies

OpenText SAST supports the libraries, frameworks, and technologies listed in this section with dedicated Fortify Secure Coding Rulepacks and vulnerability coverage beyond core supported languages.

Java

Adobe Flex Blaze DS Ajanta Amazon Web Services (AWS) SDK Android Android Jetpack Apache Axiom Apache Axis Apache Beam Apache Beehive NetUI Apache Catalina Apache Cocoon Apache Commons Apache ECS Apache Hadoop Apache HttpComponents Apache Jasper Apache Log4j Apache Lucene Apache MyFaces Apache OGNL Apache ORO Apache POI Apache SLF4J	Apache Slide Apache Spring Security (Acegi) Apache Struts Apache Tapestry Apache Tomcat Apache Torque Apache Util Apache Velocity Apache Wicket Apache Xalan Apache Xerces ATG Dynamo Azure SDK Castor Display Tag Dom4j GDS AntiXSS Google Cloud Google Dataflow Google Guava Google Web Toolkit gRPC Gson Hibernate	iBatis IBM MQ IBM WebSphere Jackson Jakarta Activation Jakarta EE (Java EE) Jasypt Java Annotations Java Excel API JavaMail JAX-RS JAXB Jaxen JBoss JDesktop JDOM Jetty JGroups json-simple JTidy Servlet JXTA JYaml Liferay Portal MongoDB	Mozilla Rhino MyBatis MyBatis-Plus Netscape LDAP API OkHttp OpenCSV Oracle Application Development Framework (ADF) Oracle BC4J Oracle JDBC Oracle OA Framework Oracle tcDataSet Oracle XML Developer Kit (XDK) OWASP Enterprise Security API (ESAPI) OWASP HTML Sanitizer OWASP Java Encoder Plexus Archiver Realm Restlet SAP Web Dynpro Saxon SnakeYAML Spring	Spring AI Spring MVC Spring Boot Spring Data Commons Spring Data JPA Spring Data MongoDB Spring Data Redis Spring for GraphQL Spring HATEOAS Spring JMS Spring JMX Spring Messaging Spring Security Spring Webflow Spring WebSockets Spring WS Stripes Sun JavaServer Faces (JSF) Tungsten Weblogic WebSocket XStream YamlBeans ZeroTurnaround ZIP Zip4J

Kotlin

Kotlin support includes all libraries covered for Java and the following Kotlin libraries.

Kotlin standard library	Android KTX	OkHttp

Scala

Scala support includes all libraries covered for Java and the following Scala libraries.

Akka HTTP Scala Play	Scala Slick

.NET

.NET Framework, .NET Core, and .NET Standard .NET WebSockets ADO.NET Entity Framework ADODB Amazon Web Services (AWS) SDK ASP.NET MVC ASP.NET SignalR ASP.NET Web API	Azure SDK Castle ActiveRecord CsvHelper Dapper DB2 .NET Provider DotNetZip Entity Framework Entity Framework Core fastJSON gRPC	Hot Chocolate IBM Informix .NET Provider Json.NET Log4Net Microsoft ApplicationBlocks Microsoft My Framework Microsoft Practices Enterprise Library Microsoft Web Protection Library	MongoDB MySQL Connector/NET NHibernate NLog Npgsql Open XML SDK Oracle Data Provider for .NET OWASP AntiSamy Saxon	SharePoint Services SharpCompress SharpZipLib SQLite .NET Provider SubSonic Sybase ASE ADO.NET Data Provider Xamarin Xamarin Forms YamlDotNet

C

ActiveDirectory LDAP Apple System Logging (ASL)	CURL Library GLib JNI	MySQL Netscape LDAP ODBC	OpenSSL POSIX Threads SQLite	Sun RPC WinAPI

C++

Boost Smart Pointers MFC	STL WMI

SQL

Oracle ModPLSQL

PHP

ADOdb Advanced PHP Debugging CakePHP PHP Debug	PHP DOM PHP Extension PHP Hash PHP JSON PHP Mcrypt	PHP Mhash PHP Mysql PHP OCI8 PHP OpenSSL PHP PostgreSQL	PHP Reflection PHP Simdjson PHP SimpleXML PHP Smarty PHP Sodium	PHP WordPress PHP XML PHP XMLReader PHP Zend PHP Zip

JavaScript/TypeScript/HTML5

Angular Anthropic Claude Apollo Server Bluebird child-process-promise Express	Gemini API GraphQL.js Handlebars Helmet iOS JavaScript Bridge jQuery	JS-YAML LangChain Mustache Node.js Azure Storage Node.js Core OpenAI	React React Native React Native Async Storage React Router SAPUI5/OpenUI5	Sequelize Underscore.js Vertex AI Vue

Python

aiopg Amazon Web Services (AWS) Lambda Amazon SageMaker Anthropic Claude Azure Functions boto3 Django Flask	Google Cloud Graphene gRPC httplib2 Jinja2 LangChain libxml2 lxml	memcache-client _mysql MySQL Connector/Python MySQLdb OpenAI oslo.config pandas Paramiko	psycopg2 pycrypto PyCryptodome pycurl pylibmc PyMongo PySpark PyYAML	requests simplejson six TensorFlow Twisted Mail urllib3 Vertex AI WebKit

Ruby

MySQL pg	Rack SQLite	Thor

Objective-C

AFNetworking Apple AddressBook Apple AppKit Apple CFNetwork Apple ClockKit Apple CommonCrypto Apple CoreData	Apple CoreFoundation Apple CoreLocation Apple CoreServices Apple CoreTelephony Apple Foundation Apple HealthKit	Apple LocalAuthentication Apple MessageUI Apple Security Apple Social Apple UIKit	Apple WatchConnectivity Apple WatchKit Apple WebKit Hpple Objective-Zip Realm	SBJson SFHFKeychainUtils SSZipArchive ZipArchive ZipUtilities ZipZap

Swift

Alamofire Apple AddressBook Apple CFNetwork Apple ClockKit Apple CommonCrypto Apple CoreData	Apple CoreFoundation Apple CoreLocation Apple Foundation Apple HealthKit Apple LocalAuthentication	Apple MessageUI Apple Security Apple Social Apple SwiftUI Apple UIKit Apple WatchConnectivity	Apple WatchKit Apple WebKit Hpple Realm SQLite SSZipArchive	Zip ZipArchive ZIPFoundation ZipUtilities ZipZap

COBOL

Auditor CICS DLI	Micro Focus COBOL Run-time System MQ	POSIX SQL

Go

GORM logrus gRPC

Dart

Flutter

Configuration

.NET Configuration Adobe Flex (ActionScript) Configuration Ajax Frameworks Amazon Web Service (AWS) Ansible AWS CloudFormation Azure Resource Manager (ARM) Build Management	Docker Configuration (Dockerfiles) GitHub Actions Google Android Configuration iOS Property List J2EE Configuration Java Apache Axis Java Apache Log4j Configuration Java Apache Spring Security (Acegi)	Java Apache Struts Java Apache Tomcat Configuration Java Blaze DS Java Hibernate Configuration Java iBatis Configuration Java IBM WebSphere Java MyBatis Configuration	Java OWASP AntiSamy Java Spring and Spring MVC Java Spring Boot Java Spring Mail Java Spring Security Java Spring WebSockets Java Weblogic Kubernetes Mule	OpenAPI Specification Oracle Application Development Framework (ADF) PHP Configuration PHP WordPress Silverlight Configuration Terraform (AWS, Azure, GCP) WS-SecurityPolicy XML Schema

Infrastructure as Code: Amazon Web Services

API Gateway App Mesh AppSync Athena Aurora Backup Batch Certificate Manager CloudFormation CloudFront CloudTrail CloudWatch CodeBuild CodeCommit CodeStar Cognito	Config ConfigurationRecorder Database Migration Service (DMS) DataSync DocumentDB DynamoDB EC2 Elastic Block Store (EBS) Elastic Container Registry (ECR) Elastic Container Service (ECS) Elastic File System (EFS) Elastic Kubernetes Service (EKS)	Elastic Load Balancing (ELB) ElastiCache EMR FinSpace FSx Global Accelerator Glue GuardDuty HealthLake Identity and Access Management (IAM) Image Builder Key Management Service (KMS) Kinesis Kinesis Video Streams	Lightsail Location Service Lookout for Equipment Mainframe Modernization Managed Streaming for Apache Kafka (MSK) MemoryDB for Redis MQ Neptune OpenSearch Service Quantum Ledger Database (QLDB) RDS Redshift Rekognition Route 53	SageMaker Secrets Manager Simple Notification Service (SNS) Simple Queue Service (SQS) Simple Storage Service (S3) Step Functions Systems Manager Timestream Transfer Family VPC VPC Lattice WorkSpaces Family

Infrastructure as Code: Microsoft Azure

App Service Application Gateway Automation Microsoft Entra Domain Services Azure Health Data Services Azure Kubernetes Service (AKS) Batch Blob Storage	Cache for Redis Cognitive Search Container Registry Cosmos DB Database for MariaDB Database for MySQL Database for PostgreSQL Databricks Data Box	Data Factory Defender for Cloud Event Hubs Front Door Grafana Hostname Binding IoT Central IoT Hub Key Vault Logic Apps	Machine Learning MariaDB Media Services Monitor NetApp Files Private Cloud Policy Portal SignalR Service	Site Recovery Spring Apps SQL Storage Accounts Virtual Machine Scale Sets Virtual Machines Web PubSub

Infrastructure as Code: Google Cloud

Access Context Manager AlloyDB Apigee API Management App Engine Artifact Registry	Backup for GKE BigQuery Cloud Bigtable Cloud DNS Cloud Functions Cloud Key Management	Cloud Load Balancing Cloud Logging Cloud Spanner Cloud SQL Cloud Storage Compute Engine	Filestore Google Cloud Platform Google Kubernetes Engine (GKE) Identity and Access Management (IAM)	Media CDN Memorystore Pub/Sub Secret Manager Workflows

Secrets

.netrc	Defined	HashiCorp (Terraform, Vault)	New Relic	Sendbird
1Password	DES	Heroku	npm	SendGrid
Actually Good Encryption (AGE)	DigitalOcean	HexChat	NuGet	Sentry
Adafruit	Docker	HubSpot	Okta	SHA1
Adobe	Doppler	Intercom	OpenVPN	SHA256
Airtable	Droneci	Java	Password in comment	SHA512
Algolia	Dropbox	JFrog (Artifactory)	Password in connection string	Shippo
Alibaba (Aliyun)	Duffel	JSON Web Token	Password in PowerShell script	Shopify
Amazon (AWS, MWS)	Dynatrace	KDE Wallet (Kwallet)	Password in URI	Sidekiq
Apple (macOS)	EasyPost	KeePass	Password Safe	Slack
Apache HTTP	Encryption key	Kraken	PayPal (Braintree)	SonarQube
Asana	Etsy	Kucoin	Pidgin	Square
Atlassian	Facebook	LaunchDarkly	Plaid	Squarespace
Authress	Fastly	Linear	Planetscale	StackHawk
Basic access authentication	Finicity	LinkedIn	PostgreSQL	Stripe
bcrypt	Finnhub	Lob	Postman	Sumologic
Beamer	Flickr	Mailchimp	Prefect	Telegram
Bearer token	Flutterwave	Mailgun	Pulumi	Travis
Bitbucket	Frame.io	Mapbox	PuTTY	Trello
Bittrex	Freshbooks	Mattermost	PyPI	Twilio
Brevo (Sendinblue)	Git	MD5	RapidAPI	Twitch
Clojars	GitHub	MessageBird	Readme	Twitter
Code Climate	GitLab	Microsoft (Azure App Storage, Cosmos DB, Functions and Bitlocker, PowerShell, RDP, VBScript)	RSA Security	Typeform
Codecov	Gitter	Microsoft (Outlook)	Ruby (Ruby on Rails, RubyGems)	Yandex
Coinbase	GNOME	Mutt	Sauce Labs	Zendesk
Confluent	GNU (Bash)	MySQL	Secret key	
Contentful	GoCardless	Netlify	Secure Shell Protocol (SSH)	
Databricks	Google (API, Google Cloud, OAuth)			
Datadog	Grafana			

1.4.5. Supported build tools

OpenText SAST supports the build tools listed in the following table.

Build tool	Versions	Notes
Apache Ant™	1.10.x	
Bazel	6.x–7.x	Bazel integration supports Java and Python.
dotnet	6.0–10.x	
Gradle (build integration)	6.6–8.10	OpenText SAST Gradle integration supports Java, Kotlin, and C/C++.
Gradle (Gradle plugin)	6.6–8.5	OpenText SAST Gradle Plugin supports Java and Kotlin.
Apache Maven™ Software	3.6.x, 3.8.x, 3.9.x	
MSBuild	14.x–17.14	OpenText SAST MSBuild 17.4 integration is compatible with .NET 7.0 or later and .NET Framework 4.7.2 or later
xcodebuild	15.3-15.4, 16-16.4, 26	

1.4.6. Supported compilers

OpenText SAST supports the compilers listed in the following table.

Compiler	Versions	Operating systems
gcc	GNU gcc 6.x– 13	Windows, Linux, macOS
	GNU gcc 4.9–5.x	Windows, Linux, macOS, AIX
g++	GNU g++ 6.x– 13	Windows, Linux, macOS
	GNU g++ 4.9–5.x	Windows, Linux, macOS, AIX
OpenJDK javac	9, 10, 11, 12, 13, 14, 17, 21, 24, 25	Windows, Linux, macOS, AIX
Oracle javac	7, 8, 9	Windows, Linux, macOS
cl (MSVC)	2015, 2017, 2019, 2022	Windows
Clang	15.0.0, 16.0.0, 17.0.0	macOS
Swiftc	5.10, 6.0, 6.0.2, 6.0.3[1], 6.1.0, 6.1.2, 6.2	macOS

[1]OpenText SAST supports applications built in the following Xcode versions: 15.3-15.4, 16-16.4, 26.

1.4.7. OpenText Application Security Content

Fortify Secure Coding Rulepacks are backward compatible with all supported OpenText SAST versions. This ensures that Rulepack updates do not break any working OpenText SAST installation.

1.4.8. Virtual Machine support

You can run OpenText Application Security Software products on an approved operating system in virtual machine environments. You must provide dedicated CPU and memory resources that meet the minimum hardware requirements. If you find issues that cannot be reproduced on the native environments with the recommended processing, memory, and disk resources, you must work with the provider of the virtual environment to resolve them.

Note

If you run OpenText Application Security Software products in a VM environment, OpenText strongly recommends that you have CPU and memory resources fully committed to the VM to avoid performance degradation.

1.4.9. Acquiring software

OpenText SAST (Fortify Static Code Analyzer) is available as an electronic download. For instructions on how to download the software from the Software Licenses and Downloads (SLD) portal, click **Contact Us / Self Help** to review the videos and the *Quick Start Guide*.

The following table lists the available packages and describes their contents.

File name	Description
OpenText_SAST_Fortify_Windows_<*version*>.zip	OpenText SAST package for Windows This package includes: • OpenText SAST installer, which includes the following components • Fortify License and Infrastructure Manager installer • OpenText SAST Custom Rules Guide bundle • About OpenText Application Security Software Documentation **Note** OpenText Application Security Content (Rulepacks and external metadata) can be downloaded during the installation.
OpenText_SAST_Fortify_Windows_<*version*>.zip.sig	Signature file for the OpenText SAST Windows package
OpenText_SAST_Fortify_Linux-ARM_<*version*>.tar.gz	OpenText SAST package for Linux on ARM This package includes: • OpenText SAST installer, which includes the following components • OpenText SAST Custom Rules Guide bundle • About OpenText Application Security Software Documentation **Note** OpenText Application Security Content (Rulepacks and external metadata) can be downloaded during the installation.
OpenText_SAST_Fortify_Linux-ARM_<*version*>.tar.gz.sig	Signature file for the OpenText SAST Linux on ARM package
OpenText_SAST_Fortify_Linux_<*version*>.tar.gz	OpenText SAST package for Linux This package includes: • OpenText SAST installer, which includes the following components • OpenText SAST Custom Rules Guide bundle • About OpenText Application Security Software Documentation **Note** OpenText Application Security Content (Rulepacks and external metadata) can be downloaded during the installation.
OpenText_SAST_Fortify_Linux_<*version*>.tar.gz.sig	Signature file for the OpenText SAST Linux package
OpenText_SAST_Fortify_Mac_<*version*>.tar.gz	OpenText SAST package for macOS This package includes: • OpenText SAST installer, which includes the following components • OpenText SAST Custom Rules Guide bundle • About OpenText Application Security Software Documentation **Note** OpenText Application Security Content (Rulepacks and external metadata) can be downloaded during the installation.
OpenText_SAST_Fortify_Mac_<*version*>.tar.gz.sig	Signature file for the OpenText SAST macOS package

OpenText_SAST_Fortify_Mac-ARM_<*version*>.tar.gz	OpenText SAST package for macOS-ARM This package includes: • OpenText SAST installer, which includes the following components • OpenText SAST Custom Rules Guide bundle • About OpenText Application Security Software Documentation **Note** OpenText Application Security Content (Rulepacks and external metadata) can be downloaded during the installation.
OpenText_SAST_Fortify_Mac-ARM_<*version*>.tar.gz.sig	Signature file for the OpenText SAST macOS-ARM package
OpenText_SAST_Fortify_AIX_<*version*>.tar.gz	OpenText SAST package for AIX This package includes: • OpenText SAST installer • OpenText SAST Custom Rules Guide bundle • About OpenText Application Security Software Documentation
OpenText_SAST_Fortify_AIX_<*version*>.tar.gz.sig	Signature file for the OpenText SAST AIX package
OpenText_SAST_Fortify_Samples_<version>.zip	Code samples to help you learn to use OpenText SAST
OpenText_SAST_Fortify_Samples_<version>.zip.sig	Signature file for OpenText SAST code samples

1.4.10. Verifying software downloads

This topic describes how to verify the digital signature of the signed file that you downloaded from the Customer Support website. Verification ensures that the downloaded package has not been altered since it was signed and posted to the site. Before proceeding with verification, download the OpenText Application Security Software product files and their associated signature (`*.sig`) files. You are not required to verify the package to use the software, but your organization might require it for security reasons.

Preparing your system for digital signature verification

Note

These instructions describe a third-party product and might not match the specific, supported version you are using. See your product documentation for the instructions for your version.

To prepare your system for electronic media verification:

1. Go to the GnuPG website.
2. Download and install GnuPG Privacy Guard.
3. Generate a private key, as follows:
    1. Run the following command (on a Windows system, run the command without the `$` prompt):

        ```
        $ gpg --gen-key
        ```

    2. When prompted for key type, select `DSA and Elgamal`.
    3. When prompted for a key size, select `2048`.
    4. When prompted for the length of time the key should be valid, select `key does not expire`.
    5. Answer the user identification questions and provide a passphrase to protect your private key.
4. Download the OpenText GPG public keys (compressed tar file) from https://mysupport.microfocus.com/documents/10180/0/MF_public_keys.tar.gz.
5. Extract the public keys.
6. Import each downloaded key with GnuPG with the following command:

    ```
    gpg --import <path_to_key>/<key_file>
    ```

1.5. Installing OpenText SAST

This section describes how to install and uninstall OpenText SAST (Fortify Static Code Analyzer). This section also describes basic post-installation tasks. See System requirements to be sure that your system meets the minimum hardware and software requirements.

This section contains the following topics:

1.5.1. About installing OpenText SAST

This section describes how to install OpenText SAST. Several command-line tools are installed automatically with OpenText SAST (see Command-Line Tools). You can optionally include a ScanCentral SAST client and the Application Security fortifyclient utility with the OpenText SAST installation. For information about ScanCentral SAST, see the *OpenText™ ScanCentral SAST Installation, Configuration, and Usage Guide*.

You must provide a Fortify license file and optionally LIM license pool credentials during the installation. The following table lists the different ways to install OpenText SAST.

Installation method	**Instructions**
Perform the installation using a standard install wizard	Installing OpenText SAST and Applications
Perform the installation silently (unattended)	Installing OpenText SAST silently
Perform a text-based installation on non-Windows systems	Installing OpenText SAST and Applications in Text-Based Mode on Non-Windows Platforms
Perform the installation using Docker	Using Docker to Install and Run OpenText SAST

For best performance, install OpenText SAST on the same local file system where the code that you want to scan resides.

Note

On non-Windows systems, you must install OpenText SAST as a user that has a home directory with write permission. Do not install OpenText SAST as a non-root user that has no home directory.

After you complete the installation, see About the Post-Installation Tasks for additional steps you can perform to complete your system setup. You can also configure settings for runtime analysis, output, and performance of OpenText SAST by updating the installed configuration files. For information about the configuration options for OpenText SAST, see Configuration Options.

1.5.1.1. Installing OpenText SAST

To install OpenText SAST:

1. Run the installer file for your operating system to start the OpenText SAST Setup wizard:
   - Windows: `OpenText_SAST_Fortify_windows-x64_<version>.exe`
   - Linux: `OpenText_SAST_Fortify_linux-x64_<version>.run` or `OpenText_SAST_Fortify_linux-arm64_<version>.run`
   - macOS: `OpenText_SAST_Fortify_osx-x64_<version>.app.zip` or `OpenText_SAST_Fortify_osx-arm64.app.zip`

     Uncompress the ZIP file before you run the APP installer file.
   - AIX: `OpenText_SAST_Fortify_aix-ppc64_<version>.run`

   where *<version>* is the software release version, and then click **Next**.
2. Review and accept the license agreement, and then click **Next**.
3. (Optional) Select components to install, and then click **Next**.
4. If the installer detects that the system does not include the minimum software required to analyze some types of projects, a **System Requirements** page displays any missing requirements and which projects require them. Click **Next**.

   See Software requirements for all software requirements.
5. Choose where to install OpenText SAST, and then click **Next**.

   If you selected to include ScanCentral SAST client with the installation in step 3, then you must specify a location that does not include spaces in the path.

   Important

   Do not install OpenText SAST in the same directory where OpenText™ Application Security Tools is installed.
6. Specify the path to the `fortify.license` file, and then click **Next**.
7. (Optional) On the **LIM License** page, select **Yes** to manage your concurrent licenses with Fortify License and Infrastructure Manager (LIM), and then click **Next**.

   Note

   When OpenText SAST performs a task that requires a license, the application will attempt to acquire a LIM lease from the license pool. If OpenText SAST fails to acquire a license due to a communication issue with the LIM server, it will use the Fortify license file. To change this behavior, use the `com.fortify.sca.lim.WaitForInitialLicense` in the `fortify-sca.properties` file (see LIM license properties).

   1. Type the LIM API URL, the license pool name, and the license pool password.
   2. Click **Next**.

      The **LIM Proxy Settings** page opens.
   3. If connection to the LIM server requires a proxy server, type the proxy host (hostname or IP address of your proxy server) and optionally a port number.
   4. Click **Next**.
8. To update the security content for your installation:

   Note

   For deployment environments that do not have access to the internet during installation, you can update the security content using the fortifyupdate command-line tool. See Manually installing OpenText Application Security Content.

   1. Type the web address of the update server.

      To use the Fortify Rulepack update server for security content updates, keep the web address `https://update.fortify.com`. You can also use Application Security as the update server.
   2. (Optional) If connection to the update server requires a proxy server, type the proxy host and port number.
   3. If you want to update the security content manually, clear the **Update security content after installation** check box.
   4. Click **Next**.
9. Specify if you want to migrate from a previous installation on your system.

   Migrating from a previous installation preserves OpenText SAST artifact files. For more information, see About upgrading OpenText SAST.

Note

You can also migrate artifacts using the `scapostinstall` command-line tool. For information on how to use the post-install tool to migrate from a previous installation, see Migrating properties files.

To migrate artifacts from a previous installation:

1. On the **OpenText SAST (Fortify) Migration** page, select **Yes**, and then click **Next**.
2. Specify the location of the existing installation on your system, and then click **Next**.

To skip migration of artifacts from a previous release, leave the migration selection set to **No**, and then click **Next**.

10. Click **Next** on the **Ready to Install** page to install OpenText SAST, any selected components, and OpenText Application Security Content.

    If you selected to update security content, the Security Content Update Result window displays the security content update results.

11. Click **Finish** to close the Setup wizard.

1.5.1.2. Installing OpenText SAST silently

A silent installation enables you to complete the installation without any user prompts. To install silently, you need to create an option file to provide the necessary information to the installer. Using the silent installation, you can replicate the installation parameters on multiple machines.

Important
Do not install OpenText SAST in the same directory where OpenText™ Application Security Tools is installed.

When you install OpenText SAST silently, the installer does not download the Application Security by default. You can enable download of the OpenText Application Security Content in the options file or you can install the OpenText Application Security Content manually (see Manually Installing OpenText Application Security Content).

To install OpenText SAST silently:

1. Create an options file.
    1. Create a text file that contains the following line:

        ```
        fortify_license_path=<license_file_location>
        ```

        where *<license_file_location>* is the full path to your `fortify.license` file.

    2. To use a LIM license server, add the following lines with your LIM license pool credentials to the options file:

        ```
        lim_url=<lim_url>lim_pool_name=<license_pool_name>lim_pool_password=<license_pool_pwd>
        ```

    3. To use a location for OpenText Application Security Content updates that is different than the default of `https://update.fortify.com`, add the following line:

        ```
        update_server=<update_server_url>
        ```

    4. If you require a proxy server for the OpenText Application Security Content download, add the following lines:

        ```
        update_proxy_server=<proxy_server>update_proxy_port=<port_number>
        ```

    5. To enable download of OpenText Application Security Content, add the following line:

        update_security_content=1

    6. Add more installation instructions, as needed, to the options file.

        To obtain a list of installation options that you can add to your options file, open a command prompt, and then type the installer file name and the `--help` option. This command displays each available command-line option preceded with a double dash and the available parameters enclosed in angle brackets. For example, if you want to see the progress of the install displayed at the command line, add `unattendedmodeui=minimal` to your options file.

        Notes:

        - The command-line options are case-sensitive.
        - The installation options are not the same on all supported operating systems. Run the installer with `--help` to see the options available for your operating system.

        The following example Windows options file specifies the location of the license file, the location of a Application Security server and proxy information to obtain OpenText Application Security Content, a request to migrate from a previous release, and the location of the OpenText SAST installation directory:

        ```
        fortify_license_path=C:\Users\admin\Desktop\fortify.license
        update_server=https://my_ssc_host:8080/ssc
        update_proxy_server=webproxy.abc.company.com
        update_proxy_port=8080
        migrate_sca=1
        install_dir=C:\Fortify
        ```

        The following options file example is for Linux and macOS®:

        ```
        fortify_license_path=/opt/Fortify/fortify.license
        update_server=https://my_ssc_host:8080/ssc
        update_proxy_server=webproxy.abc.company.com
        update_proxy_port=8080
        migrate_sca=1
        install_dir=/opt/Fortify
        ```

2. Save the options file.

3. Run the silent install command for your operating system.

Note

You might need to run the command prompt as an administrator before you run the installer.

Windows	`OpenText_SAST_Fortify_windows-x64_<version>.exe --mode unattended --optionfile <full_path_to_options_file>`

Linux	`./OpenText_SAST_Fortify_linux-x64_<version>.run --mode unattended --optionfile <full_path_to_options_file>`
	or
	`./OpenText_SAST_Fortify_linux-arm64_<version>.run --mode unattended --optionfile <full_path_to_options_file>`

macOS®	You must uncompress the ZIP file before you run the command.
	`OpenText_SAST_Fortify_osx-x64_<version>.app/Contents/MacOS/installbuilder.sh --mode unattended --optionfile <full_path_to_options_file>`
	or
	`OpenText_SAST_Fortify_osx-arm64_<version>.app/Contents/MacOS/installbuilder.sh --mode unattended --optionfile <full_path_to_options_file>`

AIX	`./OpenText_SAST_Fortify_aix-ppc64_<version>.run --mode unattended --optionfile <full_path_to_options_file>`

The installer creates an installer log file when the installation is complete. This log file is in the following location, which depends on your operating system.

Windows	`C:\Users\<username>\AppData\Local\Temp\OpenTextSASTFortify-<version>-install.log`
Non-Windows	`/tmp/OpenTextSASTFortify-<version>-install.log`

1.5.1.3. Installing OpenText SAST in text-based mode on non-Windows platforms

You perform a text-based installation on the command line. During the installation, you are prompted for information required to complete the installation. Text-based installations are not supported on Windows systems.

Important

Do not install OpenText SAST in the same directory where OpenText™ Application Security Tools is installed.

To perform a text-based installation of OpenText SAST, run the text-based install command for your operating system as listed in the following table.

Linux	`./OpenText_SAST_Fortify_linux-x64_<version>.run --mode text`
	or
	`./OpenText_SAST_Fortify_linux-arm64_<version>.run --mode text`

MacOS	You must uncompress the provided ZIP file before you run the command.
	`OpenText_SAST_Fortify_osx-x64_<version>.app/Contents/MacOS/installbuilder.sh --mode text`
	or
	`OpenText_SAST_Fortify_osx-arm64_<version>.app/Contents/MacOS/installbuilder.sh --mode text`

AIX	`OpenText_SAST_Fortify_aix-ppc64_<version>.run --mode text`

1.5.1.4. Manually installing OpenText Application Security Content

You can install OpenText Application Security Content (Fortify Secure Coding Rulepacks and metadata) automatically during the installation. However, you can also download OpenText Application Security Content from the Fortify Rulepack update server, and then use the fortifyupdate command-line tool to install it. This option is provided for deployment environments that do not have access to the Internet during installation.

Use fortifyupdate to install OpenText Application Security Content from either a remote server or a locally downloaded file.

To install security content:

1. Open a command window and go to `<sast_install_dir>/bin/`.
2. At the command prompt, type `fortifyupdate`.

   If you have previously downloaded the OpenText Application Security Content from the Fortify Rulepack update server, run `fortifyupdate` with the `-import` option and the path to the directory where you downloaded the ZIP file.

You can also use this same tool to update your OpenText Application Security Content. For more information about the fortifyupdate command-line tool, see Updating Security Content.

1.5.2. Using Docker to install and run OpenText SAST

You can install OpenText SAST in a Docker image and then run OpenText SAST as a Docker container.

Note

You can only run OpenText SAST in Docker on supported Linux platforms.

This section contains the following topics:

1.5.2.1. Creating a Dockerfile to install OpenText SAST

This topic describes how to create a Dockerfile to install OpenText SAST in a Docker image.

The Dockerfile must include the following instructions:

1. Set a Linux system to use for the base image.

   For more information on supported platforms and architecture, see Supported platforms and architectures

   Note

   If you intend to use build tools when you run OpenText SAST, make sure that the required build tools are installed in the image. For information about using the supported build tools, see Supported build tools.

2. Copy the OpenText SAST installer, the Fortify license file, and installation options file to the Docker image using the COPY instruction.

   For instructions on how to create an installation options file, see Installing OpenText SAST silently.

3. Run the OpenText SAST installer using the RUN instruction.

   You must run the installer in unattended mode. For more information, see Installing OpenText SAST silently.

4. Run fortifyupdate to install the OpenText Application Security Content using the RUN instruction.

   Important

   OpenText SAST requires installation of the OpenText Application Security Content to perform analysis of projects. The following example installs OpenText Application Security Content from a previously downloaded local file during the build of the image. For more information about downloading and installing OpenText Application Security Content using the fortifyupdate tool, see Manually installing OpenText Application Security Content.

5. To configure the image so you can run OpenText SAST, set the entry point to the location of the installed sourceanalyzer executable using the ENTRYPOINT instruction.

   The default sourceanalyzer installation path is: `/opt/Fortify/OpenText_SAST_Fortify_<version>/bin/sourceanalyzer`.

The following is an example of a Dockerfile to install OpenText SAST:

```
FROM ubuntu:18.04
WORKDIR /app
ENV APP_HOME="/app"
ENV RULEPACK="MyRulepack.zip"

COPY fortify.license ${APP_HOME}
COPY OpenText_SAST_Fortify_linux-x64_25.4.0.run ${APP_HOME}
COPY optionFile ${APP_HOME}
COPY ${RULEPACK} ${APP_HOME}

RUN ./OpenText_SAST_Fortify_linux-x64_25.4.0.run --mode unattended \
 --optionfile "${APP_HOME}/optionFile" && \
  /opt/Fortify/OpenText_SAST_Fortify_25.4.0/bin/fortifyupdate -import ${RULEPACK} && \
  rm OpenText_SAST_Fortify_linux-x64_25.4.0.run optionFile

ENTRYPOINT ["/opt/Fortify/OpenText_SAST_Fortify_25.4.0/bin/sourceanalyzer"]
```

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

```
docker buildx build -f <docker_file> -t <image_name> "."
```

1.5.2.2. Running the container

This topic describes how to run the OpenText SAST image as a container and provides example Docker run commands for translation and scan.

Note

When you run OpenText SAST in a container and especially if you also leverage runtime container protections, make sure that OpenText SAST has the appropriate permission to run build commands (for example, javac).

To run the OpenText SAST image as a container, you must mount two directories from the host file system to the container:

- The directory that contains the source files you want to analyze.
- A temporary directory to store the OpenText SAST build session between the translate and scan phases and to share the output files (logs and FPR file) with the host.

  Specify this directory using the `–project-root` command-line option in both the OpenText SAST translate and scan commands.

The following example commands mount the input directory `/sources` in `/src` and the temporary directory in `/scratch_docker`. The image name in the example is `fortify-sast`.

Example Docker run commands for translation and scan

The following example mounts the temporary directory and the sources directory, and then runs OpenText SAST from the container for the translation phase:

```
docker run -v /scratch_local/:/scratch_docker -v /sources/:/src
-it fortify-sast –b MyProject -project-root /scratch_docker [<sca_options>] /src
```

The following example mounts the temporary directory, and then runs OpenText SAST from the container for the analysis phase:

```
docker run -v /scratch_local/:/scratch_docker
-it fortify-sast –b MyProject -project-root /scratch_docker –scan [<sca_options>] –f /scratch_docker/MyResults.fpr
```

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

1.5.3. Upgrading OpenText SAST

To upgrade OpenText SAST, install the new version in a different location than where your current version is installed and choose to migrate settings from the previous installation. This migration preserves and updates the artifact files located in the `<sast_install_dir>/Core/config` directory.

If you choose not to migrate any settings from a previous release, OpenText recommends that you save a backup of the following data if it has been modified:

- `<sast_install_dir>/Core/config/rules` folder
- `<sast_install_dir>/Core/config/customrules` folder
- `<sast_install_dir>/Core/config/ExternalMetadata` folder
- `<sast_install_dir>/Core/config/CustomExternalMetadata` folder
- `<sast_install_dir>/Core/config/server.properties` file
- `<sast_install_dir>/Core/config/scales` folder

After you install the new version, you can uninstall the previous version. For more information, see About Uninstalling OpenText SAST.

Note

You can leave the previous version installed. If you have multiple versions installed on the same system, the most recently installed version is used when you run the command from the command line.

1.5.4. About uninstalling OpenText SAST

This section describes how to uninstall OpenText SAST. You can use the standard install wizard, or you can silently install OpenText SAST. You can also perform a text-based uninstallation on non-Windows systems.

This section contains the following topics:

1.5.4.1. Uninstalling OpenText SAST

To uninstall OpenText SAST:

1. Go to the installation directory.
2. Run the uninstall command for your operating system as described in the following table.

OS	Uninstall command
Windows	`Uninstall_OpenTextSASTFortify.exe` Alternatively, you can uninstall the application from the Windows interface. See the Microsoft Windows documentation for instructions.
Linux AIX	`./Uninstall_OpenTextSASTFortify`
macOS®	`Uninstall_OpenTextSASTFortify.app`

3. You are prompted to indicate whether to remove the entire application or individual components. Make your selection, and then click **Next**.

   If you are uninstalling specific components, select the components to remove on the **Select Components to Uninstall** page, and then click **Next**.

4. You are prompted to indicate whether to remove all application settings. Do one of the following:
   - Click **Yes** to remove the application settings for the components installed with the version of OpenText SAST that you are uninstalling.

     The OpenText SAST (`sca<version>`) application settings folder is not removed.
   - Click **No** to retain the application settings on your system.

1.5.4.2. Uninstalling OpenText SAST silently

To uninstall OpenText SAST silently:

1. Go to the installation directory.
2. Run the uninstall command for your operating system as described in the following table.

OS	Uninstall command
Windows	`Uninstall_OpenTextSASTFortify.exe --mode unattended`
Linux AIX	`./Uninstall_OpenTextSASTFortify --mode unattended`
macOS®	`Uninstall_OpenTextSASTFortify.app/Contents/MacOS/installbuilder.sh --mode unattended`

Note

For Windows, Linux, and macOS®, the uninstaller removes the application settings for the components installed with the version of OpenText SAST that you are uninstalling.

1.5.4.3. Uninstalling OpenText SAST in text-based mode on non-Windows platforms

To uninstall OpenText SAST in text-based mode:

1. Go to the installation directory.
2. Run the uninstall command for your operating system as described in the following table.

OS	Uninstall command
Linux AIX	`./Uninstall_OpenTextSASTFortify --mode text`
macOS®	`Uninstall_OpenTextSASTFortify.app/Contents/MacOS/installbuilder.sh --mode text`

1.5.5. Post-installation tasks

Post-installation tasks prepare you to start using OpenText SAST.

This section contains the following topics:

1.5.5.1. Running the post-install tool

You can use the post-install command-line tool to migrate properties files from a previous version of OpenText SAST, configure OpenText Application Security Content updates, and configure settings to connect to Application Security.

To run the post-install tool:

1. Go to `<sast_install_dir>/bin/`.
2. At the command prompt, type `scapostinstall`.
3. Type one of the following:
    - To display settings, type `s`.
    - To return to the previous prompt, type `r`.
    - To exit the tool, type `q`.

1.5.5.2. Migrating properties files

To migrate properties files from a previous version of OpenText SAST to the current version installed on your system:

1. Go to `<sast_install_dir>/bin/`.
2. At the command prompt, type `scapostinstall`.
3. Type `1` to select `Migration`.
4. Type `1` to select `Static Code Analyzer Migration`.
5. Type `1` to select `Migrate from an existing Fortify installation`.
6. Type `1` to select `Set previous Fortify installation directory`.
7. Type the previous install directory.
8. Type `s` to confirm the settings.
9. Type `2` to perform the migration.
10. Type `y` to confirm.

1.5.5.3. Specifying a locale

English is the default locale for an OpenText SAST installation.

To change the locale for your OpenText SAST installation:

1. Go to *<sast_install_dir>*/bin/.
2. At the command prompt, type `scapostinstall`.
3. Type `2` to select `Settings`.
4. Type `1` to select `General`.
5. Type `1` to select `Locale`.
6. Type one of the following locale codes:
    - `en` (English)
    - `es` (Spanish)
    - `ja` (Japanese)
    - `ko` (Korean)
    - `pt_BR` (Brazilian Portuguese)
    - `zh_CN` (Simplified Chinese)
    - `zh_TW` (Traditional Chinese)

1.5.5.4. Configuring Fortify Security Content updates

Specify how you want to obtain OpenText Application Security Content. You must also specify proxy information if it is required to reach the server.

To specify settings for OpenText Application Security Content updates:

1. Go to *<sast_install_dir>*/bin/.
2. At the command prompt, type `scapostinstall`.
3. Type `2` to select `Settings`.
4. Type `2` to select `Fortify Update`.
5. To change the Fortify Rulepack update server URL, type `1`, and then type the URL.

   The default Fortify Rulepack update server URL is `https://update.fortify.com`.

6. To specify a proxy for OpenText Application Security Content updates, do the following:

   1. Type `2` to select `Proxy Server`, and then type the name of the proxy server.

      Exclude the protocol and port number (for example, some.secureproxy.com).

   2. Type `3` to select `Proxy Server Port`, and then type the proxy server port number.
   3. (Optional) You can also specify a proxy server user name (option `4`) and password (option `5`).

1.5.5.5. Configuring the connection to Application Security

Specify how to connect to Application Security. If your network uses a proxy server to reach the Application Security server, you must specify the proxy information.

To specify settings for connecting to Application Security:

1. Go to *<sast_install_dir>*/bin/.
2. At the command prompt, type `scapostinstall`.
3. Type `2` to select `Settings`.
4. Type `3` to select `Software Security Center Settings`.
5. Type `1` to select `Server URL`, and then type the Application Security server URL.
6. To specify proxy settings for the connection, do the following:
    1. Type `2` to select `Proxy Server`, and then type the name of the proxy server.

       Exclude the protocol and port number (for example, some.secureproxy.com).
    2. Type `3` to select `Proxy Server Port`, and then type the proxy server port number.
    3. To specify a proxy server user name and password, use option `4` for the username and option `5` for the password.
7. (Optional) You can also specify the following:
    - Whether to update OpenText Application Security Content from your Application Security server (option `6`)
    - The Application Security user name (option `7`)

1.5.5.6. Removing proxy server settings

If you previously specified proxy server settings for the Fortify Rulepack update server or Application Security and it is no longer required, you can remove these settings.

To remove the proxy settings for obtaining OpenText Application Security Content updates or connecting to Application Security:

1. Go to `<sast_install_dir>/bin/`.
2. At the command prompt, type `scapostinstall`.
3. Type `2` to select `Settings`.
4. Type `2` to select `Fortify Update` or type `3` to select `Software Security Center Settings`.
5. Type the number that corresponds to the proxy setting you want to remove, and then type a minus sign (`-`) to remove the setting.
6. Repeat step 5 for each proxy setting you want to remove.

1.5.5.7. Adding trusted certificates

Connection from OpenText SAST to other OpenText Application Security Software products and external systems might require communication over HTTPS. Some examples include:

- OpenText SAST by default requires an HTTPS connection to communicate with the LIM server for license management.

  The property `com.fortify.sca.lim.RequireTrustedSSLCert` determines whether the connection with the LIM server requires a trusted SSL certificate. For more information about this property, see LIM Properties.

- The fortifyupdate command-line tool uses an HTTPS connection either automatically during a Windows system installation or manually (see Manually installing OpenText Application Security Content) to update OpenText Application Security Content.

- OpenText SAST configured as a ScanCentral SAST sensor uses an HTTPS connection to communicate with the Controller.

When using HTTPS, OpenText SAST and its applications will by default apply standard checks to the presented SSL server certificate, including a check to determine if the certificate is trusted. If your organization runs its own certificate authority (CA) and OpenText SAST needs to trust connections where the server presents a certificate issued by this CA, you must configure OpenText SAST to trust the CA. Otherwise, the use of HTTPS connections might fail.

You must add the trusted certificate of the CA to the OpenText SAST keystore. The OpenText SAST keystore is in the `<sast_install_dir>/jre/lib/security/cacerts` file. You can use the keytool command to add the trusted certificate to the keystore.

To add a trusted certificate to the OpenText SAST keystore:

1. Open a command prompt, and then run the following command:

   ```
   <sast_install_dir>/jre/bin/keytool -importcert -alias <alias_name> -cacerts -file <cert_file>
   ```

   where:

   - `<alias_name>` is a unique name for the certificate you are adding.
   - `<cert_file>` is the name of the file that contains the trusted root certificate in PEM or DER format.

2. Enter the keystore password.

   Note
   The default password is `changeit`.

3. When prompted to trust this certificate, select **yes**.

1.6. Analysis process overview

This section contains the following topics:

1.6.1. Scanning Basics

The following is the fundamental sequence of commands to translate and analyze code:

1. Remove all existing OpenText SAST temporary files for the specified build ID.

   ```
   sourceanalyzer -b MyProject -clean
   ```

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

2. Translate the project code. Where available, we recommend using build integration to automate picking up your source files and configuring the translation settings correctly.
   Build integration typically takes the form:

   ```
   sourceanalyzer -b MyProject ... <build_command>
   ```

   Or manually:

   ```
   sourceanalyzer -b MyProject <files_to_analyze> <options_specific_to_language>
   ```

   For more details about translation, check under the section of the programming language you are trying to analyze.

3. Analyze the project code and save the results in a Fortify Project Results(FPR) file.

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

   For more information, see Analysis Phase.

   This can also be simplified or even performed remotely via OpenText™ ScanCentral SAST. For more information, see the *OpenText™ ScanCentral SAST Installation, Configuration, and Usage Guide*.

1.6.2. Translation phase

To successfully translate a project that is normally compiled, make sure that you have any dependencies required to build the project available. For languages that have any specific requirements, see the sections for the specific source code type.

The basic command-line syntax to perform the first step of the analysis process, file translation, is:

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

or

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

The translation phase consists of one or more invocations of OpenText SAST using the `sourceanalyzer` command. OpenText SAST uses a build ID (`-b` option) to tie the invocations together. Subsequent invocations of `sourceanalyzer` add any newly specified source or configuration files to the file list associated with the build ID.

After translation, you can use the `-show-build-warnings` directive to list any warnings and errors that occurred in the translation phase:

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

To view the files associated with a build ID, use the `-show-files` directive:

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

Special considerations for the translation phase

Consider the following special considerations before you perform the translation phase on your project:

- When you translate dynamic languages (JavaScript/TypeScript, PHP, Python, and Ruby), you must specify all source files together in one invocation. OpenText SAST does not support adding new files to the file list associated with the build ID on subsequent invocations.
- Generated code is automatically generated by a script or a tool such as a parsing tool. This code can be optimized, minimized, or large and complex. Therefore, OpenText recommends that you exclude it from translation because it would be challenging to fix any vulnerabilities OpenText SAST might report in this code. Use the `-exclude` command-line option to exclude this type of code from translation.
- To translate the project on a build machine, and then run the scan on a better performance system, see Using mobile build sessions.

1.6.3. Analysis phase

The analysis phase scans the intermediate files created during translation and creates the vulnerability results file (FPR).

This phase consists of one invocation of `sourceanalyzer`. You specify the build ID and include the `-scan` directive with any other required analysis or output options (see Analysis Options and Output Options).

The following example shows the command-line syntax to perform the analysis phase and save the results in an FPR file:

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

Note

By default, OpenText SAST includes the source code in the FPR file.

To combine multiple builds into a single scan command, add the additional builds to the command line:

```
sourceanalyzer -b MyProject1 -b MyProject2 -b MyProject3 -scan -f MyResults.fpr
```

1.6.4. Translation and analysis phase verification

Fortify Audit Workbench certification indicates whether the code analysis from a scan is complete and valid. The project summary in Fortify Audit Workbench shows the following specific information about OpenText SAST scanned code:

- List of files scanned, with file sizes and timestamps
- Java class path used for the translation (if applicable)
- Rulepacks used for the analysis
- OpenText SAST runtime settings and command-line options
- Any errors or warnings encountered during translation or analysis
- Machine and platform information

Note

To obtain result certification, you must specify FPR for the analysis phase output format.

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

1.7. Analyzing Java, Kotlin and JSP projects

This section describes how to translate Java, Kotlin as well as JSP projects, as well as projects that use a combination of these languages.

OpenText SAST supports analysis of Jakarta EE (Java EE) applications (including JSP files, configuration files, and deployment descriptors), Java Bytecode, and Java code with Lombok annotations.

This section contains the following topics:

1.7.1. Integrating with Gradle

OpenText SAST provides translation integration with projects that are built with Gradle. You can either integrate without modifying your build script or use the OpenText SAST Gradle plugin, which invokes OpenText SAST using tasks.

This section contains the following topics:

- Using Gradle integration
- Troubleshooting Gradle integration
- Using the Gradle plugin

1.7.1.1. Using Gradle integration

You can translate projects that are built with Gradle without any modification of the `build.gradle` file. When the build runs, OpenText SAST translates the source files as they are compiled. Alternatively, you can use the OpenText SAST Gradle Plugin to perform the analysis from within your Gradle build script (see Using the OpenText SAST Gradle Plugin).

See Build tools for platforms and languages supported specifically for Gradle integration. Any files in the project in unsupported languages for Gradle integration are not translated (with no error reporting). These files are therefore not analyzed, and any existing potential vulnerabilities can go undetected.

To integrate OpenText SAST into your Gradle build, make sure that the `sourceanalyzer` executable is included in the PATH environment variable. Always use the `sourceanalyzer` executable from the system PATH for all Gradle commands to build the project.

Note

If you have multiple OpenText SAST installations, make sure that the version you want to use for your Gradle projects is defined before all other OpenText SAST versions included in the PATH environment variable.

Prepend the Gradle command line with the `sourceanalyzer` command as follows:

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

Gradle integration examples

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

If your build file name is different than `build.gradle`, then include the build file name with the `--build-file` option as shown in the following example:

```
sourceanalyzer -b MyProject gradle --build-file sample.gradle clean assemble
```

You can also use the Gradle Wrapper (`gradlew`) as shown in the following example:

```
sourceanalyzer -b MyProject gradlew [<gradle_options>]
```

Translate a project and exclude a file from the translation:

```
sourceanalyzer -b MyProject -exclude "src\test\**\*" gradlew build
```

If your application uses XML or property configuration files, translate these files with a separate `sourceanalyzer` command. Use the same build ID that you used for the project files. The following are examples:

```
sourceanalyzer -b MyProject <path_to_xml_files>
sourceanalyzer -b MyProject <path_to_properties_files>
```

After OpenText SAST translates the project with gradle or gradlew, you can then perform the analysis phase and save the results in an FPR file as shown in the following example:

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

See Also

Using the OpenText SAST Gradle Plugin

1.7.1.2. Troubleshooting Gradle integration

If you use configuration caching (`--configuration-cache` option) in your Gradle build with OpenText SAST Gradle integration, the build reports the following messages:

```
Configuration cache problems found in this build.
```

You also might see a message similar to the following:

```
FAILURE: Build failed with an exception...
```

You can safely ignore this message with respect to the OpenText SAST translation because the project is translated. You can verify that the project is translated using the `-show-files` option. For example:

```
sourceanalyzer -b mybuild -show-files
```

1.7.1.3. Using the Gradle plugin

The OpenText SAST installation includes a Gradle plugin located in `<sast_install_dir>/plugins/gradle`. To use the OpenText SAST Gradle Plugin, you need to first configure the plugin for your Java or Kotlin project and then use the plugin to analyze your project. The Gradle plugin provides three OpenText SAST tasks for the analysis: sca.clean, sca.translate, and sca.scan. See Build tools for platforms and languages supported specifically for OpenText SAST Gradle plugin.

Note

If you have multiple OpenText SAST installations, make sure that the version you want to use for your Gradle projects is defined before all other OpenText SAST versions included in the PATH environment variable.

To configure the OpenText SAST Gradle Plugin:

1. Edit the Gradle settings file to specify the path to the plugin:

   Groovy DSL (`settings.gradle`):

   ```
   pluginManagement {
    repositories {
    gradlePluginPortal()
    maven {
    url = uri("file://<sast_plugin_path>")
    }
    }
   }
   ```

   Kotlin DSL (`settings.gradle.kts`):

   ```
   pluginManagement {
    repositories {
    maven(url = uri("file://<sast_plugin_path>"))
    gradlePluginPortal()
    }
   }
   ```

2. Add entries to the build script as shown in the following examples:

   Groovy DSL (`build.gradle`):

   ```
   id 'com.fortify.sca.plugins.gradlebuild' version '25.4'
   ```

   and

   ```
   SCAPluginExtension {
    buildId = "MyProject"
    options = ["-encoding", "utf-8", "-logfile", "MyProject.log",
    "-debug-verbose"]
   }
   ```

   or the following example entry excludes files from the translation:

   ```
   SCAPluginExtension {
    buildId = "MyProject"
    options = ["-encoding", "utf-8", "-logfile", "MyProject.log",
    "-debug-verbose", "-exclude", "src/test/**/*"]
   }
   ```

   Kotlin DSL (`build.gradle.kts`):

   ```
   plugins { id ("com.fortify.sca.plugins.gradlebuild") version "25.4" ...
   }
   ```

   and

   ```
   SCAPluginExtension {
    buildId = "MyProject"
    options = listOf("-encoding", "utf-8", "-logfile", "MyProject.log",
    "-debug-verbose")
   }
   ```

   or the following example entry excludes files from the translation:

```
SCAPluginExtension {
 buildId = "MyProject"
 options = listOf("-encoding", "utf-8", "-logfile", "MyProject.log",
 "-debug-verbose", "-exclude", "src/test/**/*")
}
```

3. Save and close the Gradle settings and Gradle build files.

Analyze a Java or Kotlin project with following command sequence:

- To remove all existing OpenText SAST temporary files for an existing Java or Kotlin project build, run the following:

```
gradlew sca.clean
```

- To run the translation phase for the configured Java or Kotlin project, run the following:

```
gradlew sca.translate
```

- To analyze the configured Java or Kotlin project, run the following:

```
gradlew sca.scan
```

This task runs successfully if OpenText SAST has already translated the project using the OpenText SAST Gradle Plugin.

Working with Java or Kotlin projects that have subprojects

If you have a Java or Kotlin multi-project build (with subprojects), then you must configure the OpenText SAST Gradle plugin using an `allprojects` block. This is shown in the following examples.

Groovy DSL (`build.gradle`)

```
allprojects {
 apply plugin: "com.fortify.sca.plugins.gradlebuild"
 SCAPluginExtension {
 buildId = "MyProject"
 options = ["-encoding", "utf-8", "-logfile", "MyProject.log",
 "-debug-verbose"]
 ...
 }
}
```

Kotlin DSL (`build.gradle.kts`):

```
allprojects {
 apply(plugin = "com.fortify.sca.plugins.gradlebuild")
 SCAPluginExtension {
 buildId = "MyProject"
 options = listOf("-encoding", "utf-8", "-logfile", "MyProject.log",
 "-debug-verbose")
 ...
 }
}
```

See Also

Using Gradle Integration

1.7.2. Integrating with Maven

OpenText SAST includes a Maven plugin that provides a way to add the following capabilities to your Maven project builds:

- OpenText SAST clean, translate, scan
- OpenText SAST export mobile build session (MBS) for a translated project
- Send translated code to ScanCentral SAST
- Upload results to Application Security

You can use the plugin directly or integrate its functionality into your build process.

This section contains the following topics:

1.7.2.1. Installing and updating the Fortify Maven Plugin

The Fortify Maven Plugin is located in `<sast_install_dir>/plugins/maven`. This directory contains a binary and a source version of the plugin in both zip and tarball archives. To install the plugin, extract the version (binary or source) that you want to use, and then follow the instructions in the included `README.TXT` file. Perform the installation in the directory where you extracted the archive.

For information about supported versions of Maven, see Build tools.

If you have a previous version of the Fortify Maven Plugin installed, then install the latest version.

Uninstalling the Fortify Maven Plugin

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

1.7.2.2. Testing the Fortify Maven Plugin installation

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

To test the Fortify Maven Plugin using the Eightball sample file:

1. Add the directory that contains the `sourceanalyzer` executable to the path environment variable.

   For example:

   ```
   export set PATH=$PATH:/<sast_install_dir>/bin
   ```

   or

   ```
   set PATH=%PATH%;<sast_install_dir>/bin
   ```

2. Type `sourceanalyzer -version` to test the path setting.

   OpenText SAST displays the version information if the path setting is correct.

3. Go to the sample Eightball directory: `<root_dir>/samples/EightBall`.

4. Type the following command:

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

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

   Note

   To see the version of the Fortify Maven Plugin, open the `pom.xml` file that you extracted in `<root_dir>` in a text editor. The Fortify Maven Plugin version is specified in the `<version>` element.

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

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

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

1.7.2.3. Using the Fortify Maven Plugin

There are two ways to perform an analysis on a maven project:

- In an OpenText SAST build integration

  In this method, prepend the maven command used to build your project with the `sourceanalyzer` command and any OpenText SAST options. To analyze your files as part of an OpenText SAST build integration:

  1. Clean out the previous build:

     ```
     sourceanalyzer -b MyProject -clean
     ```

  2. Translate the code:

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

     Examples:

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

     ```
     sourceanalyzer -b MyProject -exclude "**/Test/*.java" mvn clean install
     ```

     See Command-Line Interface for descriptions of available OpenText SAST options.

  3. Run the scan and save the results in an FPR file as shown in the following example:

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

- As a Maven Plugin

  In this method, you perform the analysis tasks as goals with the `mvn` command. For example, use the following command to translate source code:

  ```
  mvn com.fortify.sca.plugins.maven:sca-maven-plugin:25.4.0:translate
  ```

  For example, use the following command to translate source code and exclude test files:

  ```
  mvn -Dfortify.sca.exclude=“**/Test/*.java” com.fortify.sca.plugins.maven:sca-maven-plugin:25.4.0:translate
  ```

  To analyze your code this way, see the documentation included with the Fortify Maven Plugin. The following table describes where to find the documentation after you install the Fortify Maven Plugin.

Package type	Documentation location
Binary	`<root_dir>/docs/index.html`
Source	`<root_dir>/sca-maven-plugin/target/site/index.html`

1.7.3. Integrating with Bazel

To integrate with Bazel builds, OpenText SAST translates the source files as they are compiled. Therefore, a prerequisite for Bazel builds is that the Bazel build runs successfully. See Build tools for supported Bazel versions.

To integrate with Bazel, navigate to the Bazel workspace directory, and then run sourceanalyzer with the Bazel target you want to build. You can specify other sourceanalyzer options for the translation as follows:

```
sourceanalyzer -b <build_id> <sca_options> bazel build <target>
```

Translate a project and exclude a file from the translation:

```
sourceanalyzer -b MyProjectC -exclude C:\test\MY-JAVA-APP\src\proj\content.py bazel build //projc:my-python-prj
```

1.7.3.1. Java Bazel integration examples

Translate a project for a specific target:

```
sourceanalyzer -b MyProjectA bazel build //proja:my-prj
```

Translate target `abc` in package `proja/abc`:

```
sourceanalyzer -b MyProjectA bazel build //proja/abc
```

or

```
sourceanalyzer -b MyProjectA bazel build //proja/abc:abc
```

Translate all targets in the package `proja/abc`:

```
sourceanalyzer -b MyProjectA bazel build //proja/abc:all
```

Translate all targets within the `projb/` directory:

```
sourceanalyzer -b MyProjectB bazel build //projb/...
```

Specify a specific JDK version for the translation:

```
sourceanalyzer -b MyProjectC -jdk 17 bazel build //projc:my-java-prj
```

Translate a project and exclude a file from the translation:

```
sourceanalyzer -b MyProjectC -exclude C:\test\MY-JAVA-APP\src\main\java\com\example\HelpContent.java bazel build //projc:my-java-prj
```

OpenText SAST Bazel integration does not support multiple targets and related actions such as excluding targets.

1.7.4. Integrating with Ant

You can translate Java source files for projects that use an Ant build file. You can apply this integration on the command line without modifying the Ant `build.xml` file. When the build runs, OpenText SAST intercepts all `javac` task invocations and translates the Java source files as they are compiled. Make sure that you pass any properties to Ant by adding them to the ANT_OPTS environment variable. Do not include them in the sourceanalyzer command.

Note

You must translate any JSP files, configuration files, or any other non-Java source files that are part of the application in a separate step.

To use the Ant integration, make sure that the `sourceanalyzer` executable is in the PATH environment variable.

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

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

For example, to translate a Java project and exclude a file from the translation:

```
sourceanalyzer -b MyProjectA -logfile MyProjectA.log -exclude src/module-info.java ant
```

1.7.5. Manual Java and Kotlin translation syntax

To translate Java or Kotlin code manually, include all source file on the command line and provide all of the dependencies via .jar files, .class files, or source files. Failing to provide dependencies may lead to suboptimal scan results.

Kotlin to Java interoperability does not support Kotlin files provided by the `–sourcepath` option. For more information about the `–sourcepath` option, see Java Command-Line Options.

The basic command-line syntax to translate Java or Kotlin code is shown in the following example:

```
sourceanalyzer -b <build_id> -cp <classpath> [<translation_options>] <files> | <file_specifiers>
```

where:

- *<translation_options>* are options passed to the compiler.
- `-cp` *<classpath>* specifies the class path to use for resolving Java and Kotlin symbols.

  Include all JAR dependencies normally used to build the project. Separate multiple paths with semicolons (Windows) or colons (non-Windows).

  Similar to javac, OpenText SAST loads classes in the order they appear in the class path. If there are multiple classes with the same name in the list, OpenText SAST uses the first loaded class. In the following example, if both `A.jar` and `B.jar` include a class called `MyData.class`, OpenText SAST uses the `MyData.class` from `A.jar`.

  ```
  sourceanalyzer -cp A.jar:B.jar myfile.java
  ```

  OpenText strongly recommends that you avoid using duplicate classes with the `-cp` option.

  OpenText SAST loads JAR files in the following order:

  1. From the `-cp` option
  2. From `jre/lib`
  3. From *<sast_install_dir>*`/Core/default_jars`

  This enables you to override a library class by including the similarly-named class in a JAR specified with the `-cp` option.

For descriptions of all the available Java-specific command-line options, see "Java/J2EE Command-Line Options".

With Java code, OpenText SAST can additionally emulate the compiler to help integrate more easily into custom build scripts.

To have OpenText SAST emulate the compiler, type:

```
sourceanalyzer -b <build_id> javac [<translation_options>]
```

1.7.5.1. Java, Kotlin and JSP command-line options

The following table describes the Java command-line options (for Java SE and Jakarta EE).

Java, Kotlin or Jakarta EE option	Description
`-appserver weblogic | websphere`	Specifies the application server to process JSP files. **Equivalent property name:** `com.fortify.sca.AppServer`
`-appserver-home <dir>`	Specifies the application server's home. • For Oracle® WebLogic®, this is the path to the directory that contains the `server/lib` directory. • For IBM® WebSphere®, this is the path to the directory that contains the `JspBatchCompiler` script. **Equivalent property name:** `com.fortify.sca.AppServerHome`
`-appserver-version <version>`	Specifies the version of the application server. **Equivalent property name:** `com.fortify.sca.AppServerVersion`

`-cp <paths>` | `-classpath <paths>`	Specifies the class path used to resolve Java and Kotlin dependencies. The format is the same as javac: a semicolon- or colon-separated list of directories. You can use OpenText SAST file specifiers as shown in the following example:
	`-cp "build/classes:lib/*.jar"`
	For information about file specifiers, see Specifying files and directories.
	Equivalent property name:
	`com.fortify.sca.JavaClasspath`

`-extdirs <dirs>`	Similar to the javac `extdirs` option, accepts a semicolon- or colon-separated list of directories. Any JAR files found in these directories are included implicitly on the class path. **Equivalent property name:** `com.fortify.sca.JavaExtdirs`
`-java-build-dir <dirs>`	Specifies one or more directories that contain compiled Java sources.
`-source <version>` | `-jdk <version>`	Indicates the Java™ Development Kit (JDK) version for which the Java or Kotlin code is written. For supported versions, see Supported languages. The default is version 11. **Equivalent property name:** `com.fortify.sca.JdkVersion`
`-custom-jdk-dir`	Specifies a directory that contains a JDK. Use this option to specify a version that is not included in the OpenText SAST installation (*<sast_install_dir>*`/Core/bootcp/`). For supported versions, see Supported languages. **Equivalent property name:** `com.fortify.sca.CustomJdkDir`
`-show-unresolved-symbols`	Displays any unresolved types, fields, and functions referenced in translated Java source files at the end of the translation. It lists only field and function references for which the receiver type is a resolved Java type. Displays each class, field, and function with the source information of the first translated occurrence in the code. This information is also written in the log file. **Equivalent property name:** `com.fortify.sca.ShowUnresolvedSymbols`
`-sourcepath <dirs>`	Specifies a semicolon- or colon-separated list of directories that contain source code that is not included in the scan but is used for name resolution. The source path is similar to class path, except it uses source files instead of class files for resolution. Only source files that are referenced by the target file list are translated. **Equivalent property name:** `com.fortify.sca.JavaSourcePath`
`-jvm-default <mode>`	Specifies the generation of the `DefaultImpls` class for methods with bodies in Kotlin interfaces. The valid values for *<mode>* are: • `disable`—Specifies to generate the `DefaultImpls` class for each interface that contains methods with bodies. • `all`—Specifies to generate the `DefaultImpls` class if an interface is annotated with `@JvmDefaultWithCompatibility`. • `all-compatibility`—Specifies to generate the `DefaultImpls` class unless an interface is annotated with `@JvmDefaultWithoutCompatibility`. **Equivalent property name:** `com.fortify.sca.KotlinJvmDefault`

Java and Kotlin Properties

1.7.5.2. Java command-line examples

To translate a single file named `MyServlet.java` with `javaee.jar` as the class path, type:

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

To translate all `.java` files in the `src` directory using all JAR files in the `lib` directory as a class path, type:

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

To translate and compile the `MyCode.java` file with the javac compiler, type:

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

1.7.5.3. Kotlin command-line examples

To translate a single file named `MyKotlin.kt` with `A.jar` as the class path, type:

```
sourceanalyzer -b MyProject -cp lib/A.jar MyKotlin.kt
```

To translate all `.kt` files in the `src` directory using all JAR files in the `lib` directory as a class path, type:

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

To translate a gradle project using gradlew, type:

```
sourceanalyzer -b MyProject gradlew clean assemble
```

To translate all files in the `src` directory using Java dependencies from `src/java` and all JAR files in the `lib` directory and subdirectories as a class path, type:

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

1.7.6. Analyzing Kotlin scripts

OpenText SAST supports translation of Kotlin scripts excluding experimental script customization. Script customization includes adding external properties, providing static or dynamic dependencies, and so on. Script definitions (templates) are used to create custom scripts and the template is applied to the script based on the `*.kts` extension. OpenText SAST translates `*.kts` files but does not apply these templates.

1.7.7. Kotlin and Java translation interoperability

If your project contains Kotlin code that refers to Java code, you can provide Java files to the translator the same way as Kotlin files that refers to another Kotlin file. You can provide them as part of the translated project source or as `–sourcepath` parameters.

If your project contains Java code that refers to Kotlin code, make sure that the Java and Kotlin code are translated in the same OpenText SAST instance so that the Java references to Kotlin elements are resolved correctly. Kotlin to Java interoperability does not support Kotlin files provided by the `–sourcepath` option. For more information about the `–sourcepath` option, see Java, Kotlin and JSP command-line options.

1.7.8. Handling Java warnings

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

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

Java translation warnings

You might see the following warnings in the Java code translation.

Warning	Resolution
Unable to resolve type... Unable to resolve function... Unable to resolve field... Unable to locate import... Unable to resolve symbol...	These warnings are typically caused by missing resources. For example, some of the `.jar` and `.class` files required to build the application might not have been specified. To resolve these warnings, make sure that you include all the required files that your application uses.
Multiple definitions found for class...	This warning is typically caused by duplicate classes in the Java files. To resolve these warnings, make sure that the source files displayed in the warning are not duplicates of the same file included several times in the sources to translate (for example if it contains two versions of the same project). If a duplicate exists, remove one of them from the files to translate. Then OpenText SAST can determine which version of the class to use. This warning can also indicate that classes are missing. To resolve this, make sure to add all required JAR files to the classpath.

1.7.9. Analyzing Jakarta EE (Java EE) applications

To translate Jakarta EE applications, OpenText SAST processes Java source files and Jakarta EE components such as JSP files, deployment descriptors, and configuration files. While you can process all the pertinent files in a Jakarta EE application in one step, your project might require that you break the procedure into its components for integration in a build process or to meet the needs of various stakeholders in your organization.

This section contains the following topics:

1.7.9.1. Translating Java files

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

1.7.9.2. Translating JSP projects, configuration files, and deployment descriptors

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

For example:

```
sourceanalyzer -b MyJavaApp "/**/*.jsp" "/**/*.xml"
```

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

1.7.9.3. Jakarta EE (Java EE) translation warnings

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

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

This warning indicates that your web application is not deployed in the standard WAR directory format or does not contain the full set of required libraries. To resolve the warning, make sure that your web application is in an exploded WAR directory format with the correct `WEB-INF/lib` and `WEB-INF/classes` directories that contain all the `.jar` and `.class` files required for your application. Also verify that you have all the `TLD` files for all your tags and the corresponding JAR files with their tag implementations.

1.7.10. Analyzing Java bytecode

OpenText recommends that you do not translate Java bytecode and JSP/Java code in the same call to `sourceanalyzer`. Use multiple invocations of `sourceanalyzer` with the same build ID to translate a project that contains both bytecode and JSP/Java code.

To translate bytecode:

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

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

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

   This specifies how OpenText SAST processes `.class` and `.jar` files.

2. Do one of the following:
   - Request that OpenText SAST decompile the bytecode classes to regular Java files for inclusion in the translation.

     Add the following property to the `fortify-sca.properties` file:

     ```
     com.fortify.sca.DecompileBytecode=true
     ```

     or include this property on the command line for the translation phase with the `-D` option:

     ```
     sourceanalyzer -b MyProject -Dcom.fortify.sca.DecompileBytecode=true -cp "lib/*.jar" "src/**/*.class"
     ```

   - Request that OpenText SAST translate bytecode without decompilation.

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

     Include bytecode in the translation phase by specifying the Java bytecode files that you want to translate. For best performance, specify only the `.jar` or `.class` files that require scanning. In the following example, the `.class` files are translated:

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

1.7.11. Troubleshooting JSP translation and analysis issues

The following sections provide troubleshooting information for JSP analysis.

Unable to translate some JSPs

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

> Failed to translate the following jsps into analysis model. Please see the log file for any errors from the jsp parser and the user manual for hints on f ixing those
> *<list_of_jsp_files>*

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

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

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

1. Open the OpenText SAST log file in an editor.
2. Search for the following strings:
    - `Jsp parser stdout:`
    - `Jsp parser stderr:`

The JSP parser generates these errors. Resolve the errors and rerun OpenText SAST.

For more information about how to analyze Jakarta EE applications, see Translating Jakarta EE (Java EE) applications.

Increased issues count in JSP-related categories

If the analysis results contain a considerable increase in the number of vulnerabilities in JSP-related categories such as cross-site scripting compared with earlier OpenText SAST versions, you can specify the `-legacy-jsp-dataflow` option in the analysis phase (with the `-scan` option). This option enables additional filtering on JSP-related dataflow to reduce the number of spurious false positives detected.

The equivalent property for this option that you can specify in the `fortify-sca.properties` file is `com.fortify.sca.jsp.LegacyDataflow`.

1.8. Analyzing Android projects

This section describes how to translate Java source code for Android applications. You can use OpenText SAST to scan the code with Gradle from either:

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

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

Note

You can also scan Android code directly from Android Studio with the Fortify Analysis Plugin for IntelliJ IDEA and Android Studio. For more information, see the *OpenText™ Fortify Analysis Plugin for IntelliJ IDEA and Android Studio User Guide*.

This section contains the following topics:

1.8.1. Android project translation prerequisites

The following are the prerequisites for translating Android projects:

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

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

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

1.8.2. Android code analysis command-line syntax

Use gradlew to scan Android projects, which is similar to using Gradle except that you use the Gradle Wrapper. For information about how to translate your Android project using the Gradle Wrapper, see Gradle Integration.

1.8.3. Filtering issues detected in Android layout files

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

OpenText recommends that Issues reported in any layout file be included in your standard audit so you can carefully determine if any of them are false positives. After you identify issues in layout files that you are not interested in, you can filter them out as described in Optimizing results. You can filter out the issues based on the Instance ID.

1.9. Analyzing Visual Studio projects

OpenText SAST provides a build integration to support translation of the following Visual Studio project types:

- C/C++ projects
- C# projects that target .NET Framework and .NET Core
- ASP.NET applications that target ASP.NET framework and ASP.NET Core
- Xamarin applications that target Android™ and iOS platforms

For a list of supported versions of relevant programming languages and frameworks, as well as Visual Studio and MSBuild versions, see Supported languages and Supported build tools.

This section contains the following topics:

1.9.1. Visual Studio project translation prerequisites

OpenText recommends that each project you translate is complete and that you perform the translation in an environment where you can build it without errors. For a list of software environment requirements, see Software requirements. A complete project contains the following:

- All necessary source code files (C/C++, C#, or VB.NET).
- All required reference libraries.

  This includes those from relevant frameworks, NuGet packages, and third-party libraries.
- For C/C++ projects, include all necessary header files that do not belong to the Visual Studio or MSBuild installation.
- For ASP.NET and ASP.NET Core projects, include all the necessary ASP.NET page files.

  The supported ASP.NET page types are ASAX, ASCX, ASHX, ASMX, ASPX, AXML, BAML, CSHTML, Master, RAZOR, VBHTML, and XAML.

1.9.2. Visual Studio Project command-line syntax

The basic syntax to translate a Visual Studio solution or project is to specify the corresponding build option for your project as part of the OpenText SAST translation command. This starts a build integration that analyzes your solution and project files and automatically executes the appropriate translation steps.

Important

To ensure that the build integration correctly pulls in all of the appropriate project dependencies and resources, you must run the OpenText SAST command from a command prompt with access to your build environment configuration. OpenText strongly recommends you run this command from the Developer Command Prompt for Visual Studio to ensure an optimal environment for the translation.

In the following examples, OpenText SAST translates all the projects contained in the Visual Studio solution `Sample.sln`. You can also translate one or more specific projects by providing a semicolon-separated list of projects.

By default, test projects are excluded from the translation. Projects in your solution that reference NUnit, xunit, or MSTest are considered a test project. To include test projects in the translation, add the MSBuild option `/p:ScaForceTranslateTestProjects=True` to your sourceanalyzer command.

- For a .NET 6.0 or later solution on Windows or Linux, use the following commands to translate the solution:
  1. Optionally, run the following command to remove any intermediate files from previous project builds:

     ```
     dotnet clean Sample.sln
     ```

  2. Optionally, run the following command to ensure that all required reference libraries are downloaded and installed in the project. Run this command from the top-level folder of the project:

     ```
     dotnet restore Sample.sln
     ```

  3. Run one of the following OpenText SAST commands depending on how your project build is implemented. You can include any additional build parameters in this command:

     ```
     sourceanalyzer –b MyProject dotnet msbuild Sample.sln
     ```

     or

     ```
     sourceanalyzer –b MyProject dotnet build Sample.sln
     ```

- For a C, C++, and .NET Framework solution (4.8.x or earlier) on Windows, use the following command to translate the solution:

  ```
  sourceanalyzer –b MyProject msbuild /t:rebuild [<msbuild_options>] Sample.sln
  ```

  Note

  If you run OpenText SAST from a Windows Command Prompt instead of the Visual Studio Developer Command Prompt, you must set up the environment and make sure the path to the MSBuild executable required to build your project is included in the PATH environment variable.

After the translation is complete, perform the analysis phase and save the results in an FPR file as shown in the following example:

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

1.9.3. Handling special cases for translating Visual Studio projects

This section contains the following topics:

1.9.3.1. Running translation from a script

To perform the translation in a non-interactive mode such as with a script, establish an optimal environment for translation by executing the following command before you run the OpenText SAST translation:

```
cmd.exe /k <vs_install_dir>/Common7/Tools/VSDevCmd.bat
```

where *<vs_install_dir>* is the directory where you installed Visual Studio.

1.9.3.2. Translating plain .NET and ASP.NET projects

You can translate plain .NET and ASP.NET projects from the Windows Command Prompt as well as from a Visual Studio environment. When you translate from the Windows Command Prompt, make sure the path to the MSBuild executable required to build your project is included in the PATH environment variable.

1.9.3.3. Translating C/C++ and Xamarin projects

You must translate C/C++ and Xamarin projects either from a Developer Command Prompt for Visual Studio or from the Fortify Extension for Visual Studio.

Note

For Xamarin projects, there is no need to use a custom rule for the Xamarin.Android API if a rule for the corresponding native Android API exists in the Fortify Secure Coding Rulepacks. Doing so can cause duplicate issues to be reported.

1.9.3.4. Translating projects with settings containing spaces

If your project is built with a configuration or other settings file that contains spaces, make sure to enclose the setting values in quotes. For example, to translate a Visual Studio solution `Sample.sln` that is built with configuration `My Configuration`, use the following command:

```
sourceanalyzer –b MySampleProj msbuild /t:rebuild
/p:Configuration="My Configuration" Sample.sln
```

1.9.3.5. Translating a single project from a Visual Studio solution

If your Visual Studio solution contains multiple projects, you have the option to translate a single project instead of the entire solution. Project files have a file name extension that ends with `proj` such as `.vcxproj` and `.csproj`. To translate a single project, specify the project file instead of the solution as the parameter for the MSBuild command.

The following example translates the `Sample.vcxproj` project file:

```
sourceanalyzer –b MySampleProj msbuild /t:rebuild Sample.vcxproj
```

1.9.3.6. Analyzing projects that build multiple executable files

If your Visual Studio or MSBuild project builds multiple executable files (such as files with the file name extension `*.exe`), OpenText strongly recommends that you run the analysis phase separately for each executable file to avoid false positive issues in the analysis results. To do this, use the `–binary-name` option when you run the analysis phase and specify the executable file name or .NET assembly name as the parameter.

The following example shows how to translate and analyze a Visual Studio solution `Sample.sln` that consists of two projects, Sample1 (a C++ project with no associated .NET assembly name) and Sample2 (a .NET project with .NET assembly name `Sample2`). Each project builds a separate executable file, `Sample1.exe` and `Sample2.exe`, respectively. The analysis results are saved in `Sample1.fpr` and `Sample2.fpr` files.

```
sourceanalyzer -b MySampleProj msbuild /t:rebuild Sample.sln
sourceanalyzer -b MySampleProj -scan -binary-name Sample1.exe -f Sample1.fpr
sourceanalyzer -b MySampleProj -scan -binary-name Sample2.exe -f Sample2.fpr
```

For more information about the `-binary-name` option, see Analysis Options.

1.9.4. Alternative ways to translate Visual Studio projects

This section describes alternative methods of translating Visual Studio projects.

This section contains the following topics:

1.9.4.1. Alternative translation options for Visual Studio solutions

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

- Use the Fortify Extension for Visual Studio

  The Fortify Extension for Visual Studio runs the translation and analysis (scan) phases together in one step.

- Append a devenv command to the OpenText SAST command

  The following command translates the Visual Studio solution `Sample.sln`:

  ```
  sourceanalyzer –b MySampleProj devenv Sample.sln /rebuild
  ```

  Note that OpenText SAST converts a devenv invocation to the equivalent MSBuild invocation, therefore in this case, the solution with this command is built by MSBuild instead of the devenv tool.

1.9.4.2. Translating without explicitly running OpenText SAST

You have the option to translate your Visual Studio project without invoking OpenText SAST directly. This requires the `Fortify.targets` file, which is located in `<sast_install_dir>\Core\private-bin\sca\MSBuildPlugin` in the `DotNet` and `Framework` directory. You can specify the file using an absolute or relative path in the build command line that builds your project. Use the path with the `Dotnet` or `Framework` directory depending on the build command you are using: `dotnet.exe` or `MSBuild.exe` respectively. For example:

```
dotnet.exe msbuild /t:rebuild /p:CustomAfterMicrosoftCommonTargets=<sast_install_dir>\Core\private-bin\sca\MSBuildPlugin\Dotnet\Fortify.targets
 Sample.sln
```

or

```
msbuild.exe /t:rebuild
/p:CustomAfterMicrosoftCommonTargets=<sast_install_dir>\Core\private-bin\sca\MSBuildPlugin\Framework\Fortify.targets Sample.sln
```

There are several environment variables that you can set to configure the translation of your project. Most of them have default values, which OpenText SAST uses if the variable is not set. These variables are listed in the following table.

Environment variable	**Description**	**Default value**
FORTIFY_MSBUILD_BUILDID	Specifies the OpenText SAST build ID for translation. Make sure that you set this value. This is equivalent to the OpenText SAST `-b` option.	None
FORTIFY_MSBUILD_DEBUG	Enables debug mode. This is equivalent to the OpenText SAST `–debug` option.	False
FORTIFY_MSBUILD_DEBUG_VERBOSE	Enables verbose debug mode. This is equivalent to the OpenText SAST `–debug-verbose` option. Takes precedence over FORTIFY_MSBUILD_DEBUG variable if both are set to true.	False
FORTIFY_MSBUILD_MEM	Specifies the memory requirements for translation in the form of the JVM `-Xmx` option. For example, `-Xmx2G`.	Automatic allocation based on physical memory available on the system
FORTIFY_MSBUILD_SCALOG	Specifies the location (absolute path) of the OpenText SAST log file. This is equivalent to the OpenText SAST `-logfile` option.	`%LOCALAPPDATA%/Fortify/sca/log/sca.log`

1.10. Analyzing JavaScript and TypeScript code

You can analyze JavaScript projects that contain JavaScript, TypeScript, JSX, and TSX source files, as well as JavaScript embedded in HTML files.

Some JavaScript frameworks are transpiled (source-to-source compilation) to plain JavaScript, which is generated code. Use the `-exclude` command-line option to exclude this type of code.

When you translate JavaScript and TypeScript code, make sure that you specify all source files together in one invocation. OpenText SAST does not support adding new files to the file list associated with the build ID on subsequent invocations.

OpenText SAST does not translate minified JavaScript (*.min.js).

Note

There are some types of minified JavaScript files that OpenText SAST cannot automatically detect for exclusion from the translation. Use the `-exclude` command-line option to exclude these files directly.

This section contains the following topics:

1.10.1. Translating pure JavaScript projects

The basic command-line syntax to translate JavaScript is:

```
sourceanalyzer –b <build_id> <js_file_or_dir>
```

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

1.10.2. Excluding dependencies

You can avoid translating specific dependencies by adding them to the appropriate property setting in the `fortify-sca.properties` file. Files specified in the following properties are *not* translated:

- `com.fortify.sca.skip.libraries.ES6`
- `com.fortify.sca.skip.libraries.jQuery`
- `com.fortify.sca.skip.libraries.javascript`
- `com.fortify.sca.skip.libraries.typescript`

Each property specifies a list of comma- or colon-separated file names (without path information).

The files specified in these properties apply to both local files and files on the internet. Suppose, for example, that the JavaScript code includes the following local file reference:

```
<script src="js/jquery-ui.js" type="text/javascript" charset="utf-8"></script>
```

By default, the `com.fortify.sca.skip.libraries.jQuery` property in the `fortify-sca.properties` file includes `jquery-us.js`, and therefore OpenText SAST does not translate the file shown in the previous example.

You can use regular expressions for the file names. Note that OpenText SAST automatically inserts the regular expression `'(-?\d+\.\d+\.\d+)?'` before `.min.js` or `.js` for each file name included in the `com.fortify.sca.skip.libraries.jQuery` property value.

Note

You can also exclude local files or entire directories with the `-exclude` command-line option. For more information about this option, see Translation Options.

To provide a thorough analysis, dependent files are included in the translation even if the dependency is in a language that is disabled with the `-disable-language` option. For more information about the option to disable languages, see Translation Options).

1.10.3. Excluding NPM Dependencies

By default, OpenText SAST translates only the NPM dependencies that are imported in the code. You can change this behavior with the following two properties:

- The `com.fortify.sca.follow.imports` property directs OpenText SAST to resolve all imported files and include them in the translation.

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

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

  This property is enabled by default to avoid translating dependencies that are not needed for the final project such as those only required for the build system.

1.10.4. NPM dependencies

By default, OpenText SAST does not report issues in NPM dependencies (files in the `node_modules` directory). This is configured with the `com.fortify.sca.exclude.node.modules` property, which is set to `true` by default.

Note

OpenText does not recommend using the -exclude option to exclude node modules if `com.fortify.sca.exclude.node.modules` is set to `true`, because it can change the quality of the results.

See Also

Examples of Excluding node_modules Dependencies

1.10.4.1. Examples of excluding NPM dependencies

The following examples illustrate three different scenarios for excluding NPM dependencies. All these examples use the following directory structure:

```
./
 RootProjectDir
 innerSrcDir
 node_modules
 innerProjectReferencedModule
   index.ts
 moduleNotReferencedByProject
  index.ts
 innerProject.ts (contains import from innerProjectReferencedModule)
 node_modules
 projectReferencedModule
 index.ts
 moduleNotReferencedByProject
 index.ts
 projectMain.ts (contains import from projectReferencedModule)
```

Example 1

This example shows the files are translated with `com.fortify.sca.exclude.unimported.node.modules` set to `false`. In this case, `com.fortify.sca.follow.imports` and `com.fortify.sca.exclude.unimported.node.modules` are both set to `true`.

```
sourceanalyzer RootProjectDir/ -Dcom.fortify.sca.exclude.node.modules=false
```

The following files are included in the translation for Example 1:

```
./RootProjectDir/innerSrcDir/innerProject.ts
./RootProjectDir/innerSrcDir/node_modules/innerProjectReferencedModule/index.ts
./RootProjectDir/projectMain.ts
./RootProjectDir/node_modules/projectReferencedModule/index.ts
```

Example 2

This example shows that in addition to modules referenced by the project, modules found during resolution but not referenced by the project are also included in the translation.

```
sourceanalyzer RootProjectDir/ -Dcom.fortify.sca.exclude.unimported.node.modules=false
```

The following files are included in the translation for Example 2:

```
./RootProjectDir/innerSrcDir/innerProject.ts
./RootProjectDir/innerSrcDir/node_modules/innerProjectReferencedModule/index.ts
./RootProjectDir/innerSrcDir/node_modules/moduleNotReferencedByProject/index.ts
./RootProjectDir/projectMain.ts
./RootProjectDir/node_modules/projectReferencedModule/index.ts
./RootProjectDir/node_modules/moduleNotReferencedByProject/index.ts
```

Example 3

This example shows use of the -exclude option to exclude all files under any `node_modules` directory. The `-exclude` option overrides resolution of modules based on the configuration of the `com.fortify.sca.follow.imports` and `com.fortify.sca.exclude.unimported.node.modules` properties.

```
sourceanalyzer RootProjectDir/ -exclude "**/node_modules/*.*"
```

The following files are included in the translation for Example 3:

```
./RootProjectDir/innerSrcDir/innerProject.ts
./RootProjectDir/projectMain.ts
```

1.10.5. Translating JavaScript projects with HTML files

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

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

When you set the `com.fortify.sca.EnableDOMModeling` property to true, this can decrease false negative reports of DOM-related attacks, such as DOM-related cross-site scripting issues.

Note

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

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

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

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

1.10.6. Including external JavaScript or HTML in the translation

To include external JavaScript or HTML files that are specified with the `src` attribute, you can specify which domains OpenText SAST can download and include in the translation phase. To do this, specify one or more domains with the `com.fortify.sca.JavaScript.src.domain.whitelist` property.

Note

You can also set this property globally in the `fortify-sca.properties` file.

For example, you might have the following statement in your HTML file:

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

If you are confident that the `xyzdomain.com` domain is a safe location from which to download files, then you can include it in the translation phase by adding the following property specification on the command line:

```
-Dcom.fortify.sca.JavaScript.src.domain.whitelist="xyzdomain.com/foo"
```

Note

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

To trust more than one domain, include each domain separated by the vertical bar character (`|`) as shown in the following example:

```
-Dcom.fortify.sca.JavaScript.src.domain.whitelist=
"xyzdomain.com/foo|abcdomain.com|123.456domain.com”
```

If you are using a proxy server, then you need to include the proxy server information on the command line as shown in the following example:

```
-Dhttp.proxyHost=example.proxy.com -Dhttp.proxyPort=8080
```

For a complete list of proxy server options, see the Networking Properties Java documentation.

1.11. Analyzing Python and Jupyter Notebooks

OpenText SAST translates Python applications, and processes files with the `.py` extension as Python source code. Files with the extension `.ipynb` are recognized as Jupyter Notebooks. OpenText SAST supports translation of Jupyter notebooks and the Django and Flask frameworks.

This section contains the following topics:

1.11.1. Integrating with Bazel

To integrate with Bazel builds, OpenText SAST translates the source files as they are compiled. Therefore, a prerequisite for Bazel builds is that the Bazel build runs successfully. See Build tools for supported Bazel versions.

To integrate with Bazel, navigate to the Bazel workspace directory, and then run sourceanalyzer with the Bazel target you want to build. You can specify other sourceanalyzer options for the translation as follows:

```
sourceanalyzer -b <build_id> <sca_options> bazel build <target>
```

Translate a project and exclude a file from the translation:

```
sourceanalyzer -b MyProjectC -exclude C:\test\MY-JAVA-APP\src\proj\content.py bazel build //projc:my-python-prj
```

1.11.1.1. Python Bazel integration examples

Translate a project for a specific target:

```
sourceanalyzer -b MyProjectA bazel build //proja:my-prj
```

Translate target `abc` in package `proja/abc`:

```
sourceanalyzer -b MyProjectA bazel build //proja/abc
```

or

```
sourceanalyzer -b MyProjectA bazel build //proja/abc:abc
```

Translate all targets in the package `proja/abc`:

```
sourceanalyzer -b MyProjectA bazel build //proja/abc:all
```

Translate all targets within the `projb/` directory:

```
sourceanalyzer -b MyProjectB bazel build //projb/...
```

Specify Python project dependencies for the translation:

```
sourceanalyzer -b MyProjectD -python-path /usr/local/lib/python3.6/ bazel build //projd:my-python-app
```

OpenText SAST Bazel integration does not support multiple targets and related actions such as excluding targets.

1.11.2. Python translation command-line syntax

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

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

Note

When you translate Python code, make sure that you specify all source files together in one invocation. OpenText SAST does not support adding new files to the file list associated with the build ID on subsequent invocations.

1.11.2.1. Python command-line options

The following table describes the Python options.

Python option	Description
`-python-version <version>`	Specifies the Python source code version to scan. The valid values for *<version>* are `2` and `3`. The default value is `3`. **Equivalent property name:** `com.fortify.sca.PythonVersion`
`-python-no-auto-root-calculation`	Disables the automatic calculation of a common root directory of all project source files to use for importing modules and packages. **Equivalent property name:** `com.fortify.sca.PythonNoAutoRootCalculation`
`-python-path <dirs>`	Specifies a semicolon-separated (Windows) or colon-separated (non-Windows) list of additional import directories. You can use the `-python-path` option to specify all paths used to import packages or modules. Include all paths to namespace package directories with this option. OpenText SAST sequentially searches the specified paths for each imported file and uses the first file encountered. **Equivalent property name:** `com.fortify.sca.PythonPath`
`-django-template-dirs <dirs>`	Specifies a semicolon-separated (Windows) or colon-separated (non-Windows) list of directories that contain Django templates. OpenText SAST sequentially searches the specified paths for each Django template file and uses the first template file encountered. **Equivalent property name:** `com.fortify.sca.DjangoTemplateDirs`
`-django-disable-autodiscover`	Specifies that OpenText SAST does not automatically discover Django templates. **Equivalent property name:** `com.fortify.sca.DjangoDisableAutodiscover`
`-jinja-template-dirs <dirs>`	Specifies a semicolon-separated (Windows) or colon-separated (non-Windows) list of directories that contain Jinja2 templates. OpenText SAST sequentially searches the specified paths for each Jinja2 template file and uses the first template file encountered. **Equivalent property name:** `com.fortify.sca.JinjaTemplateDirs`
`-disable-template-autodiscover`	Specifies that OpenText SAST does not automatically discover Django or Jinja2 templates. **Equivalent property name:** `com.fortify.sca.DisableTemplateAutodiscover`

Python Properties

1.11.2.2. Python command-line examples

Translate Python 3 code on Windows:

```
sourceanalyzer -b Python3Proj -python-path "C:\Python312\Lib;C:\Python312\Lib\site-packages" src/*.py
```

Translate Python 2 code on Windows:

```
sourceanalyzer -b MyPython2 -python-version 2 -python-path "C:\Python27\Lib;C:\Python27\Lib\site-packages" src/*.py
```

Translate Python 3 code on non-Windows:

```
sourceanalyzer -b Python3Proj -python-path /usr/lib/python3.12:/usr/local/lib/python3.12/site-packages src/*.py
```

Translate Python 2 code on non-Windows:

```
sourceanalyzer -b MyPython2 -python-version 2 -python-path /usr/lib/python2.7:/usr/local/lib/python2.7/site-packages src/*.py
```

1.11.3. Translating Python in a virtual environment

This section describes how to translate Python projects in virtual environments. Make sure that all project dependencies are installed in your virtual environment. To translate a Python project in a virtual environment, include the `-python-path` option to specify the project dependencies.

Python virtual environment example

To translate a Python project where the virtual environment name is `myenv` and the dependencies for the project are installed in the `myenv/lib/python<version>/site-packages` directory, type:

```
sourceanalyzer –b mybuild -python-path "myenv/lib/python<version>/site-packages/" myproject/
```

Conda environment example

To translate a Python project where the conda environment name is `myenv` and the project dependencies are installed in the `<conda_install_dir>/envs/myenv/lib/python<version>/site-packages` directory, type:

```
sourceanalyzer –b mybuild -python-path "<conda_install_dir>/envs/myenv/lib/python<version>/site-packages/" myproject/
```

1.11.4. Including imported modules and packages

To translate Python applications and prepare for a scan, OpenText SAST searches for any imported modules and packages used by the application. OpenText SAST does not respect the `PYTHONPATH` environment variable, which the Python runtime system uses to find imported modules and packages.

OpenText SAST searches for imported modules and packages using the list of directories in the following order:

1. The common root directory for all project source files. which OpenText SAST calculates automatically. For example, if there are two project directories `PrimaryDir/project1/*` and `PrimaryDir/project2/*`, the common root directory is `PrimaryDir`.

   To remove the common root directory as a search target for imported modules and packages, include the `-python-no-auto-root-calculation` option in the translation command.

2. The directories specified with the `-python-path` option.

   OpenText SAST includes a subset of modules from the standard Python library (module "builtins", all modules originally written in C, and others) in the translation. OpenText SAST first searches for a standard Python library module in the set included with OpenText SAST and then in the paths specified with the `-python-path` option. If your Python code imports any module that OpenText SAST cannot find, it produces a warning. To make sure that all modules of the standard Python library are found, add the path to your standard Python library in the `-python-path` list.

3. The current directory that contains the file being translated. For example, when OpenText SAST translates a `PrimaryDir/project1/a.py`, the directory `PrimaryDir/project1` is added as the last directory to search for imported modules and packages.

1.11.5. Including namespace packages

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

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

Include `/path_1;/path_2` with the `-python-path` option in the sourceanalyzer command line.

1.11.6. Translating Django and Flask

By default, OpenText SAST attempts to discover Django and Jinja2 templates in the project root directory. All detected Django and Jinja2 templates are automatically added to the translation. You can specify additional locations of Django or Jinja2 template files by adding the `-django-template-dirs` or the `-jinja-template-dirs` option to the sourceanalyzer command.

If you do not want OpenText SAST to automatically discover Django and Jinja2 templates, use the `-disable-template-autodiscover` option. If your project requires Django or Jinja2 templates, but the project is configured such that the templates are in an unexpected location, use the `-django-template-dirs` or `-jinja-template-dirs` option to specify the directories that contain the templates in addition to the `-disable-template-autodiscover` option as shown in the following non-Windows examples:

```
sourceanalyzer -b djangoProj -python-path /usr/lib/python3.12:/usr/local/lib/python3.12/site-packages djangoProj -django-template-dirs djangoProj/templatedir1:/djangoProj/dir2 -disable-template-autodiscover
```

```
sourceanalyzer -b flaskProj -python-path /usr/lib/python3.12:/usr/local/lib/python3.12/site-packages flaskProj -jinja-template-dirs flaskProj/templatedir1:/flaskProj/dir2 -disable-template-autodiscover
```

The following example translates a Python project that has a combination of Django and Jinja2 templates on Windows:

```
sourceanalyzer -b pythonProj -python-path "C:\Python312\Lib;C:\Python312\Lib\site-packages" flaskProj -django-template-dirs "C:\djangoProj\templatedir1;C:\djangoProj\dir2" -jinja-template-dirs "C:\flaskProj\templatedir1;C:\flaskProj\dir2" -disable-template-autodiscover
```

1.12. Analyzing C and C++ code

This section describes how to translate C and C++ code. OpenText SAST supports standard ANSI C and C++ and might not support all non-standard C++ constructs.

Important

This section describes how to translate C and C++ code that is *not* a part of a Visual Studio or MSBuild project. For instructions on how to translate Visual Studio or MSBuild projects, see Translating Visual Studio and MSBuild Projects.

This section contains the following topics:

1.12.1. C and C++ Code translation prerequisites

Make sure that you have any dependencies required to build the project available, including headers for third-party libraries. OpenText SAST translation does not require object files and static/dynamic library files.

1.12.2. Integrating with Make

To integrate OpenText SAST with make, run `sourceanalyzer` with Make for the build process. For example, if you build your project with the following build commands:

```
make clean
make
make install
```

You can simultaneously translate and compile the entire project with the following example commands:

```
make clean
sourceanalyzer -b MyProject make
make install
```

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

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

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

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

When you use the same build ID for each operation, OpenText SAST automatically combines each of the separately-translated files into a single translated project.

1.12.3. Integrating with CMake

On non-Windows systems, you can translate projects that are built with CMake by incorporating a JSON compilation database in the OpenText SAST command. This is only supported for Makefile and Ninja generators (see the CMake Reference Documentation for more information).

To integrate OpenText SAST with a CMake build:

1. Generate a `compile_commands.json` file for your CMake project.

   Add `-DCMAKE_EXPORT_COMPILE_COMMANDS=yes` to the `cmake` configure command. For example:

   ```
   cmake -G Ninja -DCMAKE_EXPORT_COMPILE_COMMANDS=yes
   ```

2. Include the JSON compilation database in your `sourceanalyzer` command as follows:

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

1.12.4. Integrating with Gradle

Gradle integration has a prerequisite on the C++ Application Plugin. Please make sure it is added to your Gradle file in one of the following formats:

```
apply plugin: 'cpp'
```

```
plugins {
 id 'cpp-application'
}
```

Gradle integration is as simple as prepending the Gradle or gradlew command line with the `sourceanalyzer` command as follows:

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

For more detailed guides, see the Java and Kotlin integration: Using Gradle integration

1.12.5. Manual C and C++ translation syntax

Command-line options passed to the compiler affect preprocessor execution and can enable or disable language features and extensions. For OpenText SAST to interpret your source code in the same way as the compiler, the translation phase for C/C++ source code requires the complete compiler command line. Prefix your original compiler command with the `sourceanalyzer` command and options.

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

```
sourceanalyzer -b <build_id> [<sca_options>] <compiler> [<compiler_options>] <file>.c
```

where:

- *<sca_options>* are options passed to OpenText SAST.
- *<compiler>* is the name of the C/C++ compiler you use, such as `gcc`, `g++`, or `cl`. See Supported languages for a list of supported C/C++ compilers.
- *<compiler_options>* are options passed to the C/C++ compiler.
- *<file>.c* must be in ASCII or UTF-8 encoding.

Note

All OpenText SAST options must precede the compiler options.

The compiler command must successfully complete when executed on its own. If the compiler command fails, then the OpenText SAST command prefixed to the compiler command also fails.

For example, if you compile a file with the following command:

```
gcc -I. -o hello.o -c helloworld.c
```

then you can translate this file with the following command:

```
sourceanalyzer -b MyProject gcc -I. -o hello.o -c helloworld.c
```

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

1.12.6. Scanning pre-processed C and C++ code

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

1.12.7. C/C++ Precompiled Header Files

Some C/C++ compilers support Precompiled Header Files, which can improve compilation performance. Some compilers' implementations of this feature have subtle side-effects. When the feature is enabled, the compiler might accept erroneous source code without warnings or errors. This can result in a discrepancy where OpenText SAST reports translation errors even when your compiler does not.

If you use your compiler's Precompiled Header feature, disable Precompiled Headers, and then perform a full build to make sure that your source code compiles cleanly.

1.13. Analyzing iOS and Xcode projects

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

This section contains the following topics:

1.13.1. iOS project translation prerequisites

The following are the prerequisites for translating iOS projects:

- Objective-C++ projects must use the non-fragile Objective-C runtime (ABI version 2 or 3).
- Use Apple's `xcode-select` command-line tool to set your Xcode path. OpenText SAST uses the system global Xcode configuration to find the Xcode toolchain and headers.
- Make sure that all source files required for a successful Xcode build are provided.

  You can exclude files from the analysis using the `-exclude` option (see iOS Code Analysis Command-Line Syntax).

- Make sure that you have any dependencies required to build the project available.
- To translate Swift code, make sure that you have available all third-party modules, including CocoaPods. Bridging headers must also be available. However, Xcode usually generates them automatically during the build.
- If your project includes property list files in binary format, you must first convert them to XML format. You can do this with the Xcode `putil` command.

- To translate Objective-C projects, ensure that the headers for third-party libraries are available.
- To translate Watchkit® applications, make sure that you translate both the iPhone application target and the WatchKit extension target.

1.13.2. iOS code analysis command-line syntax

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

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

where `<compiler_options>` are the supported options that are passed to the Xcode compiler. You must include the `build` option with any `<compiler_options>`. The OpenText SAST Xcodebuild integration does not support the output format of alternate build commands such as `xcodebuild archive`.

Note

Xcodebuild compiles the source code when you run this command.

To exclude files from the analysis, use the `-exclude` option (see Translation Options). All source files that match the exclude specification are not translated, even if they are included in the Xcode build. The following is an example:

```
sourceanalyzer -b MyProject -exclude "**/TestFile.swift" xcodebuild clean build
```

If your application uses any property list files (for example, `<file>.plist`), translate these files with a separate `sourceanalyzer` command. Use the same build ID that you used to translate the project files. The following is an example:

```
sourceanalyzer -b MyProject <path_to_plist_files>
```

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

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

After the translation is complete, you can perform the analysis phase and save the results in an FPR file, as shown in the following example:

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

1.14. Analyzing PHP code

The syntax to translate a single PHP file named `MyPHP.php` is shown in the following example:

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

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

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

For more information about the `-php-source-root` option, see the description in PHP Command-Line Options.

When you translate PHP code, make sure that you specify all source files together in one invocation. OpenText SAST does not support adding new files to the file list associated with the build ID on subsequent invocations.

This section contains the following topics:

- PHP command-line options

1.14.1. PHP command-line options

The following table describes the PHP-specific command-line options.

PHP option	Description
`-php-source-root <path>`	Specifies an absolute path to the project root directory. The relative path name first expands from the current directory. If the file is not found, then the path expands from the specified PHP source root directory. **Equivalent property name:** `com.fortify.sca.PHPSourceRoot`
`-php-version <version>`	Specifies the PHP version. The default version is 8.2. For a list of valid versions, see Supported languages. **Equivalent property name:** `com.fortify.sca.PHPVersion`

PHP Properties

1.15. Analyzing Go code

This section describes how to translate Go code. OpenText SAST supports analysis of Go code on Windows, Linux, and macOS®.

This section contains the following topics:

1.15.1. Go command-line syntax

For the best results, your project must be compilable and you must have all required dependencies available.

The following entities are excluded from the translation (and the scan):

- Vendor folder
- All projects defined by any `go.mod` files in subfolders, except the project defined by the `go.mod` file under the %PROJECT_ROOT%
- All files with the `_test.go` suffix (unit tests)

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

```
sourceanalyzer -b <build_id> [-gopath <dir>] [-goroot <dir>] <files>
```

For best results, OpenText recommends that you use Go modules for all Go projects and translate the Go code one module at a time. Ensure that the values for the *<files>* parameter for the sourceanalyzer command are in the directory that contains the `go.mod` file. This is the same directory where you run the `go build` command to build the project. If the project consists of more than one module, you can run the sourceanalyzer command multiple times with the same *<build_id>* value to tie the translation results for all modules together.

Use of the GOPATH development mode for builds is still supported but be aware that this can cause problems if you are trying to compare two scans in tools such as Fortify Audit Workbench or Application Security. Without a `go.mod` file to define a fixed identifier path for the module, the Go language system identifies each module by its absolute path on the local file system. Therefore, two scans of the same module from different subdirectories or on different machines produce different module identifiers, which prevents matching issues from correlating properly across the two scans. The GOPATH development mode is deprecated for the Go compiler and SDK and will be removed in a future Go 1.xx release.

1.15.2. Go command-line options

The following table describes the command-line options that are specifically for translating Go code.

Go option	Description
`-gotags <go_build_tags>`	Specifies a comma-separated list of custom build tags for a Go project. This is equivalent to the `-tags` option for the `go` command. For more information, see Including Custom Go Build Tags. **Equivalent property name:** `com.fortify.sca.gotags`

`-gopath <dir>`	Specifies the value of the GOPATH environment variable to use for translating a Go project. If this option is not specified, then OpenText SAST uses the existing value of the GOPATH system environment variable.
	You must specify the gopath directory as an absolute path. The following examples show valid values for `<dir>`:
	`/home/projects/go_workspace/my_proj`
	`C:\projects\go_workspace\my_proj`
	The following example is an invalid value for `<dir>`:
	`go_workspace/my_proj`
	If this option and the GOPATH system environment variable is not set, then the gopath defaults to a subdirectory named `go` in the user's home directory (`$HOME/go` on Linux and `%USERPROFILE%\go` on Windows), unless that directory contains a Go distribution.
	When using modules, the GOPATH environment variable is not required to resolve package imports. However, GOPATH still determines the output directory to use when downloading missing module dependencies.
	Note
	OpenText SAST does not fully support older Go projects that rely solely on the GOPATH environment variable to resolve package imports.
	Equivalent property name:
	`com.fortify.sca.GOPATH`

`-goroot <dir>`	Specifies the location of the Go installation. If this option is not specified, the GOROOT system environment variable is used. If this option is not specified and the GOROOT system environment variable is not set, then OpenText SAST uses the Go compiler included in the OpenText SAST installation. **Equivalent property name:** `com.fortify.sca.GOROOT`
`-goproxy <url>`	Specifies one or more comma-separated proxy URLs. You can also specify `direct` or `off` (to disable network usage). If this option is not specified and the GOPROXY system environment variable is not set, then OpenText SAST uses `https://proxy.golang.org,direct`. **Equivalent property name:** `com.fortify.sca.GOPROXY`

Go properties

1.15.3. Including custom Go build tags

If your Go project includes files that require custom build tags, then you can include these build tags in the OpenText SAST translation using the `-gotags` option. For example:

```
sourceanalyzer -b MyProject -gotags release "src/**/*.go"
```

The OpenText SAST `-gotags` option does not allow you to override automatic build tags for the operating system, architecture, or Go version (for example, `//go:build linux`, `//go:build arm`, `//go:build go1.21`). To translate your Go project for a different operating system or architecture, set the appropriate cross-compile targets in the GOOS and GOARCH environment variables. To set a specific Go version, specify the path for the Go SDK version in the GOROOT environment variable or the `-goroot` option.

1.15.4. Resolving dependencies

OpenText SAST supports two dependency management systems built into Go:

- Modules

  To translate a Go project that uses modules, the project must include a `go.mod` file that specifies the required dependencies, and a corresponding `go.sum` file for verifying downloaded dependencies. Specify the directory that contains the `go.mod` file as the project root in the sourceanalyzer command.

  OpenText SAST downloads all required dependencies using the native Go toolchain. If access to the internet is restricted on the machine where you run OpenText SAST, then do one of the following:

  - If you are using an artifact management system such as Artifactory, set the GOPROXY environment variable or use the `-goproxy` option described in Go Command-Line Options.
  - Download all required dependencies using modules and vendoring.

    If you use manual vendoring, set the GOFLAGS environment variable to `-mod=vendor` before you start the translation.

- GOPATH dependency resolution

  If you are using a third-party dependency management system such as dep, you must download all dependencies before you start the translation.

  The GOPATH development mode identifies dependencies using the absolute path on the local file system, which can cause problems when correlating scans from different subdirectories or on different machines.

See Also

Go command-line syntax

1.16. Analyzing Dart and Flutter code

This section describes how to translate Dart and Flutter code. OpenText SAST supports analysis of Dart and Flutter code on Windows and Linux.

This section contains the following topics:

1.16.1. Dart and Flutter translation prerequisites

The following are the prerequisites for translating Dart and Flutter projects:

- Make sure that you have a supported Dart SDK (for Dart-only projects) and the Flutter SDK (for Flutter projects) installed on your system. See Supported languages for the supported Dart and Flutter SDK versions.
- Download the project dependencies by running one of the following commands:
  - For Flutter projects, use `flutter pub get`.
  - For Dart-only projects, use `dart pub get` .

  For example, to download the dependencies for a Flutter project that has the project root `myproject`, run the following commands:

```
cd myproject
flutter pub get
```

Important

If the project includes nested packages with different `pubspec.yaml` files, you must run `dart pub get` or `flutter pub get` for each package root.

Important

Make sure that the following are included in the project directory:

- The `pubspec.yaml` file, which specifies the dependencies
- The `.dart_tool` directory, which includes the `package_config.json` file automatically generated by the `pub` tool

1.16.2. Dart and Flutter command-line syntax

The basic command-line syntax to translate Dart and Flutter code is:

```
sourceanalyzer –b <build_id> <translation_options> <dirs>
sourceanalyzer –b <build_id> <translation_options> <files>
```

1.16.3. Dart and Flutter command-line examples

To translate a Dart or Flutter project with the `my_app` project root directory:

```
sourceanalyzer -b MyProject my_app/
```

To translate the `a_widget.dart` file in the `my_app` project root directory:

```
sourceanalyzer -b MyProject my_app/a_widget.dart
```

To translate all dart source files in the `my_dart_proj` directory:

```
sourceanalyzer -b MyProject "my_dart_proj/**/*.dart"
```

1.17. Analyzing Salesforce Apex and Visualforce code

This section contains the following topics:

1.17.1. Apex and Visualforce translation prerequisites

To translate Apex and Visualforce projects, make sure that all the source code to scan is available on the same machine where you have installed OpenText SAST.

To scan your custom Salesforce® app, download it to your local computer from your Salesforce organization (org) where you develop and deploy it. The downloaded version of your app consists of:

- Apex classes in files with the `.cls` extension
- Visualforce web pages in files with the `.page` extension
- Apex code files called database “trigger” functions in files with the `.trigger` extension
- Visualforce component files in files with the `.component` extension
- Objects in files with the `.object` extension

Use the Ant Migration Tool available on the Salesforce website to download your app from your org in the Salesforce cloud to your local computer. Make sure that the project manifest files are set up correctly for the specified target in your `build.xml` file. For example, the following `package.xml` manifest file provides OpenText SAST with all classes, custom objects, pages, and components.

```
<?xml version="1.0" encoding="UTF-8"?>
<Package xmlns=http://soap.sforce.com/2006/04/metadata>
 <types>
 <members>*</members>
 <name>ApexClass</name>
 </types>
 <types>
 <members>*</members>
 <name>ApexTrigger</name>
 </types>
 <types>
 <members>*</members>
 <name>ApexPage</name>
 </types>
 <types>
 <members>*</members>
 <name>ApexComponent</name>
 </types>
 <types>
 <members>*</members>
 <name>CustomObject</name>
 </types>
 <version>55.0</version>
</Package>
```

Configure the retrieve targets using the Ant Migration Tool documentation. If your organization uses any apps from the app exchange, make sure that these are downloaded as packaged targets.

1.17.2. Apex and Visualforce command-line syntax

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

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

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

Important

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

1.18. Analyzing ABAP code

ABAP code translation requires additional preparation steps to extract the code from the SAP® database and prepare it for scanning. See Importing the Transport Request for more information. This section assumes you have a basic understanding of SAP and ABAP.

This section contains the following topics:

1.18.1. About downloading source files

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

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

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

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

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

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

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

This final list of names is used with `READ REPORT` to get code out of the SAP system. This method reads 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. OpenText SAST translates and scans this set of files to produce an FPR file that you can open with Fortify Audit Workbench.

ABAP Properties

1.18.1.1. INCLUDE processing

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

1.18.2. Importing the transport request

To scan ABAP code, you need to import the Fortify ABAP Extractor transport request on your SAP Server. You can find the transport request in `<sast_install_dir>/Tools/SAP_Extractor.zip`.

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

- `K900<release_number>.<system_id>`
- `R900<release_number>.<system_id>`

These files make up the SAP transport request that you must import into your SAP system from outside your local Transport Domain. Have your SAP administrator or an individual authorized to install transport requests on the system import the transport request. These files contain a program, a transaction (YSCA), and the program user interface. After you import them into your system, you can extract your code from the SAP database and prepare it for OpenText SAST scanning.

Installation note

If you get the transport request import error: `Install release does not match the current version`, then the transport request installation has failed. See Software requirements for supported ABAP versions.

To try to resolve this issue, perform the following steps:

1. Re-run the transport request import.

   The Import Transport Request dialog box opens.

2. Select the **Options** tab.
3. Select the **Ignore Invalid Component Version** check box.
4. Complete the import procedure.

If this does not resolve the issue or if your system runs on an SAP version with a different table structure, OpenText recommends that you export your ABAP file structure using your own technology so that OpenText SAST can scan the ABAP code.

1.18.3. Adding OpenText SAST to your Favorites list

Adding OpenText SAST to your Favorites list is optional, but doing so can make it quicker to access and start OpenText SAST 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 OpenText SAST entry, make sure that the SAP server is running and you are in the SAP Easy Access area of your web-based client.

To add OpenText SAST to your Favorites list:

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

   The **SAP Menu** opens.

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

   The **Manual entry of a transaction** dialog box opens.

3. Type `YSCA` in the **Transaction Code** box.
4. Click the green check mark icon.

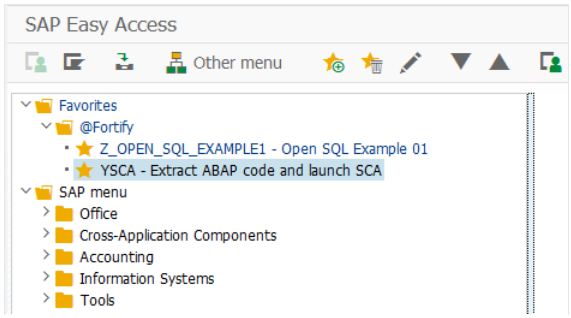
   The **Extract ABAP code and launch SCA** item appears in the **Favorites** list.

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

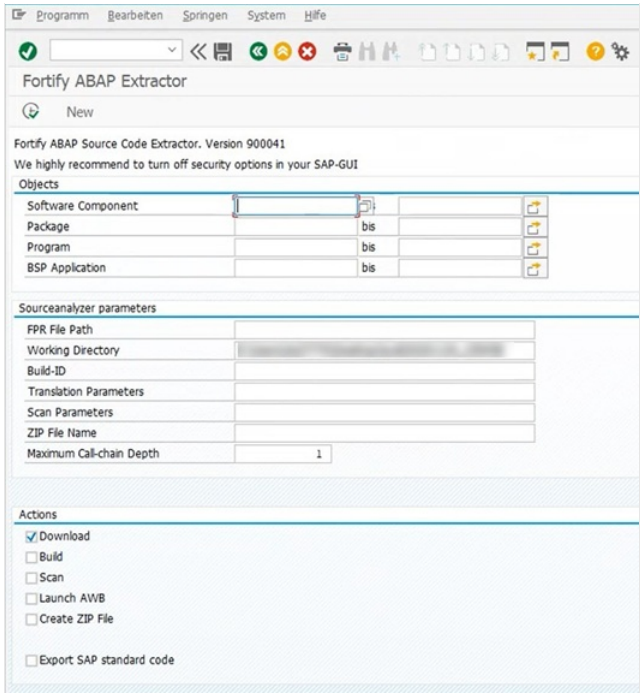
1.18.4. Running the Fortify ABAP Extractor

To run the Fortify ABAP Extractor:

1. Start the Fortify ABAP Extractor from the **Favorites** link, the transaction code, or manually start the Extractor object.

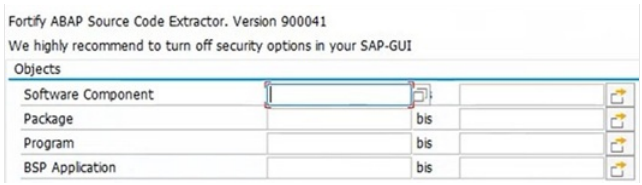


This opens the Fortify ABAP Extractor.



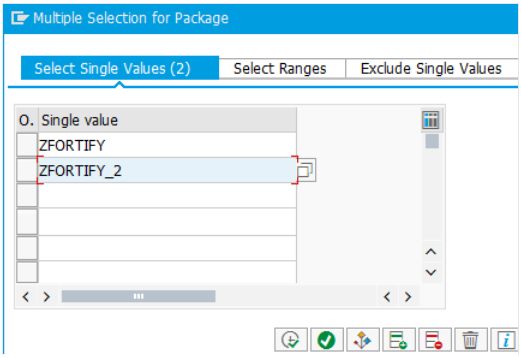
2. Select the code to download.

   Provide the start and end name for the range of software components, packages, programs, or BSP applications that you want to scan.



Note

You can specify multiple objects or ranges.



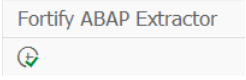
3. Provide the OpenText SAST-specific parameters described in the following table.

Field	Description
FPR File Path	(Optional) Type or select the directory where you want to store the scan results file (FPR). Include the name for the FPR file in the path name. You must provide the FPR file path to automatically scan the downloaded code on the same machine where you are running the extraction process.
Working Directory	Type or select the directory where you want to store the extracted source code.
Build-ID	(Optional) Type the build ID for the scan. OpenText SAST uses the build ID to identify the translated source code, which is necessary to scan the code. You must specify the build ID to automatically translate the downloaded code on the same machine where you are running the extraction process.
Translation Parameters	(Optional) Type any additional OpenText SAST command-line translation options. You must specify translation options to automatically translate the downloaded code on the same machine where you are running the extraction process or to customize the translation options.
Scan Parameters	(Optional) Type any OpenText SAST command-line scan options. You must specify scan options to scan the downloaded code automatically on the same machine where you are running the extraction process or to customize the scan options.
ZIP File Name	(Optional) Type a ZIP file name if you want your output in a compressed package.
Maximum Call-chain Depth	A global SAP-function F is not downloaded unless F was explicitly selected or unless F can be reached through a chain of function calls that start in explicitly-selected code and whose length is this number or less. OpenText recommends that you do not specify a value greater than 2 unless directed to do so by Customer Support.

4. Provide action information described in the following table.

Field	Description
Download	Select the **Download** check box to have OpenText SAST download the source code extracted from your SAP database.
Build	Select the **Build** check box to have OpenText SAST translate all downloaded ABAP code and store it using the specified build ID. This action requires that you have an installed version of OpenText SAST on the machine where you are running the Fortify ABAP Extractor. It is often easier to move the downloaded source code to a system where OpenText SAST is installed.
Scan	Select the **Scan** check box to have OpenText SAST run a scan of the specified build ID. This action requires that the translate (build) action was previously performed. This action requires that you have an installed version of OpenText SAST on the machine where you are running the Fortify ABAP Extractor. It is often easier to move the downloaded source code to a predefined OpenText SAST machine.
Launch AWB	Select the **Launch AWB** check box to start Fortify Audit Workbench and open the specified FPR file.
Create ZIP File	Select the **Create ZIP File** check box to compress the output. You can also manually compress the output after the source code is extracted from your SAP database.
Export SAP standard code	Select the **Export SAP standard code** check box to export SAP standard code as well as custom code.

5. Click **Execute**.



1.18.5. Uninstalling the Fortify ABAP Extractor

To uninstall the ABAP extractor:

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

1.19. Analyzing COBOL code

The COBOL translation runs on Windows systems only and supports modern COBOL dialects. Alternatively, you can use the legacy COBOL translation (see Using the Legacy COBOL Translation).

For a list of supported technologies for translating COBOL code, see Supported languages. OpenText SAST does not currently support custom rules for COBOL applications.

Note

To scan COBOL with OpenText SAST, you must have an OpenText SAST license file that specifically includes COBOL scanning capabilities. Contact Customer Support for more information about how to obtain the required license file.

This section contains the following topics:

- Preparing COBOL source and copybook files for translation
- COBOL command-line syntax
- Using Legacy COBOL translation

1.19.1. Preparing COBOL source and copybook files for translation

Before you can analyze a COBOL program, you must copy the following program components to the Windows system where you run OpenText SAST:

- COBOL source code

  OpenText strongly recommends that your COBOL source code files have extensions `.CBL`, `.cbl`, `.COB`, or `.cob`. If your source code files do not have extensions or have non-standard extensions, you must follow the instructions in Translating COBOL Source Files Without File Extensions and Translating COBOL Source Files with Arbitrary File Extensions.

- All copybook files that the COBOL source code uses

  This includes All SQL INCLUDE files that the COBOL source code references (a SQL INCLUDE file is technically a copybook file)

  Important

  The copybook files must have the extension `.CPY` or `.cpy`.

  If your COBOL source code contains:

  ```
  COPY FOO
  ```

  or

  ```
  EXEC SQL INCLUDE FOO END-EXEC
  ```

  then `FOO` is the name of a COBOL copybook and the corresponding copybook file has the name `FOO.CPY` or `FOO.cpy`.

OpenText recommends that you place your COBOL source code files in a directory called `sources` and your copybook files in a directory called `copybooks`. Create these directories at the same level.

1.19.2. COBOL command-line syntax

The basic syntax used to translate a single COBOL source code file is:

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

The basic syntax used to scan a translated COBOL program and save the analysis results in an FPR file is:

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

See Also

Specifying Files and Directories

1.19.2.1. Translating COBOL source files without file extensions

If you have COBOL source files (not copybook files) retrieved from a mainframe without `.COB` or `.CBL` file extensions (which is typical for COBOL file names), then you must include the following in the translation command line:

```
-noextension-type COBOL
```

The following example command translates COBOL source code without file extensions:

```
sourceanalyzer -b MyProject -noextension-type COBOL -copydirs copybooks sources
```

1.19.2.2. Translating COBOL source files with arbitrary file extensions

If you have COBOL source files with an arbitrary extension `.xyz`, then you must include the following in the translation command line:

```
-Dcom.fortify.sca.fileextensions.xyz=COBOL
```

You must also include the expression `*.xyz` in the file or directory specifier, if any (see Specifying Files and Directories).

1.19.2.3. COBOL command-line options

The following table describes the COBOL command-line options. To use legacy COBOL translation, see Legacy COBOL Translation Command-Line Options.

COBOL option	Description
`-copydirs <dirs>`	Specifies one or more semicolon-separated directories where OpenText SAST looks for copybook files. **Note** This option does not accept wildcards. **Equivalent property name:** `com.fortify.sca.CobolCopyDirs`
`-dialect <dialect>`	Specifies the COBOL dialect. The valid values for *<dialect>* are `COBOL390` and `MICROFOCUS`. The dialect value is case insensitive. The default value is `COBOL390`. **Equivalent property name:** com.fortify.sca.CobolDialect
`-checker-directives <directives>`	Specifies one or more semicolon-separated COBOL checker directives. **Note** This option is intended for advanced users of OpenText™ Server Express. **Equivalent property name:** `com.fortify.sca.CobolCheckerDirectives`

1.19.3. Using Legacy COBOL translation

Use the legacy COBOL translation if either of the following is true:

- You run OpenText SAST on a non-Windows operating system.

  For supported non-Windows platforms and architectures, see Platforms and architectures.

- Your COBOL dialect is different than what is supported by the default COBOL translation (see the `-dialect` option in COBOL Command-Line Options).

Prepare the COBOL source code and copybook files as described in Preparing COBOL Source and Copybook Files for Translation and use the command-line syntax described in COBOL Command-Line Syntax. Note that the legacy COBOL translation accepts copybook files with or without file extensions. If the copybook files have file extensions, use the `-copy-extensions` command-line option (see Legacy COBOL Translation Command-Line Options).

1.19.3.1. Legacy COBOL translation command-line options

The following table describes the command-line options for the legacy COBOL translation.

Legacy COBOL option	Description
`-cobol-legacy`	Specifies translation of COBOL code using legacy COBOL translation. This option is required to enable legacy COBOL translation. **Equivalent Property Name:** `com.fortify.sca.CobolLegacy`
`-copydirs <dirs>`	Specifies one or more semicolon- or colon-separated directories where OpenText SAST looks for copybook files. **Equivalent Property Name:** `com.fortify.sca.CobolCopyDirs`
`-copy-extensions <ext>`	Specifies one or more semicolon- or colon-separated copybook file extensions. **Equivalent Property Name:** `com.fortify.sca.CobolCopyExtensions`
`-fixed-format`	Specifies fixed-format COBOL to direct OpenText SAST to only look for source code between columns 8–72 in all lines of code. The default is free-format. IBM® Enterprise COBOL code is typically fixed-format. The following are indications that you might need the `-fixed-format` option: • The COBOL translation appears to hang indefinitely • OpenText SAST reports numerous parsing errors in the COBOL translation **Equivalent Property Name:** `com.fortify.sca.CobolFixedFormat`

1.20. Analyzing Ruby code

This section contains the following topics:

1.20.1. Ruby command-line syntax

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

```
sourceanalyzer –b <build_id> <file>
```

where `<file>` is the name of the Ruby file you want to scan. To include multiple Ruby files, separate them with a space, as shown in the following example:

```
sourceanalyzer –b <build_id> file1.rb file2.rb file3.rb
```

In addition to listing individual Ruby files, you can use the asterisk (*) wildcard to select all Ruby files in a specified directory. For example, to find all the Ruby files in a directory called `src`, use the following `sourceanalyzer` command:

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

Note

When you translate Ruby code, make sure that you specify all source files together in one invocation. OpenText SAST does not support adding new files to the file list associated with the build ID on subsequent invocations.

1.20.1.1. Ruby command-line options

The following table describes the Ruby translation options.

Ruby option	Description
`-ruby-path <dirs>`	Specifies one or more paths to directories that contain Ruby libraries (see Adding Libraries) **Equivalent property name:** `com.fortify.sca.RubyLibraryPaths`
`-rubygem-path <dirs>`	Specifies the path(s) to a RubyGems location (see Adding Gem Paths) **Equivalent property name:** `com.fortify.sca.RubyGemPaths`

Ruby Properties

1.20.2. Adding libraries

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

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

Separate multiple libraries with semicolons (Windows) or colons (non-Windows). The following is an example of the option on non-Windows system:

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

1.20.3. Adding gem paths

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

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

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

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

If you have multiple `gems` directories, separate them with semicolons (Windows) or colons (non-Windows) such as:

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

Note

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

1.21. Analyzing other languages and configurations

This section contains the following topics:

1.21.1. Analyzing Solidity code

The basic command-line syntax to translate and scan Solidity code is:

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

Importing dependencies

OpenText SAST translation only supports import statements for files with relative and absolute paths. Import statements for libraries is not supported.

Managing compiler versions

OpenText SAST downloads compilers that are referenced in the code with the pragma statement from the Solidity compiler repository. By default, OpenText SAST downloads Solidity compilers to `${flight.workdir}/solidity`.

If a file does not contain a pragma statement, then the default of `^0.8.0` is used. You can specify different default compiler version to use in the analysis by including the `flight.solidity.defaultCompilerVersion` property on the command line. The version you specify must exist in the Solidity compiler repository. For example:

```
sourceanalyzer -b MyProject ./
sourceanalyzer -b MyProject -scan -Dflight.solidity.defaultCompilerVersion=0.8.16 -f MyResults.fpr
```

If a proxy is required for the connection to download Solidity compilers, include the proxy information with `-Dhttps.proxyHost` and `-Dhttps.proxyPort`. For example:

```
sourceanalyzer -b MyProject ./
sourceanalyzer -b MyProject -scan -Dhttps.proxyHost=MyProxyHost -Dhttps.proxyPort=1234 -f MyResults.fpr
```

You can add `flight.solidity.defaultCompilerVersion` to the `fortify-sca.properties` file.

See Also

OpenText SAST Properties Files

1.21.2. Analyzing Flex and ActionScript

The basic command-line syntax to translate ActionScript is:

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

where:

`<libs>` is a semicolon-separated (Windows) or a colon-separated (non-Windows) list of library names to which you want to "link" and `<files>` are the files to translate.

1.21.2.1. Flex and ActionScript command-line options

Use the following command-line options to translate Flex files. You can also specify this information in the properties configuration file (`fortify-sca.properties`) as noted in each description.

Flex and ActionScript option	Description
`-flex-sdk-root <dir>`	Specifies the location of the root of a valid Flex SDK. This directory must contain a frameworks folder that contains a `flex-config.xml` file. It must also contain a `bin` folder that contains an MXMLC executable. **Equivalent property name:** `com.fortify.sca.FlexSdkRoot`
`-flex-libraries <libs>`	Specifies a semicolon-separated (Windows) or a colon-separated (non-Windows) list of library names to which you want to link. In most cases, this list includes `flex.swc`, `framework.swc`, and `playerglobal.swc` (usually found in `frameworks/libs/` in your Flex SDK root). **Note** You can specify SWC or SWF files as Flex libraries (SWZ is not currently supported). **Equivalent property name:** `com.fortify.sca.FlexLibraries`
`-flex-source-roots <dirs>`	Specifies a semicolon-separated (Windows) or a colon-separated (non-Windows) list of root directories where MXML sources are located. Normally, these contain a subfolder named `com`. For example, if the Flex source root specified is `foo/bar/src`, then `foo/bar/src/com/fortify/manager/util/Foo.mxml` is transformed into an object named `com.fortify.manager.util.Foo` (an object named `Foo` in the package `com.fortify.manager.util`). **Equivalent property name:** `com.fortify.sca.FlexSourceRoots`

Note

The `-flex-sdk-root` and `–flex-source-roots` options are primarily for MXML translation, and are optional if you are scanning pure ActionScript. Use `–flex-libraries` for to resolve all ActionScript linked libraries.

OpenText SAST translates MXML files into ActionScript, and then runs them through an ActionScript parser. The generated ActionScript is simple to analyze; not rigorously correct like the Flex runtime model. Consequently, you might get parse errors with MXML files. For instance, the XML parsing might fail, translation to ActionScript might fail, and the parsing of the resulting ActionScript might also fail. If you see any errors that do not have a clear connection to the original source code, notify Customer Support.

Flex and ActionScript Properties

1.21.2.2. ActionScript command-line examples

The following examples provide command-line syntax to translation ActionScript.

Example 1

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

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

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

Example 2

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

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

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

Example 3

In this example, the Flex SDK root and Flex libraries are specified in the properties file because typing the information for each sourceanalyzer run is time consuming and the data does not change often. Divide the application into two sections and store them in folders: a main section folder and a modules folder. Each folder contains a `src` folder where the paths start. File specifiers contain wild cards to pick up all the `.mxml` and `.as` files in both `src` folders. An `MXML` file in `main/src/com/foo/util/Foo.mxml` is translated as an ActionScript class named `Foo` in the package `com.foo.util`, for example, with the source roots specified here:

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

1.21.2.3. Handling resolution warnings

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

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

ActionScript warnings

You might receive a message similar to the following:

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

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

1.21.3. Analyzing ColdFusion code

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

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

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

1.21.3.1. ColdFusion command-line syntax

The basic command-line syntax to translate ColdFusion source code is:

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

where:

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

  For a description of how to use `<file_specifiers>`, see Specifying Files.

Note

OpenText SAST calculates the relative path to each CFML source file with the `-source-base-dir` directory as the starting point. OpenText SAST uses these relative paths when it generates instance IDs. If you move the entire application source tree to a different directory, the OpenText SAST- generated instance IDs remain the same if you specify an appropriate parameter for the `-source-base-dir` option.

1.21.3.2. ColdFusion (CFML) command-line options

The following table describes the CFML options.

ColdFusion option	Description
`-source-base-dir` *`<web_app_root_dir>`* *`<files>`*|*`<file_specifiers>`*	The web application root directory. **Equivalent property name:** `com.fortify.sca.SourceBaseDir`

ColdFusion (CFML) Properties

1.21.4. Analyzing SQL

On Windows (and Linux for .NET projects only), OpenText SAST assumes that files with the `.sql` extension are T-SQL rather than PL/SQL. If you have PL/SQL files with the `.sql` extension on Windows, you must configure OpenText SAST to treat them as PL/SQL.

The basic syntax to translate and scan PL/SQL is:

```
sourceanalyzer -b <build_id> -sql-language PL/SQL <files>
sourceanalyzer -b <build_id> -sql-language PL/SQL -scan -f <results>.fpr
```

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

The basic syntax to translate and scan T-SQL is:

```
sourceanalyzer -b <build_id> -sql-language TSQL <files>
sourceanalyzer -b <build_id> -scan -f <results>.fpr
```

SQL Properties

1.21.4.1. PL/SQL command-line example

The following example commands translate and scan two PL/SQL files:

```
sourceanalyzer -b MyProject -sql-language PL/SQL x.pks y.pks
sourceanalyzer -b MyProject -sql-language PL/SQL -scan -f MyResults.fpr
```

The following example commands translate and scan all PL/SQL files in the `sources` directory:

```
sourceanalyzer -b MyProject -sql-language PL/SQL "sources/**/*.pks"
sourceanalyzer -b MyProject -sql-language PL/SQL -scan -f MyResults.fpr
```

1.21.4.2. T-SQL command-line example

The following example translates two T-SQL files:

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

The following example translates all T-SQL files in the `sources` directory:

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

Note

This example assumes the `com.fortify.sca.fileextensions.sql` property in `fortify-sca.properties` is set to `TSQL`, which is the property's default value.

1.21.5. Analyzing Scala code

Translating Scala code requires the following:

- The Akka compiler plugin

  You can download this plugin from the Maven Central Repository.

- An Akka (formerly Lightbend) license file

  This license file is included with the OpenText SAST installation in the `<sast_install_dir>/plugins/lightbend` directory

For instructions on how set up the license and translate Scala code, see the Akka documentation Fortify SCA for Scala.

Important

If your project contains source code other than Scala, you must translate the Scala code using the Scala Fortify compiler plugin, and then translate other source code with sourceanalyzer using the same build ID before you run the analysis phase.

1.21.6. Analyzing Infrastructure as Code (IaC)

OpenText SAST understands Azure Resource Manager (ARM), Bicep, AWS CloudFormation, and HCL templates.

Note

HCL analysis support is specific to Terraform and supported cloud provider Infrastructure as Code (IaC) configurations.

For best results, make sure that the template files are deployment valid. The templates must not contain:

- Validation errors that are static and locally detectable (for example, type errors or references to undefined variables or functions).
- Predeployment errors that occur during template interpretation, but before any resources are deployed or modified (for example, invalid array indexing operations).
- Deployment errors that occur in the cloud (for example, dynamically referencing a non-existent resource).

OpenText recommends that AWS CloudFormation file name extensions are `.json`, `.yaml`, `.template`, or `.txt`. OpenText SAST supports other extensions only if they are not commonly used by other languages or file types (such as `.java` or `.html`).

By default, OpenText SAST translates files with the HCL extensions `.hcl` and `.tf`.

ARM translation command-line examples

Translate an ARM template:

```
sourceanalyzer -b MyProject ArmTemplate.json
```

Translate all ARM templates in a directory:

```
sourceanalyzer -b MyProject "src/**/*.json"
```

Bicep translation command-line examples

Translate a single Bicep template:

```
sourceanalyzer -b MyProject BicepTemplate.bicep
```

Translate all Bicep templates in a directory:

```
sourceanalyzer -b MyProject "src/**/*.bicep"
```

AWS CloudFormation translation command-line examples

Translate AWS CloudFormation templates that have different extensions:

```
sourceanalyzer -b MyProject CFTemplateA.template CFTemplateB.yaml CFTemplateC.json CFTemplateD.customext
```

Translate all AWS CloudFormation templates in a directory that have the `.template` extension:

```
sourceanalyzer -b MyProject "src/**/*.template"
```

Translate all AWS CloudFormation templates in a directory that have either the `.json` or `.yaml` extension:

```
sourceanalyzer -b MyProject "src/**/*.json" "src/**/*.yaml"
```

HCL translation command-line examples

Translate two HCL templates with different extensions:

```
sourceanalyzer -b MyProject HCLTemplateA.hcl HCLTemplateB.tf
```

Translate all HCL templates in a directory:

```
sourceanalyzer -b MyProject "src/**/*.tf" "src/**/*.hcl"
```

Translating JSON

Translating YAML

1.21.7. Analyzing JSON

By default, OpenText SAST translates files with the JSON extension `.json` as JSON. The following example translates a JSON file:

```
sourceanalyzer -b MyProject x.json
```

The following example translates all JSON files in the `sources` directory:

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

1.21.8. Analyzing YAML

By default, OpenText SAST translates files with the YAML extensions `.yaml` and `.yml`. The following example translates two YAML files with different file extensions:

```
sourceanalyzer -b MyProject x.yaml y.yml
```

The following example translates all YAML files in the `sources` directory:

```
sourceanalyzer -b MyProject "sources/**/*.yaml" "sources/**/*.yml"
```

1.21.9. Analyzing Dockerfiles

By default, OpenText SAST recognizes the files as Dockerfiles if they are named in one of the following formats: `Dockerfile*`, `dockerfile*`, `*.Dockerfile`, and `*.dockerfile`.

Note

You can modify the file name extension used to detect Dockerfiles using the `com.fortify.sca.fileextensions` property. See Translation and Analysis Phase Properties.

OpenText SAST accepts the following escape characters in Dockerfiles: backslash (`\`) and backquote (`` ` ``). If the escape character is not set in the Dockerfile, then OpenText SAST assumes that the backslash is the escape character.

The syntax to translate a directory that contains Dockerfiles is shown in the following example:

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

If the Dockerfile is malformed, OpenText SAST writes an error to the log file to indicate that the file cannot be parsed and skips the analysis of the Dockerfile. The following is an example of the error written to the log:

```
Unable to parse dockerfile ProjA.Dockerfile, error on Line 1:20: mismatched input '\n' expecting {LINE_EXTEND, WHITESPACE}

Unable to parse config file C:/Users/jsmith/MyProj/docker/dockerfile/ProjA.Dockerfile
```

1.21.10. Analyzing ASP/VBScript virtual roots

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

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

The following example shows the ASP/VBScript code for an application that uses *virtual* includes:

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

For this example, the previous ASP code refers to the file in the following physical location:

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

The real directory replaces the virtual directory name `Include` in this example.

Accommodating virtual roots

To provide the mapping of each virtual directory to OpenText SAST, you must set the `com.fortify.sca.ASPVirtualRoots.name_of_virtual_directory` property in your OpenText SAST command-line invocation as shown in the following example:

```
sourceanalyzer -Dcom.fortify.sca.ASPVirtualRoots.<virtual_directory>=<full_path_to_corresponding_physical_directory>
```

Note

On Windows, if the physical path includes spaces, you must enclose the property setting in quotes: `sourceanalyzer "-Dcom.fortify.sca.ASPVirtualRoots.<virtual_directory>=<full_path_to_corresponding_physical_directory>"`

To expand on the example in the previous section, pass the following property value to OpenText SAST:

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

This maps `Include` to `C:\WebServer\CustomerOne\inc` and `Library` to `C:\WebServer\CustomerTwo\Stuff`.

When OpenText SAST encounters the `#include` directive:

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

OpenText SAST determines if the project contains a physical directory named `Include`. If there is no such physical directory, OpenText SAST looks through its runtime properties and finds the `-Dcom.fortify.sca.ASPVirtualRoots.Include=`

`"C:\WebServer\CustomerOne\inc"` setting. OpenText SAST then looks for this file: `C:\WebServer\CustomerOne\inc\Task1\foo.inc`.

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

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

Note

The previous version of the ASPVirtualRoot property is still valid. You can use it on the OpenText SAST command line as follows:

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

`C:\WebServer\CustomerOne\inc`

This prompts OpenText SAST to search through the listed directories in the order specified when it resolves a virtual include directive.

Using virtual roots example

You have a file as follows:

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

To specify this file, use the following include:

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

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

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

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

The sequence to specify virtual roots is as follows:

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

1.21.11. Classic ASP command-line example

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

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

1.21.12. VBScript command-line example

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

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

1.22. Analyzing Library code

Library code refers to reusable software components or modules that are designed to be integrated into other applications. Unlike application code, which contains the business logic and entry points of a specific program, library code is typically:

- Generic and reusable across multiple projects
- Lacks a main entry point (e.g., main() method)
- Provides functionality that other applications consume (e.g., utility classes, frameworks, SDKs)

As library code is intended to be called from other application code, it typically will not provide interfaces for user-controllable data itself, minimizing the results that SAST technologies can typically find.

Library code and application code comparision

Feature	Application code	Library code
Entry point	Typically, yes	No
Purpose	Implement business logic	Provides reusable functionality
Usage	Standalone or deployed	Embedded in other apps
Analysis focus	Full program behavior	API exposure and usage patterns

Analyzing library code effectively

To scan library code effectively, you should configure the OpenText SAST to treat the code as a library.

Translate the code as normal as per the language. Go to the appropriate section of this user guide for finding more information about analyzing the appropriate language.

Once ready to scan, set the following property during the scan step:

```
com.fortify.sca.rules.IsLibrary=true
```

When this property is enabled, the analysis engine understands to mimic calls from an outside application calling the library code in order to provide a more thorough analysis.

Other use cases

In addition to libraries, there are many declarative endpoint frameworks that make application code appear similar to library code.

If your web API is using a framework that we do not currently have coverage for (see [Supported technologies]), then enabling this property may also mimic coverage of the framework, though it may also lead to some additional incorrect flows.

Note

This feature is currently supported only for Java code.

Caution

Enabling the property mimics outside code calling into the application, vastly increasing the attack surface, which can lead to significantly more issues and use more resources. This should generally not be enabled on application code except for the stated use cases or unless advised to. In addition, this property does not need to be enabled to support the many declarative endpoint frameworks that we already have coverage for, such as Spring Boot and JAX-RS.

1.23. Scanning for Secrets

OpenText SAST scans are made up a series of analyzers, one of which is able to find information generally across any file type. This enables OpenText SAST to find information hidden in plain view such as secrets, and weaknesses that may be vulnerable agnostic of programming language, such as using attacks involving invisible control characters.

For more information on how to configure this, see Regular expression analysis.

1.23.1. Regular expression analysis

Regular expression (regex) analysis provides the ability for using regular expression rules to detect vulnerabilities in both file content and file names. This analysis can detect vulnerable secrets such as passwords, keys, and credentials in project files.

Important

Regex analysis is language agnostic and therefore it might detect vulnerabilities in file types that OpenText SAST does not officially support.

Regex analysis recursively examines all file paths and path patterns included in the translation phase. Every file found is analyzed unless it is specifically excluded. To manage the files that are included in regex analysis, the following options are available:

- Exclude any file or directory with the `-exclude` option in the translation phase.

  For more information about this option, see Translation Options.

- By default, regex analysis excludes all detectible binary files. To include binary files in the analysis, add the following property to the `fortify-sca.properties` file (or include this property on the command line using the `-D` option):

  ```
  com.fortify.sca.regex.ExcludeBinaries = false
  ```

- By default, regex analysis excludes files larger than 10 MB to ensure that the scan time is acceptable. You can change the maximum file size (in megabytes) with the following property:

  ```
  com.fortify.sca.regex.MaxSize = <max_file_size_mb>
  ```

Regex analysis is enabled by default. To disable regex analysis, add the following property to the `fortify-sca.properties` file or include it on the command line:

```
com.fortify.sca.regex.Enable = false
```

Regex Analysis Properties

1.24. Optimizing results

This section provides guidelines and tips to optimize results when analyzing different codebases with OpenText SAST.

1.24.1. Applying a scan policy to the analysis

For the analysis (scan) phase, you can specify a scan policy to help you identify the most serious vulnerabilities so you can remediate the code quickly. The following table describes the three provided scan policies.

Policy name	Description
security	This is the default scan policy, which excludes issues related to code quality, dataflow from sources that are typically trusted, and issues that are typically noisy from the analysis results. Use this policy to focus code remediation on the security issues.
devops	This scan policy expands on the security policy, by excluding additional issues that might be considered noise, and reducing more low priority issues. Use this scan policy when scan speed is a priority, and developers review results directly (without any intermediate auditing). Issues that remain after you apply this scan policy are probably serious security issues that require remediation. **Note** This devops scan policy does not automatically include any customization made to the local security scan policy.
classic	This scan policy does not exclude any issues. Use this scan policy to see all issues, or if you prefer to filter issues with project templates so it is easier to see hidden issues.

To specify a scan policy for your analysis, include the `-scan-policy` (or `-sc`) option in the analysis phase as shown in the following example:

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

Alternatively, you can specify the scan policy with the `com.fortify.sca.ScanPolicy` property in the `fortify-sca.properties` file. For example:

```
com.fortify.sca.ScanPolicy=devops
```

Note

You can apply a filter file (see Excluding Issues with Filter Files) in addition to a scan policy setting for an analysis. In this case, OpenText SAST applies both the scan policy and the filter file to the analysis.

Creating custom scan policies

The scan policy files reside in the `<sast_install_dir>/Core/config/scales` directory. There is one file for each scan policy. You can change the settings in these policy files to customize your scan policies or you can create your own scan policy files. For information about the syntax used for the scan policy files, see Excluding Issues with Filter Files.

To create a custom scan policy file:

1. Go to `<sast_install_dir>/Core/config/scales/`.
2. Open a text editor and create a file named `scan-policy-<name>.txt`, where *<name>* is the name for your custom scan policy.
3. Add filters to the `scan-policy-<name>.txt` file and save it.
4. To use the custom scan policy for your analysis, type the command as shown in the following example. In this example, the scan policy file name is `scan-policy-myscanpolicy.txt`.

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

   Alternatively, you can specify the custom scan policy in the `fortify-sca.properties` file.

See Also

Translation and Analysis Phase Properties

1.24.2. Excluding issues with filter files

You can create a file to filter out particular vulnerability instances, rules, and vulnerability categories when you run the `sourceanalyzer` command. You specify the file with the `-filter` analysis option.

A filter file is a text file that you can create with any text editor. You specify only the filter items that you *do not* want in this file.

Note
The filter types described in this section apply to both filter files and scan policy files (see Applying a Scan Policy to the Analysis).

The following table lists the available filter types and provides examples for each.

Filter type	Notes	Examples
Category	Specifying only a category will filter out all subcategories **Note** OpenText SAST applies category filters in the initialization phase before any analysis has taken place.	`Poor Error Handling` `J2EE Bad Practices: Leftover Debug Code`
Instance ID	An instance ID of a specific issue **Note** OpenText SAST applies instance ID filters after the analysis phase.	`6291C6A33303ED270C269917AA8A1005`
Rule ID	A rule ID that leads to the reporting of a specific issue **Note** OpenText SAST applies rule ID filters in the initialization phase before any analysis has taken place.	`823FE039-A7FE-4AAD-B976-9EC53FFE4A59`
Priority[1]	The priority values in ascending order are `low`, `medium`, `high`, and `critical`.	`priority <= low` `priority < medium`

Taint flags	Enclose taint flag expressions in parentheses. Use the logical `&&`, `||`, and `!` operators to specify an expression. For a list of taint flags, see *OpenText™ Static Application Security Testing Custom Rules Guide*.	`(SYSTEMINFO || EXCEPTIONINFO)`
		`(WEB || (DATABASE && PRIVATE))`
		`(NETWORK && !XSS)`

Impact[1]		`impact < 0.5`
Likelihood[1]		`likelihood <= 1.5`
Confidence[1]		`confidence < 1.8`
Probability[1]		`probability <= 1.2`
Accuracy[1]		`accuracy <= 1.0`

[1]For the priority and metadata filters, use less than (`<`) or less than or equal to (`<=`).

Composite Filters

When you specify a filter on different lines, OpenText SAST will apply each filter line by line, one at a time. Additionally, you can combine them on one line and use boolean logical operators (`&&`, `||`, `!`) and braces `{}` to group expressions to create more advanced filters.

For example, if you want to filter out `Cross-Site Scripting` issues, given that the issue had a `confidence` less than `4.0`, and the taint flags contained either `DATABASE` or `LDAP`.

You can use the following filter:

```
{ Cross-Site Scripting && confidence < 4.0 } && ( DATABASE || LDAP )
```

If any part of the composite filter is a filter type that can only run post-scan, it will run post-scan regardless of it having items that typically filter pre-scan.

Note

Taint flag filters must be surrounded by parentheses regardless of curly braces.

See Also

Filter File Example

1.24.2.1. Filter file example

As an example, the following output is from a scan of the `EightBall.java` sample. This sample project is included in the `OpenText_SAST_Fortify_Samples_<version>.zip` archive in the `basic/eightball` directory.

The following commands are executed to produce the analysis results:

```
sourceanalyzer -b eightball EightBall.java
sourceanalyzer -b eightball -scan
```

The following results show five detected issues:

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

[6291C6A33303ED270C269917AA8A1005 : high : Path Manipulation : dataflow ]
EightBall.java(12) : ->new FileReader(0)
 EightBall.java(8) : <=> (filename)
 EightBall.java(8) : <->Integer.parseInt(0->return)
 EightBall.java(6) : <=> (filename)
 EightBall.java(4) : ->EightBall.main(0)

[176CC0B182267DD538992E87EF41815F : critical : Path Manipulation : dataflow ]
EightBall.java(12) : ->new FileReader(0)
 EightBall.java(6) : <=> (filename)
 EightBall.java(4) : ->EightBall.main(0)

[E4B3ACF92911ED6D98AAC15876739EC7 : high : Unreleased Resource : Streams : controlflow ]

 EightBall.java(12) : start -> loaded : new FileReader(...)
 EightBall.java(14) : loaded -> end_of_scope : end scope : Resource leaked

 EightBall.java(12) : start -> loaded : new FileReader(...)
 EightBall.java(12) : java.io.IOException thrown
 EightBall.java(12) : loaded -> loaded : throw
 EightBall.java(12) : loaded -> end_of_scope : end scope : Resource leaked : java.io.IOException thrown

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

The following is an example filter file that performs the following:

- Remove all results related to the J2EE Bad Practice category
- Remove the Path Manipulation based on its instance ID
- Remove any dataflow issues that were generated from a specific rule ID

```
#This is a category to filter from scan output
J2EE Bad Practices
#This is an instance ID of a specific issue to be filtered
#from scan output
6291C6A33303ED270C269917AA8A1005
#This is a specific Rule ID that leads to the reporting of a
#specific issue in the scan output: in this case the
#dataflow sink for a Path Manipulation issue.
823FE039-A7FE-4AAD-B976-9EC53FFE4A59
```

To test the filtered output, copy the above text and paste it into a file with the name `test_filter.txt`.

To apply the filtering in the `test_filter.txt` file, execute the following command:

```
sourceanalyzer -b eightball -scan -filter test_filter.txt
```

The filtered analysis produces the following results:

```
[176CC0B182267DD538992E87EF41815F : critical : Path Manipulation : dataflow ]
EightBall.java(12) : ->new FileReader(0)
 EightBall.java(6) : <=> (filename)
 EightBall.java(4) : ->EightBall.main(0)

[E4B3ACF92911ED6D98AAC15876739EC7 : high : Unreleased Resource : Streams : controlflow ]

 EightBall.java(12) : start -> loaded : new FileReader(...)
 EightBall.java(14) : loaded -> end_of_scope : end scope : Resource leaked

 EightBall.java(12) : start -> loaded : new FileReader(...)
 EightBall.java(12) : java.io.IOException thrown
 EightBall.java(12) : loaded -> loaded : throw
 EightBall.java(12) : loaded -> end_of_scope : end scope : Resource leaked : java.io.IOException thrown
```

1.24.3. Using filter sets to exclude issues

You can use filter sets in an issue template created in Fortify Audit Workbench to filter issues from the analysis results. When you apply a filter set that hides issues from view during the analysis phase, OpenText SAST does not write the hidden issues to the FPR. To do this, use Fortify Audit Workbench to create a filter set, and then run the OpenText SAST scan with the filter set and the issue template, which contains the filter set. For more detailed instructions about how to create filters and filter sets in Fortify Audit Workbench, see the *OpenText™ Fortify Audit Workbench User Guide*.

The following example describes the basic steps for how to create and use a filter in an issue template to remove issues from an FPR:

1. Suppose you use OWASP Top 10 2021, and you only want to see issues categorized within this standard. In Fortify Audit Workbench, create a new filter set called `OWASP_Filter`
2. In Fortify Audit Workbench, create a visibility filter in the `OWASP_Filter` filter set:

   ```
   If [OWASP Top 10 2021] does not contain A Then hide issue
   ```

   This filter looks through the issues and if an issue does not map to an OWASP Top 10 2021 category with ‘A’ in the name, then it hides it. Because all OWASP Top 10 2021 categories start with ‘A’ (A01, A02, ..., A10), then any category without the letter ‘A’ is not in the OWASP Top 10 2021. The filter hides the issues from view in Fortify Audit Workbench, but they are still in the FPR.
3. In Fortify Audit Workbench, export the issue template to a file called `IssueTemplate.xml`.
4. Using OpenText SAST, specify the filter set in the analysis phase with the following command:

   ```
   sourceanalyzer -b MyProject -scan -project-template IssueTemplate.xml
   -Dcom.fortify.sca.FilterSet=OWASP_Filter -f MyFilteredResults.fpr
   ```

Although filtering issues with a filter set can reduce the size of the FPR, it does not usually reduce the scan time. OpenText SAST examines the filter set after it calculates the issues to determine whether to write them to the FPR file. The filters in a filter set determine the rule types that OpenText SAST loads.

1.24.4. Filtering using FortifyRemove comments

Similar to linters, compilers, and static analysis tools built directly into IDEs, developers are accustomed to controlling the results of these tools directly from the code. Similarly if required, developers can use inline comments to manage issues triggered by OpenText SAST. Developers can prevent issues from being reported by specifying either the rule ID that triggers the issue or the category of the finding in the `FortifyRemove()`.

When issues are removed with comments, OpenText SAST logs the issues that are removed, including their location and category.

Note

This functionality is available and enabled by default for Java and C# code. The functionality can disabled in `fortify-rules.properties` by setting `com.fortify.sca.rules.EnableRuleComments=false`. For more information, see fortify-rules.properties

Basic Comments

For example, consider the following `Java Hello World` application.

```
public class MyClass {

   public static void main(String[] args) {

      System.out.println("Hello World");

   }

}
```

Consider there is a rule with an ID `625EEE1F-464F-42DC-85D6-269A637EF747` that triggers on the *main function* as *J2EE Bad Practices: Leftover Debug Code.*

If the developer disagrees and they do not want this issue to display any longer, either of the following configurations will prevent the issue from appearing.

```
public class MyClass {

   // FortifyRemove(ID="625EEE1F-464F-42DC-85D6-269A637EF747")

   public static void main(String[] args) {

      System.out.println("Hello World");

   }

}
```

Or

```
public class MyClass {

   // FortifyRemove(Category="J2EE Bad Practices: Leftover Debug Code")

   public static void main(String[] args) {

      System.out.println("Hello World");

   }

}
```

Note: the string argument can use either " or '

Wildcards

The `*` wildcard can be used to expand a category to cover multiple subcategories or multiple matching categories.
For example:

```
// FortifyRemove(Category="Cross-Site Scripting: *")
```

Would remove all variants of Cross-Site Scripting issues.

Whereas:

```
// FortifyRemove(Category="Cross-Site *")
```

Would remove all variants of Cross-Site Scripting issues, along with any categories that start with "Cross-Site", such as "Cross-Site Request Forgery".

Multiple conditions

Other than using wildcards you can specify multiple categories or rule IDs using the `Categories` or `IDs` properties respectively, which take arrays of strings.
For example

```
// FortifyRemove(Categories=["Cross-Site Scripting: Reflected", "Cross-Site Scripting: Persistent"])
```

would prevent either `Cross-Site Scripting: Reflected` or `Cross-Site Scripting: Persistent` issues appearing on the following line.

```
// FortifyRemove(IDs=["A", "B", "C", "D"]
```

Would prevent rules with the IDs `A`, `B`, `C`, or `D` from triggering on the following line.
Additionally you can specify multiple criteria together, separated by a semi-colon (`;`).
For example:

```
// FortifyRemove(Category="SQL Injection"; ID="ABCD-1234")
```

Would prevent `SQL Injection` issues appearing on the following line, as well as prevent issues from rule ID `ABCD-1234` triggering.

Adding Justifications

Issues are logged as removed by `FortifyRemove` comments. A `justification` property can be specified that accepts a string that will be logged alongside the removal information that can help expand on why the issue is being removed.
For example:

```
// FortifyRemove(Category="Cross-Site Scripting: *"; Justification="We remove XSS here because we're using custom framework XYZ that automati
cally protects against the attack")
```

1.24.5. Fortify Java annotations

OpenText provides two versions of the Fortify Java annotations library.

- Annotations with the retention policy set to CLASS (`FortifyAnnotations-CLASS.jar`).

  With this version of the library, Fortify Java annotations are propagated to the bytecode during compilation.

- Annotations with the retention policy set to SOURCE (`FortifyAnnotations-SOURCE.jar`).

  With this version of the library, Fortify Java annotations are not propagated to the bytecode after the code that uses them is compiled.

If you use OpenText Application Security Software products to analyze bytecode of your applications (for example, with OpenText™ Core Application Security assessments), then use the version with the annotation retention policy set to CLASS. If you use OpenText Application Security Software products to analyze the source code of your applications, you can use either version of the library. However, OpenText strongly recommends that you use the library with a retention policy set to SOURCE.

Important

It is a security risk to leave Fortify Java annotations in production code because they can leak information about potential security problems in the code. OpenText recommends that you use annotations with the retention policy set to CLASS only for internalanalysis, and never use them in your application production builds.

This section outlines the annotations available. A sample application is included in the `OpenText_SAST_Fortify_Samples_<version>.zip` archive in the `advanced/javaAnnotations` directory. A `README.txt` file included in the directory describes the sample application, problems that might arise from it, and how to fix these problems using Fortify Java annotations.

There are two limitations with Fortify Java annotations:

- Each annotation can specify only one input and/or one output.
- You can apply only one annotation of each type to the same target.

OpenText provides three main types of annotations:

- Dataflow Annotations
- Field and Variable Annotations
- Other Annotations

You also can write rules to support your own custom annotations. Contact Customer Support for more information.

1.24.5.1. Dataflow annotations

There are four types of Dataflow annotations, similar to Dataflow rules: Source, Sink, Passthrough, and Validate. All are applied to methods and specify the inputs and/or outputs by parameter name or the strings `this` and `return`. Additionally, you can apply the Dataflow Source and Sink annotations to the function arguments.

Source annotations

The acceptable values for the annotation parameter are `this`, `return`, or a function parameter name. For example, you can assign taint to an output of the target method.

```
@FortifyDatabaseSource("return")
String [] loadUserProfile(String userID) {
 ...
}
```

For example, you can assign taint to an argument of the target method.

```
void retrieveAuthCode(@FortifyPrivateSource String authCode) {
 ...
}
```

In addition to specific source annotations, OpenText provides a generic *untrusted* taint source called `FortifySource`.

The following is a complete list of source annotations:

- `FortifySource`
- `FortifyDatabaseSource`
- `FortifyFileSystemSource`
- `FortifyNetworkSource`
- `FortifyPCISource`
- `FortifyPrivateSource`
- `FortifyWebSource`

Passthrough annotations

Passthrough annotations transfer any taint from an input to an output of the target method. It can also assign or remove taint from the output, in the case of `FortifyNumberPassthrough` and `FortifyNotNumberPassthrough`. The acceptable values for the `in` annotation parameter are `this` or a function parameter name. The acceptable values for the `out` annotation parameter are `this`, `return`, or a function parameter name.

```
@FortifyPassthrough(in="a",out="return")
String toLowerCase(String a) {
 ...
}
```

Use `FortifyNumberPassthrough` to indicate that the data is purely numeric. Numeric data cannot cause certain types of issues, such as cross-site scripting, regardless of the source. Using `FortifyNumberPassthrough` can reduce false positives of this type. If a program decomposes character data into a numeric type (int, int[], and so on), you can use `FortifyNumberPassthrough`. If a program concatenates numeric data into character or string data, then use `FortifyNotNumberPassthrough`.

The following is a complete list of passthrough annotations:

- `FortifyPassthrough`
- `FortifyNumberPassthrough`
- `FortifyNotNumberPassthrough`

Sink annotations

Sink annotations report an issue when taint of the appropriate type reaches an input of the target method. Acceptable values for the annotation parameter are `this` or a function parameter name.

```
@FortifyXSSSink("a")
void printToWebpage(int a) {
 ...
}
```

You can also apply the annotation to the function argument or the return parameter. In the following example, an issue is reported when taint reaches the argument `a`.

```
void printToWebpage(int b, @FortifyXSSSink String a) {
 ...
}
```

The following is a complete list of the sink annotations:

- `FortifySink`
- `FortifyCommandInjectionSink`
- `FortifyPCISink`
- `FortifyPrivacySink`
- `FortifySQLSink`
- `FortifySystemInfoSink`
- `FortifyXSSSink`

Validate annotations

Validate annotations remove taint from an output of the target method. Acceptable values for the annotation parameter are `this`, `return`, or a function parameter name.

```
@FortifyXSSValidate("return")
String xssCleanse(String a) {
 ...
}
```

The following is a complete list of validate sink annotations:

- `FortifyValidate`
- `FortifyCommandInjectionValidate`
- `FortifyPCIValidate`
- `FortifyPrivacyValidate`
- `FortifySQLValidate`
- `FortifySystemInfoValidate`
- `FortifyXSSValidate`

1.24.5.2. Field and variable annotations

You can apply these annotations to fields and (in most cases) variables.

Password and private annotations

Use password and private annotations to indicate whether the target field or variable is a password or private data.

```
@FortifyPassword String x;
@FortifyNotPassword String pass;
@FortifyPrivate String y;
@FortifyNotPrivate String cc;
```

In the previous example, string x will be identified as a password and checked for privacy violations and hardcoded passwords. The string pass will not be identified as a password. Without the annotation, it might cause false positives. The `FortifyPrivate` and `FortifyNotPrivate` annotations work similarly, only they do not cause privacy violation issues.

Non-negative and non-zero annotations

Use these annotations to indicate disallowed values for the target field or variable.

```
@FortifyNonNegative int index;
@FortifyNonZero double divisor;
```

In the previous example, an issue is reported if a negative value is assigned to `index` or zero is assigned to `divisor`.

1.24.5.3. Other annotations

Check return value annotation

Use the `FortifyCheckReturnValue` annotation to add a target method to the list of functions that require a check of the return values.

```
@FortifyCheckReturnValue
int openFile(String filename) {
 ...
}
```

Dangerous annotations

With the `FortifyDangerous` annotation, any use of the target function, field, variable, or class is reported. Acceptable values for the annotation parameter are `CRITICAL`, `HIGH`, `MEDIUM`, or `LOW`. These values indicat how to categorize the issue based on the Fortify Priority Order values).

```
@FortifyDangerous{"CRITICAL"}
public class DangerousClass {
 @FortifyDangerous{"HIGH"}
 String dangerousField;
 @FortifyDangerous{"LOW"}
 int dangerousMethod() {
 ...
 }
}
```

1.25. Optimizing performance

This section provides guidelines and tips to optimize memory usage and performance when analyzing different types of codebases with OpenText SAST.

This section contains the following topics:

1.25.1. Antivirus software

The use of antivirus software can negatively impact OpenText SAST performance. If you notice long scan times, OpenText recommends that you temporarily exclude the internal OpenText SAST files from your antivirus software scan. You can also do the same for the directories where the source code resides, however the performance impact on the analysis is less than with the internal directories.

By default, OpenText SAST creates internal files in the following location:

- Windows: `c:\Users\<username>\AppData\Local\Fortify\sca<version>`
- Non-Windows: `<userhome>/.fortify/sca<version>`

where *<version>* is the version of OpenText SAST you are using.

1.25.2. Hardware considerations

The variety of source code makes accurate predictions of memory usage and scan times impossible. The factors that affect memory usage and performance consists of many different factors including:

- Code type
- Codebase size and complexity
- Ancillary languages used (such as JSP, JavaScript, and HTML)
- Number of vulnerabilities
- Type of vulnerabilities (analyzer used)

OpenText developed the following set of "best guess" hardware recommendations based on real-world application scan results. The following table lists these recommendations based on the complexity of the application. In general, increasing the number of available cores might improve scan times.

Application complexity	CPU cores	RAM (GB)	Average scan time	Description
Simple	4	16	1 hour	A standalone system that runs on a server or desktop such as a batch job or a command-line tool.
Medium	8	32	5 hours	A standalone system that works with complex computer models such as a tax calculation system or a scheduling system.
Complex	16	128	4 days	A three-tiered business system with transactional data processing such as a financial system or a commercial website.
Very Complex	32	256	7+ days	A system that delivers content such as an application server, database server, or content management system.

Note

TypeScript and JavaScript scans increase the analysis time significantly. If the total lines of code in an application consist of more than 20% TypeScript or JavaScript, use the next highest recommendation.

Hardware requirements describes the system requirements. However, for large and complex applications, OpenText SAST requires more capable hardware. This includes:

- **Disk I/O**—OpenText SAST is I/O intensive and therefore the faster the hard drive, the more savings on the I/O transactions. OpenText recommends a 7,200 RPM drive, although a 10,000 RPM drive (such as the WD Raptor) or an SSD drive is better.
- **Memory**—See Memory Tuning for more information about how to determine the amount of memory required for optimal performance.
- **CPU**—OpenText recommends a 2.1 GHz or faster processor.

1.25.3. Tuning options

OpenText SAST can take a long time to process complex projects. The time is spent in different phases:

- Translation
- Analysis

OpenText SAST can produce large analysis result files (FPRs), which can take a long time to audit and upload to Application Security. This is referred to as the following phase:

- Audit/Upload

The following table lists tips on how to improve performance in the different time-consuming phases.

Phase	Option	Description	More information
Translation	`-export-build-session` `-import-build-session`	Translate and scan on different machines	Mobile Build Sessions
Analysis	`-quick`	Run a quick scan	Quick Scan
Analysis	`-scan-precision`	Set the scan precision	Configuring Scan Speed with Speed Dial
Analysis	`-bin`	Scan the files related to a binary	Breaking Down Codebases
Analysis	`-Xmx<size>M | G`	Set maximum heap size	Memory Tuning
Analysis	`-Xss<size>M | G`	Set stack size for each thread	Memory Tuning
Analysis Audit/Upload	`-filter <file>`	Apply a filter using a filter file	Using Filter Files
Analysis Audit/Upload	`-disable-source-bundling`	Exclude source files from the FPR file	Excluding Source Code from the FPR

1.25.4. Quick scan

Quick scan mode provides a way to quickly scan your projects for critical- and high-priority issues. OpenText SAST performs the scan faster by reducing the depth of the analysis. It also applies the Quick View filter set. Quick scan settings are configurable. For more details about the configuration of quick scan mode, see fortify-sca-quickscan.properties.

Quick scans are a great way to get many applications through an assessment so that you can quickly find issues and begin remediation. The performance improvement you get depends on the complexity and size of the application. Although the scan is faster than a full scan, it does not provide as robust a result set. OpenText recommends that you run full scans whenever possible.

Limiters

The depth of the OpenText SAST analysis sometimes depends on the available resources. OpenText SAST uses a complexity metric to trade off these resources with the number of vulnerabilities that it can find. Sometimes, this means giving up on a particular function when it does not look like OpenText SAST has enough resources available.

OpenText SAST enables the user to control the “cutoff” point by using OpenText SAST limiter properties. The different analyzers have different limiters. You can run a predefined set of these limiters using a quick scan. See the fortify-sca-quickscan.properties for descriptions of the limiters.

To enable quick scan mode, use the `-quick` option with `-scan` option. With quick scan mode enabled, OpenText SAST applies the properties from the `<sast_install_dir>/Core/config/fortify-sca-quickscan.properties` file, in addition to the standard `<sast_install_dir>/Core/config/fortify-sca.properties` file. You can adjust the limiters that OpenText SAST uses by editing the `fortify-sca-quickscan.properties` file. If you modify `fortify-sca.properties`, it also affects quick scan behavior. OpenText recommends that you do performance tuning in quick scan mode, and leave the full scan in the default settings to produce a highly accurate scan. For description of the quick scan mode properties, see OpenText SAST Properties Files.

Using quick scan and full scan

- **Run full scans periodically**—A periodic full scan is important as it might find issues that quick scan mode does not detect. Run a full scan at least once per software iteration. If possible, run a full scan periodically when it will not interrupt the development workflow, such as on a weekend.
- **Compare quick scan with a full scan**—To evaluate the accuracy impact of a quick scan, perform a quick scan and a full scan on the same codebase. Open the quick scan results in Fortify Audit Workbench and merge it into the full scan. Group the issues by **New Issue** to produce a list of issues detected in the full scan but not in the quick scan.
- **Quick scans and Application Security**—To avoid overwriting the results of a full scan, by default Application Security ignores uploaded FPR files scanned in quick scan mode. However, you can configure a Application Security application version so that FPR files scanned in quick scan are processed. For more information, see analysis results processing rules in the *OpenText™ Application Security User Guide*.

1.25.5. Configuring scan speed with speed dial

You can configure the speed and depth of the scan by specifying a precision level for the analysis phase. You can use these precision levels to adjust the scan time to fit for example, into a pipeline and quickly find a set of vulnerabilities while the developer is still working on the code. Although scans with the speed dial settings are faster than a full scan, it does not provide as robust a result set. OpenText recommends that you run full scans whenever possible.

The precision level controls the depth and precision of the scan by associating configuration properties with each level. The configuration properties files for each level are in the `<sast_install_dir>/Core/config/scales` directory. There is one file for each level: (`level-<precision_level>.properties`). You can modify the settings in these files to create your own specific precision levels.

Notes:

- By default, Application Security blocks uploaded analysis results that were created with a precision level less than four. However, you can configure your Application Security application version so that uploaded audit projects scanned with these precision levels are processed.
- If you merge a speed dial scan with a full scan, this might remove issues from previous scans that still exist in your application (and would be detected again with a full scan).

To specify the speed dial setting for a scan, include the `-scan-precision` (or `-p`) option in the scan phase as shown in the following example:

```
sourceanalyzer -b MyProject -scan -scan-precision <level> -f MyResults.fpr
```

Note

You cannot use the speed dial setting and the `-quick` option in the same scan command.

The following table describes the four precision levels.

Precision level	Description
1	This is the quickest scan and is recommended to scan a few files. By default, a scan with this precision level disables the Buffer Analyzer, Control Flow Analyzer, Dataflow Analyzer, and Null Pointer Analyzer.
2	By default, a scan with this precision level enables all analyzers. The scan runs quicker by performing with reduced limiters. This results in fewer issues detected.
3	This precision level improves intermediate development scan speeds by up to 50% (with a reduction in reported issues). Specifically, this level improves the scan time for typed languages such as Java and C/C++.
4	This is equivalent to a full scan.

You can also specify the scan precision level with the `com.fortify.sca.PrecisionLevel` property in the `fortify-sca.properties` file. For example:

```
com.fortify.sca.PrecisionLevel=1
```

1.25.6. Breaking down codebases

It is more efficient to break down large projects into independent modules. For example, if you have a portal application that consists of several modules that are independent of each other or have few interactions, you can translate and scan the modules separately. The caveat to this is that you might lose dataflow issue detection if some interactions exist.

For C/C++, you might reduce the scan time by using the `–bin` option with the `–scan` option. You need to pass the binary file as the parameter (such as `-bin <filename>.exe -scan` or `-bin <filename>.dll -scan`). OpenText SAST finds the related files associated with the binary and scans them. This is useful if you have several binaries in a makefile.

The following table lists some useful OpenText SAST command-line options to break down codebases.

Option	Description
`-bin <binary>`	Specifies a subset of source files to scan. Only the source files that were linked in the named binary at build time are included in the scan. You can use this option multiple times to specify the inclusion of multiple binaries in the scan.
`-show-binaries`	Displays all objects that were created but not used in the production of any other binaries. If fully integrated into the build, it lists all the binaries produced.
`-show-build-tree`	When used with the `-bin` option, displays all files used to create the binary and all files used to create those files in a tree layout. If the `-bin` option is not present, OpenText SAST displays the tree for each binary.

1.25.7. Limiting analyzers and languages

Occasionally, you might find that a significant amount of the scan time is spent either running one analyzer or analyzing a particular language. It is possible that this analyzer or language is not important to your security requirements. You can limit the specific analyzers that run and the specific languages that OpenText SAST translates, however, this may lead to suboptimal results.

This section contains the following topics:

1.25.7.1. Disabling analyzers

To disable specific analyzers, include the `-analyzers` option to OpenText SAST at scan time with a comma- or colon-separated list of analyzers to enable. The valid parameter values for the `-analyzers` option are `buffer`, `content`, `configuration`, `controlflow`, `dataflow`, `nullptr`, `semantic`, and `structural`.

For example, to run a scan that only includes the Dataflow, Control Flow, and Buffer analyzers, use the following scan command:

```
sourceanalyzer -b MyProject -analyzers dataflow:controlflow:buffer -scan -f MyResults.fpr
```

You can also do the same thing by setting `com.fortify.sca.DefaultAnalyzers` in the OpenText SAST property file `<sast_install_dir>/Core/config/fortify-sca.properties`. For example, to achieve the equivalent of the previous scan command, set the following in the properties file:

```
com.fortify.sca.DefaultAnalyzers=dataflow:controlflow:buffer
```

1.25.7.2. Disabling languages

To disable specific languages, include the `-disable-language` option in the translation phase, which specifies a list of languages that you want to exclude. The valid language values are

`abap`, `actionscript`, `apex`, `cfml`, `cobol`, `configuration`, `cpp`, `dart`, `dotnet`, `golang`, `objc`, `php`, `python`, `ruby`, `swift`, and `vb`.

For example, to perform a translation that excludes configuration and PHP files, use the following command:

```
sourceanalyzer -b MyProject <src_files> -disable-language configuration:php
```

You can also disable languages by setting the `com.fortify.sca.DISabledLanguages` property in the OpenText SAST properties file `<sast_install_dir>/Core/config/fortify-sca.properties`. For example, to achieve the equivalent of the previous translation command, set the following in the properties file:

```
com.fortify.sca.DISabledLanguages=configuration:php
```

For languages that are not available with the `-disable-language`, use the `-exclude` option. For more information, see Translation options.

1.25.8. Optimizing FPR files

This section describes how to handle performance issues related to the audit results (FPR) file. These topics describe how to reduce the scan time, reduce FPR file size, and tips for opening large FPR files.

This section contains the following topics:

1.25.8.1. Using filter files

You can use a file to filter out specific vulnerability instances, rules, and vulnerability categories from the analysis results. If you determine that a certain issue category or rule is not relevant for a particular scan, you can stop OpenText SAST from adding them to the FPR. Using a filter file can reduce both the scan time and analysis results file size.

For example, if you scan a simple program that just reads a specified file, you might not want to see path manipulation issues, because these are not likely planned as part of the functionality. To filter out path manipulation issues, create a file that contains a single line:

`Path Manipulation`

Save this file as `filter.txt`. Use the `-filter` option in the analysis phase as shown in the following example:

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

The analysis output in `MyResults.fpr` does not include any issues with the category Path Manipulation. For more information and an example of a filter file, see Excluding Issues with Filter Files.

1.25.8.2. Using filter sets

Filters in an issue template determine how the results from OpenText SAST are shown. In addition to filters, filter sets enable you to have a selection of filters used at any one time. Each FPR has an issue template associated with it. You can use filter sets to reduce the number of issues based on conditions you specify with filters in an issue template. This can dramatically reduce the size of an FPR.

To do this, use Fortify Audit Workbench to create a filter in a filter set, and then run the OpenText SAST scan with the filter set and the containing issue template. For more information and a basic example of how to create a filter set, see Excluding Issues with Filters Sets.

Note

Although filtering issues with a filter set can reduce the size of the FPR, they do not usually reduce the scan time. OpenText SAST examines the filter set after it calculates the issues to determine whether to write them to the FPR file. The filters in a filter set determine the rule types that OpenText SAST loads.

1.25.8.3. Excluding source code from the FPR

You can reduce the size of the FPR file by excluding the source code information from the FPR. This is especially valuable for large source files or codebases. Typically, you do not get a scan time reduction for small source files using this method.

There are properties you can use to prevent OpenText SAST from including source code in the FPR. You can set either property in the `<sast_install_dir>/Core/config/fortify-sca.properties` file or specify an option on the command line. The following table describes these settings.

Property name	Description

`com.fortify.sca.FPRDisableSourceBundling=true`
Command-Line Option:
`-disable-source-bundling`
	Excludes source code from the FPR.

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

The following command-line example uses both options to exclude both the source code and code snippets from the FPR:

```
sourceanalyzer -b MyProject -disable-source-bundling
-fvdl-no-snippets -scan -f MySourcelessResults.fpr
```

1.25.8.4. Reducing the FPR file size

There are a few ways to reduce the size of FPR files. The quickest way to do this without affecting results is to exclude the source code from the FPR as described in Excluding Source Code from the FPR. You can also reduce the size of a merged FPR with the FPRUtility (see the *OpenText™ Application Security Tools Guide*).

There are a few other properties that you can use to select what is excluded from the FPR. You can set these properties in the `<sast_install_dir>/Core/config/fortify-sca.properties` file or specify an option on the command line for the analysis (scan) phase.

Property name	Description

`com.fortify.sca.FPRDisableMetatable=true`
Command-Line Option:
`-disable-metatable`
	Excludes the metatable from the FPR. Fortify Audit Workbench uses the metatable to map information in Functions view.

`com.fortify.sca.FVDLDisableDescriptions=true`
Command-Line Option:
`-fvdl-no-descriptions`
	Excludes rule descriptions from the FPR. If you do not use custom descriptions, the descriptions in the Fortify Taxonomy (https://vulncat.fortify.com) are used.

`com.fortify.sca.FVDLDisableEngineData=true`
Command-Line Option:
`-fvdl-no-enginedata`
	Excludes engine data from the FPR. This is useful if your FPR contains many warnings when you open the file in Fortify Audit Workbench.
	Note
	If you exclude engine data from the FPR, you must merge the FPR with the current audit project locally before you upload it to Application Security. Application Security cannot merge it on the server because the FPR does not contain the OpenText SAST version.

`com.fortify.sca.FVDLDisableProgramData=true`
Command-Line Option:
`-fvdl-no-progdata`
	Excludes the program data from the FPR. This removes the Taint Sources information from the Functions view in Fortify Audit Workbench. This property typically only has a minimal effect on the overall size of the FPR file.

1.25.8.5. Opening large FPR files

To reduce the time required to open a large FPR file in Fortify Audit Workbench, you can set some properties in the `<sast_install_dir>/Core/config/fortify.properties` file. For more information about these properties, see the *OpenText™ Application Security Tools Guide*. The following table describes the properties you can use to reduce the time to open large FPR files.

Property name	Description
`com.fortify.model.DisableProgramInfo=true`	Disables use of the code navigation features in Fortify Audit Workbench.
`com.fortify.model.IssueCutoffStartIndex=<num>` (inclusive) `com.fortify.model.IssueCutoffEndIndex=<num>` (exclusive)	Sets the start and end index for issue cutoff. The `IssueCutoffStartIndex` property is inclusive and `IssueCutoffEndIndex` is exclusive so that you can specify a subset of issues you want to see. For example, to see the first 100 issues, specify the following: `com.fortify.model.IssueCutoffStartIndex=0com.fortify.model.IssueCutoffEndIndex=101` Because the `IssueCutoffStartIndex` is `0` by default, you do not need to specify this property.
`com.fortify.model.IssueCutoffByCategoryStartIndex=<num>` (inclusive) `com.fortify.model.IssueCutoffByCategoryEndIndex=<num>` (exclusive)	Sets the start index for issue cutoff by category. These two properties are similar to the previous cutoff properties except these are specified for each category. For example, to see the first five issues for every category, specify the following: com.fortify.model. IssueCutoffByCategoryEndIndex=6
`com.fortify.model.MinimalLoad=true`	Minimizes the data loaded from the FPR. This also restricts usage of the Functions view and might prevent Fortify Audit Workbench from loading the source from the FPR.
`com.fortify.model.MaxEngineErrorCount=<num>`	Specifies the number of OpenText SAST reported warnings to load from the FPR. For projects with many scan warnings, reducing this number from a default of 3000 can speed up the load time of large FPR files.
`com.fortify.model.ExecMemorySetting`	Specifies the JVM heap memory size for Fortify Audit Workbench to start external command-line tools such as iidmigrator and fortifyupdate.

1.25.9. Monitoring long running scans

When you run OpenText SAST, large and complex scans can often take a long time to complete. During the scan it is not always clear what is happening. While OpenText recommends that you provide your debug logs to the Customer Support team, there are a couple of ways to see what OpenText SAST is doing and how it is performing in real-time.

This section contains the following topics:

1.25.9.1. Using the SCAState tool

The SCAState command-line tool enables you to see up-to-date state analysis information during the analysis phase. The SCAState tool is located in the `<sast_install_dir>/bin` directory. In addition to a live view of the analysis, it also provides a set of timers and counters that show where OpenText SAST spends its time during the analysis phase. For more information about how to use SCAState, see the *Checking the OpenText SAST Scan Status*.

1.25.9.2. Using JMX tools

You can use tools to monitor OpenText SAST with JMX technology. These tools can provide a way to track OpenText SAST performance over time. For more information about these tools, see the Oracle® documentation.

Note

These are third-party tools and OpenText does not provide or support them.

This section contains the following topics:

- Using JConsole
- Using Java VisualVM

1.25.9.2.1. Using JConsole

JConsole is an interactive monitoring tool that complies with the JMX specification. The disadvantage of JConsole is that you cannot save the output.

To use JConsole, you must first set some additional JVM parameters. Set the following environment variable:

```
export SCA_VM_OPTS="-Dcom.sun.management.jmxremote
-Dcom.sun.management.jmxremote.port=9090
-Dcom.sun.management.jmxremote.ssl=false
-Dcom.sun.management.jmxremote.authenticate=false"
```

After the JMX parameters are set, start a scan. During the scan, start JConsole to monitor OpenText SAST locally or remotely with the following command:

```
jconsole <host_name>:9090
```

1.25.9.2.2. Using Java VisualVM

Java VisualVM offers the same capabilities as JConsole. It also provides more detailed information on the JVM and enables you to save the monitor information to an application snapshot file. You can store these files and open them later with Java VisualVM.

Similar to JConsole, before you can use Java VisualVM, you must set the same JVM parameters described in Using JConsole.

After the JVM parameters are set, start the scan. You can then start Java VisualVM to monitor the scan either locally or remotely with the following command:

```
jvisualvm <host_name>:9090
```

1.26. Using mobile build sessions

With an OpenText SAST mobile build session (MBS), you can translate a project on one machine and scan it on another. A mobile build session (MBS file) includes all the files needed for the analysis phase including the source code. To improve scan time, you can perform the translation on the build computer, and then use a better equipped computer for the scan by doing either of the following:

- Use ScanCentral SAST client to move the MBS to sensors for analysis (see ScanCentral SAST)
- Move the build session (MBS file) to another computer that has an OpenText SAST installation, import the MBS (see Importing a mobile build session), and then run the analysis.

Note

OpenText Core Application Security (Fortify on Demand) users can generate an MBS file to package translated code for uploading some languages.

You must have the same version of OpenText Application Security Content (Rulepacks) installed on both the system where you perform the translation and the system where you perform the analysis.

This section contains the following topics:

- Mobile build session version compatibility
- Creating a mobile build session
- Importing a mobile build session

1.26.1. Mobile build session version compatibility

The OpenText SAST version on the translate machine must be compatible with the OpenText SAST version on the analysis machine. The version number format is *<major>.<minor>.<patch>.<build_number>* (for example, 25.4.0.0140). The *<major>* and *<minor>* portions of the OpenText SAST version numbers on both the translation and the analysis machines must match. For example, 25.4.0 and 25.4.x are compatible. To determine the OpenText SAST version number, type `sourceanalyzer -v` on the command line.

You can obtain the build ID and the OpenText SAST version from an MBS file with the following command:

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

1.26.2. Creating a mobile build session

On the machine where you performed the translation, issue the following command to generate a mobile build session:

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

where `<file>.mbs` is the file name you provide for the OpenText SAST mobile build session.

1.26.3. Importing a mobile build session

After you move the `<file>.mbs` file to the machine where you want to perform the scan, you can import the mobile build session into the OpenText SAST project root directory.

To import the mobile build session, type the following command:

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

After you import your OpenText SAST mobile build session, you can proceed to the analysis phase. Perform a scan with the same build ID that was used in the translation.

You cannot merge multiple mobile build sessions into a single MBS file. Each exported build session must have a unique build ID. However, after all the build IDs are imported on the same OpenText SAST installation, you can scan multiple build IDs in one scan with the `-b` option (see Analysis Phase).

1.27. Troubleshooting

This section contains the following topics:

1.27.1. Exit codes

The following table describes the possible OpenText SAST exit codes.

Exit code	Description
0	Success
1	Generic failure
2	Invalid input files (this might indicate that an attempt was made to translate a file that has an extension that OpenText SAST does not support)
3	Process timed out
4	Analysis completed with numbered warning messages written to the console and/or to the log file
5	Analysis completed with numbered error messages written to the console and/or to the log file
6	Scan phase was unable to generate issue results
7	Unable to detect a valid license or the LIM license expired at run time

By default, OpenText SAST only returns exit codes 0, 1, 2, 3, or 7.

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

The valid values are:

- `nothing`—Returns any of the default exit codes (0, 1, 2, 3, or 7).
- `warnings`—Returns exit codes 4 and 5 in addition to the default exit codes.
- `errors`—Returns exit code 5 in addition to the default exit codes.
- `no_output_file`—Returns exit code 6 in addition to the default exit codes.

1.27.2. Memory tuning

The amount of physical RAM required for a scan depends on the complexity of the code. By default, OpenText SAST automatically allocates the memory it uses based on the physical memory available on the system. This is generally sufficient. As described in Output Options, you can adjust the Java heap size with the -Xmx command-line option.

This section describes suggestions for what you can do if you encounter OutOfMemory errors during the analysis.

Note

You can set the memory allocation options discussed in this section to run for all scans by setting the `SCA_VM_OPTS` environment variable.

This section contains the following topics:

- Java heap exhaustion
- Native heap exhaustion
- Stack overflow

1.27.2.1. Java heap exhaustion

Java heap exhaustion is the most common memory problem that might occur during OpenText SAST scans. It is caused by allocating too little heap space to the Java virtual machine that OpenText SAST uses to scan the code. You can identify Java heap exhaustion from the following symptom.

Symptom

One or more of these messages appears in the OpenText SAST log file and in the command-line output:

```
There is not enough memory available to complete analysis. For details on making more memory available, please consult the user manual.
java.lang.OutOfMemoryError: Java heap space
java.lang.OutOfMemoryError: GC overhead limit exceeded
```

Resolution

To resolve a Java heap exhaustion problem, allocate more heap space to the OpenText SAST Java virtual machine when you start the scan. To increase the heap size, use the `-Xmx` command-line option when you run the OpenText SAST scan. For example, `-Xmx1G` makes 1 GB available. Before you use this parameter, determine the maximum allowable value for Java heap space. The maximum value depends on the available physical memory.

Heap sizes between 32 GB and 48 GB are not advised due to internal JVM implementations. Heap sizes in this range perform worse than at 32 GB. Heap sizes smaller than 32 GB are optimized by the JVM. If your scan requires more than 32 GB, then you need 64 GB or more. As a guideline, assuming no other memory intensive processes are running, do not allocate more than 2/3 of the available memory.

If the system is dedicated to running OpenText SAST, you do not need to change it. However, if the system resources are shared with other memory-intensive processes, subtract an allowance for those other processes.

Note

You do not need to account for other resident but not active processes (while OpenText SAST is running) that the operating system might swap to disk. Allocating more physical memory to OpenText SAST than is available in the environment might cause “thrashing,” which typically slows down the scan along with everything else on the system.

1.27.2.2. Native heap exhaustion

Native heap exhaustion is a rare scenario where the Java virtual machine can allocate the Java memory regions on startup, but is left with so few resources for its native operations (such as garbage collection) that it eventually encounters a fatal memory allocation failure that immediately terminates the process.

Symptom

You can identify native heap exhaustion by abnormal termination of the OpenText SAST process and the following output on the command line:

```
# A fatal error has been detected by the Java Runtime Environment:
#
# java.lang.OutOfMemoryError: requested ... bytes for GrET ...
```

Because this is a fatal Java virtual machine error, it is usually accompanied by an error log created in the working directory with the file name `hs_err_pidNNN.log`.

Resolution

Because the problem is a result of overcrowding within the process, the resolution is to reduce the amount of memory used for the Java memory regions (Java heap). Reducing this value should reduce the crowding problem and allow the scan to complete successfully.

1.27.2.3. Stack overflow

Each thread in a Java application has its own stack. The stack holds return addresses, function/method call arguments, and so on. If a thread tends to process large structures with recursive algorithms, it might need a large stack for all those return addresses. With the JVM, you can set that size with the `-Xss` option.

Symptoms

This message typically appears in the OpenText SAST log file, but might also appear in the command-line output:

```
java.lang.StackOverflowError
```

Resolution

The default stack size is 16 MB. To increase the stack size, pass the `-Xss` option to the `sourceanalyzer` command. For example, `-Xss32M` increases the stack to 32 MB.

1.27.3. Scanning complex functions

During a scan, the Dataflow Analyzer might encounter a function for which it cannot complete the analysis and reports the following message:

```
Function <name> is too complex for <analyzer> analysis and will be skipped (<identifier>)
```

where:

- `<name>` is the name of the source code function
- `<analyzer>` is the name of the analyzer
- `<identifier>` is the type of complexity, which is one of the following:
  - `l`: Too many distinct locations
  - `m`: Out of memory
  - `s`: Stack size too small
  - `t`: Analysis taking too much time
  - `v`: Function visits exceed the limit

The depth of analysis OpenText SAST performs sometimes depends on the available resources. OpenText SAST uses a complexity metric to trade off these resources against the number of vulnerabilities that it can find. Sometimes, this means giving up on a particular function when OpenText SAST does not have enough resources available. This is normally when you see the "Function too complex" messages.

When you see this message, it does not necessarily mean that OpenText SAST completely ignored the function in the program. For example, the Dataflow Analyzer typically visits a function many times before completing the analysis, and might not have run into this complexity limit in the previous visits. In this case, the results include everything learned from the previous visits.

You can control the "give up" point using OpenText SAST properties called limiters. Different analyzers have different limiters.

The following sections provide a discussion of a resolution for this issue.

This section contains the following topics:

1.27.3.1. Dataflow Analyzer limiters

There are three types of complexity identifiers for the Dataflow Analyzer:

- `l`: Too many distinct locations
- `m`: Out of memory
- `s`: Stack size too small
- `v`: Function visits exceed the limit

To resolve the issue identified by `s`, increase the stack size for by setting `-Xss` to a value greater than 16 MB.

To resolve the complexity identifier of `m`, increase the physical memory for OpenText SAST.

To resolve the complexity identifier of `l`, you can adjust the following limiters in the OpenText SAST property file `<sast_install_dir>/Core/config/fortify-sca.properties` or on the command line.

Property name	**Default value**
`com.fortify.sca.limiters.MaxTaintDefForVar`	1000
`com.fortify.sca.limiters.MaxTaintDefForVarAbort`	4000
`com.fortify.sca.limiters.MaxFieldDepth`	4

The `MaxTaintDefForVar` limiter is a dimensionless value expressing the complexity of a function, while `MaxTaintDefForVarAbort` is the upper bound for it. Use the `MaxFieldDepth` limiter to measure the precision when the Dataflow Analyzer analyzes any given object. OpenText SAST always tries to analyze objects at the highest precision possible.

If a given function exceeds the `MaxTaintDefForVar` limit at a given precision, the Dataflow Analyzer analyzes that function with lower precision (by reducing the `MaxFieldDepth` limiter). When you reduce the precision, it reduces the complexity of the analysis. When the precision cannot be reduced any further, OpenText SAST then proceeds with analysis at the lowest precision until either it finishes, or the complexity exceeds the `MaxTaintDefForVarAbort` limiter. In other words, OpenText SAST tries harder at the lowest precision to get at least some results from the function. If OpenText SAST reaches the `MaxTaintDefForVarAbort` limiter, it gives up on the function entirely and you get the "Function too complex" warning.

To resolve the complexity identifier of `v`, you can adjust the property `com.fortify.sca.limiters.MaxFunctionVisits`. This property sets the maximum number of times the taint propagation analyzer visits functions. The default is `50`.

1.27.3.2. Control Flow and Null Pointer analyzer limiters

There are two types of complexity identifiers for both Control Flow and Null Pointer analyzers:

- `m`: Out of memory
- `t`: Analysis taking too much time

Due to the way that the Dataflow Analyzer handles function complexity, it does not take an indefinite amount of time. Control Flow and Null Pointer analyzers, however, can take an exceptionally long time when analyzing complex functions. Therefore, OpenText SAST provides a way to abort the analysis when this happens, and then you get the "Function too complex" message with a complexity identifier of `t`.

To change the maximum amount of time these analyzers spend to analyze functions, you can adjust the following property values in the OpenText SAST property file `<sast_install_dir>/Core/config/fortify-sca.properties` or on the command line.

Property name	Description	Default value
`com.fortify.sca.CtrlflowMaxFunctionTime`	Sets the time limit (in milliseconds) for Control Flow analysis on a single function.	600000 (10 minutes)
`com.fortify.sca.NullPtrMaxFunctionTime`	Sets the time limit (in milliseconds) for Null Pointer analysis on a single function.	300000 (5 minutes)

To resolve the complexity identifier of `m`, increase the physical memory for OpenText SAST.

Note

If you increase these limiters or time settings, it makes the analysis of complex functions take longer. It is difficult to characterize the exact performance implications of a particular value for the limiters/time, because it depends on the specific function in question. If you never want to see the "Function too complex" warning, you can set the limiters/time to an extremely high value, however it can cause unacceptable scan time.

1.27.4. Issue non-determinism

Running in parallel analysis mode might introduce issue non-determinism. If you experience any problems, contact Customer Support, and disable parallel analysis mode. Disabling parallel analysis mode results in sequential analysis, which can be substantially slower but provides deterministic results across multiple scans.

To disable parallel analysis mode:

1. Open the `fortify-sca.properties` file located in the `<sast_install_dir>/Core/config` directory in a text editor.
2. Change the value for the `com.fortify.sca.MultithreadedAnalysis` property to `false`.

```
com.fortify.sca.MultithreadedAnalysis=false
```

1.27.5. Locating the log files

We will announce deprecation of the `-debug`, `-verbose`, and `-debug-verbose` options in the System Requirements doc and release notes in the future. The GUI Tools team uses these options in the tools so we need to let them know too.>>

By default, OpenText SAST creates log files in the following location:

- Windows: `C:\Users\<username>\AppData\Local\Fortify\sca<version>\log`
- Non-Windows: `<userhome>/.fortify/sca<version>/log`

where `<version>` is the version of OpenText SAST that you are using.

The following table describes the OpenText SAST default log files.

File names	Description
`sca.log` `scaX.log`	The standard log provides a log of informational messages, warnings, and errors that occurred in the run of sourceanalyzer.
`sca_FortifySupport.log` `scaX_FortifySupport.log`	The OpenText SAST Support log provides: • The same log messages as the standard log file, but with additional details • Additional detailed messages that are not included in the standard log file This log file is helpful to Customer Support or the development team to troubleshoot any issues.

To specify a log file on the command line, see Other options.

If you encounter warnings or errors that you cannot resolve, provide the OpenText SAST Support log file to Customer Support.

1.27.6. Configuring log files

You can configure the information that OpenText SAST writes to the log files by setting logging properties (see Logging Properties) and by updating the `<sast_install_dir>/Core/config/log4j2.xml` file. You can configure the following log file settings:

- The location and name of the log file

  Property: `com.fortify.sca.LogFile`

- Log level (see Understanding Log Levels)

  Property: `com.fortify.sca.LogLevel`

- Whether to overwrite the log files for each run of sourceanalyzer

  Property: `com.fortify.sca.ClobberLogFile`

  Command-line option: `-clobber-log`

For information about how to make changes to the `log4j2.xml` file, see https://logging.apache.org/log4j/2.x/manual/index.html.

Understanding log levels

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

Log level	Description
DEBUG	Includes information that Customer Support or the development team can use to troubleshoot an issue
INFO	Basic information about the translation or scan process
WARN	Information about issues where the translation or scan did not stop, but might require your attention for accurate results
ERROR	Information about an issue that might require attention
FATAL	Information about an error that caused the translation or scan to abort

1.27.7. Reporting issues and requesting enhancements

Feedback is critical to the success of this product. To request enhancements or patches, or to report issues, visit Customer Support at https://www.microfocus.com/support.

Include the following information when you contact customer support:

- Product: OpenText SAST
- Version number of OpenText SAST and any independent OpenText SAST modules: To determine the version numbers, run the following:

  ```
  sourceanalyzer -version
  ```

- Platform: (for example, Red Hat Enterprise Linux *<version>*)
- Operating system: (such as Linux)

To request an enhancement, include a description of the feature enhancement.

To report an issue, provide enough detail so that support can duplicate the issue. The more descriptive you are, the faster support can analyze and resolve the issue. Also include the log files, or the relevant portions of them, from when the issue occurred.

1.28. Command-line reference

This section describes general OpenText SAST command-line options and how to specify source files for analysis. Command-line options that are specific to a language are described in the section for that language.

This section contains the following topics:

1.28.1. Specifying files and directories

File specifiers are expressions that allow you to pass a long list of files or a directory to OpenText SAST using wildcard characters. OpenText SAST recognizes two types of wildcard characters: a single asterisk character (*) matches part of a file name, and double asterisk characters (**) recursively matches directories. You can specify one or more files, one or more file specifiers, or a combination of files and file specifiers. Separate multiple file specifiers with semicolons (Windows) or colons (non-Windows).

<files> | *<file_dir_specifiers>*

Windows and many Linux shells automatically expand parameters that contain the asterisk character (*), so you must enclose file-specifier expressions in quotes. Also, on Windows, you can use the backslash character (\) as the directory separator instead of the forward slash (/).

Note
File specifiers do not apply to languages that require compiler or build integration.

The following table describes examples of file and directory specifiers.

File or directory specifier	Description
<dir> "*<dir>*/**/*"	Matches all files in the named directory and any subdirectories or the named directory when used for a directory parameter.
"*<dir>*/**/Example.java"	Matches any file named Example.java found in the named directory or any subdirectories.
"*<dir>*/*.java" "*<dir>*/*.jar"	Matches any file with the specified extension found in the named directory.
"*<dir>*/**/*.kt" "*<dir>*/**/*.jar"	Matches any file with the specified extension found in the named directory or any subdirectories.
"*<dir>*/**/beta/**"	Matches all directories and files found in the named directory that have beta in the path, including beta as a file name.
"*<dir>*/**/classes/"	Matches all directories and files with the name classes found in the named directory and any subdirectories.
"**/test/**"	Matches all files in the current directory tree that have a test element in the path, including test as a file name.
"**/test/**/*;**/build/**/*" or "**/test/**/*:**/build/**/*"	Matches all files in the current directory tree that have a test or a build element in the path, including test or build as a file name.
"**/webgoat/*"	Matches all files in any webgoat directory in the current directory tree. Matches: • /src/main/java/org/owasp/webgoat • /test/java/org/owasp/webgoat Does not match (assignments directory does not match) • /test/java/org/owasp/webgoat/assignments

1.28.2. Directives

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

Directive	Description
`-clean`	Deletes all OpenText SAST intermediate files and build records. If you specify a build ID, only files and build records that relate to that build ID are deleted.
`-show-binaries`	Displays all objects created but not used in the production of any other binaries. If fully integrated into the build, it lists all the binaries produced.
`-show-build-ids`	Displays a list of all known build IDs.
`-show-build-tree`	When you scan with the `-bin` option, displays all files used to create the binary and all files used to create those files in a tree layout. If the `-bin` option is not present, the tree is displayed for each binary. **Note** This option can generate an extensive amount of information.
`-show-build-warnings`	Use with the `-b` option to display any errors and warnings that occurred in the translation phase on the console. **Note** Fortify Audit Workbench also displays these errors and warnings in the results Certification tab.
`-show-files`	Displays the files included in the specified build ID. When the `-bin` option is present, displays only the source files that went into the binary.
`-show-loc`	Use with the `-b` option to display the number of lines in the translated code.

1.28.2.1. LIM license directives

OpenText SAST provides directives to manage the usage of your LIM license. You can store or clear the LIM license pool credentials. You can also request (and release) a detached lease for offline analysis if the specified license pool permits detached leases.

Note

By default, OpenText SAST requires an HTTPS connection to the LIM server and you must have a trusted certificate. For more information, see Adding Trusted Certificates.

Use the directives described in the following table for a license managed by the LIM.

LIM directive	Description

`-store-license-pool-credentials "<lim_url>|<lim_pool_name>|<lim_pool_pwd>|<proxy_url>|<proxy_user>|<proxy_pwd>"`	Stores your LIM license pool credentials so that OpenText SAST uses the LIM for licensing. The proxy information is optional. OpenText SAST stores the poo password and the proxy credentials provided with this directive in the `fortify-sca.properties` file as encrypted data. If your license pool credentials chang after you have installed OpenText SAST, you can run this directive again to save the new credentials. Example:
	`sourceanalyzer -store-license-pool-credentials "https://<ip_address>:<port>|TeamA|mypassword"`
	Associated property names:
	`com.fortify.sca.lim.Url`
	`com.fortify.sca.lim.PoolName`
	`com.fortify.sca.lim.PoolPassword`
	`com.fortify.sca.lim.ProxyUrl`
	`com.fortify.sca.lim.ProxyUsername`
	`com.fortify.sca.lim.ProxyPassword`

`-clear-license-pool-credentials`	Removes the LIM license pool credentials from the `fortify-sca.properties` file. If your license pool credentials change, you can remove them with this directive, and then use the `-store-license-pool-credentials` directive to save the new credentials.
`-request-detached-lease <duration>`	Requests a detached lease from the LIM license pool f exclusive use on this system for the specified duration (in minutes). This enables you to run OpenText SAST even when disconnected from your corporate intranet **Note** To use this directive, the license pool must be configured to allow detached leases.
`-release-detached-lease`	Releases a detached lease back to the license pool.

1.28.3. Translation options

The following table describes the general translation options that can be used with most translation commands.

Translation option	Description
`-b <build_id>`	Specifies a build ID. OpenText SAST uses a build ID to track the files that are compiled and combined as part of a build, and then later, to scan those files. **Equivalent property name:** `com.fortify.sca.BuildID`
`-disable-language <languages>`	Specifies a colon-separated list of languages to exclude from the translation phase. The valid language values are `abap`, `actionscript`, `apex`, `cfml`, `cobol`, `configuration`, `cpp`, `dart`, `dotnet`, `golang`, `objc`, `php`, `python`, `ruby`, `swift`, and `vb`. **Equivalent property name:** `com.fortify.sca.DISabledLanguages`
`-enable-language <languages>`	Specifies a colon-separated list of languages to translate. The valid language values are `abap`, `actionscript`, `apex`, `cfml`, `cobol`, `configuration`, `cpp`, `dart`, `dotnet`, `golang`, `objc`, `php`, `python`, `ruby`, `swift`, and `vb`. **Equivalent property name:** `com.fortify.sca.EnabledLanguages`

`-exclude <file_specifiers>`	Specifies the files to exclude from the translation. Files excluded from translation are also not scanned. Separate multiple file paths with semicolons (Windows) or colons (non-Windows). The following example excludes all Java files in any `Test` subdirectory.
	`sourceanalyzer -b MyProject –cp "**/*.jar" "**/*" -exclude "**/Test/*.java"`
	See Specifying files and directories for more information on how to use file specifiers.
	Equivalent property name:
	`com.fortify.sca.exclude`

`-encoding <encoding_name>`	Specifies the source file encoding type. OpenText SAST enables you to scan a project that contains differently encoded source files. To work with a multi-encoded project, you must specify the `-encoding` option in the translation phase, when OpenText SAST first reads the source code file. OpenText SAST remembers this encoding in the build session and propagates it into the FVDL file. Valid encoding names are from the `java.nio.charset.Charset`. Typically, if you do not specify the encoding type, OpenText SAST uses `file.encoding` from the `java.io.InputStreamReader` constructor with no encoding parameter. In a few cases (for example with the ActionScript parser), OpenText SAST defaults to `UTF-8` encoding. **Equivalent property name:** `com.fortify.sca.InputFileEncoding`
`-nc`	When specified before a compiler command line, OpenText SAST translates the source file but does not run the compiler.
`-noextension-type <file_type>`	Specifies the file type for source files that have no extension. The valid file type values are `ABAP`, `ACTIONSCRIPT`, `APEX`, `APEX_OBJECT`, `APEX_TRIGGER`, `ARCHIVE`, `ASPNET`, `ASP`, `ASPX`, `BITCODE`, `BSP`, `BYTECODE`, `CFML`, `COBOL`, `CSHARP`, `DART`, `DOCKERFILE`, `FLIGHT`, `GENERIC`, `GO`, `HCL`, `HOCON`, `HTML`, `INI`, `JAVA`, `JAVA_PROPERTIES`, `JAVASCRIPT`, `JINJA`, `JSON`, `JSP`, `JSPX`, `JUPYTER`, `KOTLIN`, `MSIL`, `MXML`, `OBJECT`, `PHP`, `PLSQL`, `PYTHON`, `RUBY`, `RUBY_ERB`, `SCALA`, `SWIFT`, `SWC`, `SWF`, `TLD`, `SQL`, `TSQL`, `TYPESCRIPT`, `VB`, `VB6`, `VBSCRIPT`, `VISUAL_FORCE`, `VUE`, and `XML`, and `YAML` .
`-disable-compiler-resolution`	Specifies to include build script files that have the same name as a build tool (such as gradlew) during translation as source files. **Equivalent property name:** `com.fortify.sca.DisableCompilerName`
`-project-root`	Specifies the directory to store intermediate files generated in the translation and analysis phases. OpenText SAST makes extensive use of intermediate files located in this project root directory. In some cases, you can achieve better performance for analysis by making sure this directory is on local storage rather than on a network drive. **Equivalent property name:** `com.fortify.sca.ProjectRoot`

1.28.4. Analysis options

The following table describes the general analysis options (typically with `-scan`).

Analysis option	Description
`-b <build_id>`	Specifies the build ID used in a prior translation command. **Equivalent property name:** `com.fortify.sca.BuildID`
`-scan`	Causes OpenText SAST to perform a security analysis for the specified build ID.
`-scan-policy <policy_name>` | `-sc <policy_name>`	Specifies a scan policy for the analysis. The valid policy names are `classic`, `security`, and `devops`. For more information, see Applying a Scan Policy to the Analysis. **Equivalent property name:** `com.fortify.sca.ScanPolicy`
`-analyzers <analyzer_list>`	Specifies the analyzers you want to enable with a colon- or comma-separated list of analyzers. The valid analyzer names are `buffer`, `content`, `configuration`, `controlflow`, `dataflow`, `nullptr`, `semantic`, and `structural`. You can use this option to disable analyzers that are not required for your security requirements. **Equivalent property name:** `com.fortify.sca.DefaultAnalyzers`
`-p <level>` | `-scan-precision <level>`	Uses speed dial to scan the project with a scan precision level. The lower the scan precision level, the faster the scan performance. The valid values are `1`, `2`, `3`, and `4`. For more information, see Configuring Scan Speed. **Equivalent property name:** `com.fortify.sca.PrecisionLevel`
`-project-root`	Specifies the directory to store intermediate files generated in the translation and analysis phases. OpenText SAST makes extensive use of intermediate files located in this project root directory. In some cases, you can achieve better performance for analysis by making sure this directory is on local storage rather than on a network drive. **Equivalent property name:** `com.fortify.sca.ProjectRoot`
`-project-template <file>`	Specifies the issue template file to use for the scan. This only affects scans on the local machine. If you upload the FPR to Application Security, it uses the issue template assigned to the application version. **Equivalent property name:** `com.fortify.sca.ProjectTemplate`
`-quick`	Quickly scan the project for critical- and high-priority issues using the `fortify-sca-quickscan.properties` file, which provides a less in-depth analysis. By default, quick scan disables the Buffer Analyzer and the Control Flow Analyzer. In addition, it applies the Quick View filter set. For more information, see Quick Scan. **Equivalent property name:** `com.fortify.sca.QuickScanMode`
`-filter <file>`	Specifies a results filter file. For more information, see Optimizing results. **Equivalent property name:** `com.fortify.sca.FilterFile`
`-bin <binary>` | `-binary-name <binary>`	Specifies a subset of source files to scan. Only the source files that were linked in the named binary at build time are included in the scan. You can use this option multiple times to specify the inclusion of multiple binaries in the scan. **Equivalent property name:** `com.fortify.sca.BinaryName`

`-disable-default-rule-type`
 	
`<type>`
	Used to test custom rules. Disables all rules of the specified type in the default Rulepacks. You can use this option multiple times to specify multiple rule types.
	The `<type>` parameter is the XML tag minus the suffix `Rule`. For example, use `DataflowSource` for DataflowSourceRule elements. You can also specify specific sections of characterization rules, such as `Characterization:Control flow`, `Characterization:Issue`, and `Characterization:Generic`.
	The `<type>` parameter is case-insensitive.

`-no-default-issue-rules`	Used to test custom rules. Disables rules in default Rulepacks that lead directly to issues. OpenText SAST still loads rules that characterize the behavior of functions. **Note** This is equivalent to disabling the following rule types: DataflowSink, Semantic, Controlflow, Structural, Configuration, Content, Statistical, Internal, and Characterization:Issue. **Equivalent property name:** `com.fortify.sca.NoDefaultIssueRules`
`-no-default-rules`	Used to test custom rules. Disables loading of rules from the default Rulepacks. OpenText SAST processes the Rulepacks for description elements and language libraries, but processes no rules. **Equivalent property name:** `com.fortify.sca.NoDefaultRules`

`-no-default-source-rules`	Used to test custom rules. Disables source rules in the default Rulepacks. **Note** Characterization source rules are not disabled. **Equivalent property name:** `com.fortify.sca.NoDefaultSourceRules`
`-no-default-sink-rules`	Used to test custom rules. Disables sink rules in the default Rulepacks. **Note** Characterization sink rules are not disabled. **Equivalent property name:** `com.fortify.sca.NoDefaultSinkRules`
`-rules` *`<file>`*|*`<dir>`*	Specifies a custom Rulepack or directory. You can use this option multiple times to specify multiple Rulepack files. If you specify a directory, OpenText SAST includes all the files in the directory with the `.bin` and `.xml` extensions. **Equivalent property name:** `com.fortify.sca.RulesFile`

1.28.5. Output options

The following table describes the output options. Apply all these options during the analysis phase (with the `-scan` option). You can specify the `build-label`, `build-project`, and `build-version` options during the translation phase and they are overridden if specified again for the analysis phase.

Output option	Description
`-f <file>` | `-output-file <file>`	Specifies the file to which analysis results are written. If you do not specify an output file, OpenText SAST writes the output to the terminal. **Equivalent property name:** `com.fortify.sca.ResultsFile`
`-format <format>`	Controls the output format. Valid options are `fpr`, `fvdl`, `fvdl.zip`, `text`, and `auto`. The default is `auto`, which selects the output format based on the file name extension of the file provided with the `-f` option. The FVDL is an XML file that contains the detailed OpenText SAST analysis results. This includes vulnerability details, rule descriptions, code snippets, command-line options used in the scan, and any scan errors or warnings. The FPR is a package of the analysis results that includes the FVDL file as well as extra information such as a copy of the source code used in the scan, the external metadata, and custom rules (if applicable). Fortify Audit Workbench is automatically associated with the `.fpr` extension. **Note** If you use result certification, you must specify the `fpr` format. See the *OpenText™ Fortify Audit Workbench User Guide* for information about result certification. You can prevent some information from being included in the FPR or FVDL file to improve scan time or output file size. See other options in this table and see Optimizing FPR Files. **Equivalent property name:** `com.fortify.sca.Renderer`
`-append`	Appends results to the file specified with the `-f` option. The resulting FPR file contains the issues from the earlier scan as well as issues from the current scan. The build information and program data (lists of sources and sinks) sections are also merged. To use this option, the output file format must be `fpr` or `fvdl`. For information on the `-format` output option, see the description in this table. The engine data, which includes OpenText Application Security Content information, command-line options, system properties, warnings, errors, and other information about the execution of OpenText SAST (as opposed to information about the program being analyzed), is not merged. Because engine data is not merged with the `-append` option, OpenText does not certify results generated with `-append`. If this option is not specified, OpenText SAST adds any new findings to the FPR file, and labels the older result as `previous` findings. In general, only use the `-append` option when it is impossible to analyze an entire application at once. **Equivalent property name:** `com.fortify.sca.OutputAppend`
`-build-label <label>`	Specifies a label for the project to include in the analysis results. You can include this option during the translation or the analysis phase. OpenText SAST does not use this label for code analysis. If this option is specified for both translation and analysis, then only the last specified label is passed to the analysis results. **Equivalent property name:** `com.fortify.sca.BuildLabel`
`-build-project <project_name>`	Specifies a name for the project to include in the analysis results. You can include this option during the translation or the analysis phase. OpenText SAST does not use this name for code analysis. **Equivalent property name:** `com.fortify.sca.BuildProject`
`-build-version <version>`	Specifies a version for the project to include in the analysis results. You can include this option during the translation or the analysis phase. OpenText SAST does not use this version for code analysis. **Equivalent property name:** `com.fortify.sca.BuildVersion`
`-disable-source-bundling`	Excludes source files from the analysis results file. The analysis results will still include snippets. **Equivalent property name:** `com.fortify.sca.FPRDisableSourceBundling`
`-fvdl-no-descriptions`	Excludes the OpenText Application Security Content descriptions from the analysis results file. **Equivalent property name:** `com.fortify.sca.FVDLDisableDescriptions`
`-fvdl-no-enginedata`	Excludes engine data from the analysis results file. The engine data includes OpenText Application Security Content information, command-line options, system properties, warnings, errors, and other information about the OpenText SAST execution. **Equivalent property name:** `com.fortify.sca.FVDLDisableEngineData`
`-fvdl-no-progdata`	Excludes program data from the analysis results file. This removes the taint source information from the Functions view in Fortify Audit Workbench. **Equivalent property name:** `com.fortify.sca.FVDLDisableProgramData`

`-fvdl-no-snippets`	Excludes the code snippets from the analysis results file. **Equivalent property name:** `com.fortify.sca.FVDLDisableSnippets`

1.28.6. Other options

The following table describes other options.

Other option	Description

`@<file>`	Reads command-line options from the specified file. The plain text *<file>* contains options and parameters, each on a separate line.
	For example, instead of running the command `sourceanalyzer -b my_build_id -source 17 -cp lib.jar Test.java`, you can run the following command: `sourceanalyzer @optfile.txt` where the `optfile.txt` file contains:
	`"-b"`
	`"my_build_id"`
	`"-source"`
	`"17"`
	`"-cp"`
	`"lib.jar"`
	`"Test.java"`

`-h` | `-?` | `-help`	Prints a summary of the command-line options.
`-debug`	Includes debug information in the OpenText SAST Support log file, which is only useful for Customer Support to help troubleshoot. **Equivalent property name:** `com.fortify.sca.Debug`
`-debug-verbose`	This is the same as the `-debug` option, but it includes more details, specifically for parse errors. **Equivalent property name:** `com.fortify.sca.DebugVerbose`
`-debug-mem`	Includes performance information in the OpenText SAST Support log. **Equivalent property name:** `com.fortify.sca.DebugTrackMem`
`-verbose`	Sends verbose status messages to the console and to the OpenText SAST Support log file. **Equivalent property name:** `com.fortify.sca.Verbose`
`-logfile <file>`	Specifies the log file that OpenText SAST creates. For default log file locations, see Locating the log files. **Equivalent property name:** `com.fortify.sca.LogFile`
`-clobber-log`	Directs OpenText SAST to overwrite the log file for each run of sourceanalyzer. Without this option, OpenText SAST appends information to the log file. **Equivalent property name:** `com.fortify.sca.ClobberLogFile`
`-quiet`	Disables the command-line progress information. **Equivalent property name:** `com.fortify.sca.Quiet`
`-version` | `-v`	Displays the OpenText SAST version and versions of various independent modules included with OpenText SAST (all other functionality is contained in OpenText SAST).
`-autoheap`	Enables automatic allocation of memory based on the physical memory available on the system. This is the default memory allocation setting.
`-Xmx<size>M` | `G`	Manually specifies the maximum amount of memory OpenText SAST uses. **Note** OpenText recommends that you use the default memory allocation setting defined by `-autoheap` instead of manually specifying the maximum memory with this option. Heap sizes between 32 GB and 48 GB are not advised due to internal JVM implementations. Heap sizes in this range perform worse than at 32 GB. The JVM optimizes heap sizes smaller than 32 GB. If your scan requires more than 32 GB, then you need 64 GB or more. As a guideline, assuming no other memory intensive processes are running, do not allocate more than 2/3 of the available memory. When you specify this option, make sure that you do not allocate more memory than is physically available, because this degrades performance. As a guideline, and the assumption that no other memory intensive processes are running, do not allocate more than 2/3 of the available memory.

1.29. Configuration options

The OpenText SAST installer places a set of properties files on your system. Properties files contain configurable settings for OpenText SAST runtime analysis, output, and performance.

This section contains the following topics:

1.29.1. Properties files

The properties files are located in the `<sast_install_dir>/Core/config` directory. The installed properties files contain default values. OpenText recommends that you consult with your project leads before you make changes to the properties in the properties files. You can modify any of the properties in the configuration file with any text editor. You can also specify the property on the command line with the `-D` option.

The following table lists the OpenText SAST properties files. Property files for the OpenText SAST applications and tools are described in the *OpenText™ Application Security Tools Guide*.

Properties file name	Description	More information
`fortify-sca.properties`	Defines the OpenText SAST configuration properties.	fortify-sca.properties
`fortify-sca-quickscan.properties`	Defines the configuration properties applicable for an OpenText SAST quick scan.	fortify-sca-quickscan.properties
`fortify-rules.properties`	Defines the configuration properties that determine rule behavior.	fortify-rules.properties

1.29.1.1. Properties file format

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

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

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

Note

When you specify a path for Windows systems as the property value, you must escape any backslash character (\) with a backslash (for example: `com.fortify.sca.ASPVirtualRoots.Library=C:\\WebServer\\CustomerA\\inc`).

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

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

1.29.1.2. Overriding settings

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

The following table lists the order of precedence for OpenText SAST properties.

Order	Property specification	Description
1	Command line with the `-D` option	Properties specified on the command line have the highest priority and you can specify them in any scan.
2	OpenText SAST quick scan configuration file	**Note** You can specify either quick scan or a scan precision level. Therefore, these property settings both have second priority. Properties specified in the quick scan configuration file (`fortify-sca-quickscan.properties`) have the second priority, but only if you include the `-quick` option to enable quick scan mode.
	OpenText SAST scan precision property files	Properties specified in the scan precision property files have the second priority, but only if you include the `-scan-precision` option to enable scan precision.
3	OpenText SAST configuration file	Properties specified in the OpenText SAST configuration file (`fortify-sca.properties`) have the lowest priority. Edit this file to change the property values on a more permanent basis for all scans.

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

1.29.2. fortify-sca.properties

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

This section contains the following topics:

1.29.2.1. Translation and analysis phase properties

The properties for the `fortify-sca.properties` file in the following table are general properties that apply to the translation and/or analysis (scan) phase.

Property name	Description
Translation and scan	
`com.fortify.sca.BuildID`	Specifies the build ID of the build. **Value type:** String **Default:** (none) **Command-line option:** `-b`
`com.fortify.sca.CmdlineOptionsFileEncoding`	Specifies the encoding of the command-line options fil this property, for example, to specify Unicode file path `java.nio.charset.Charset` **Note** This property is only valid in the `fortify` `fortify-sca-quickscan.properites` file **Value type:** String **Default:** JVM system default encoding **Example:** `com.fortify.sca.CmdlineOptionsFileEn`
`com.fortify.sca.DISabledLanguages`	Specifies a colon-separated list of languages to exclud `abap`, `actionscript`, `apex`, `cfml`, `cobol`, `configurati` and `vb`. **Value type:** String **Default:** (none) **Command-line option:** `-disable-language`
`com.fortify.sca.EnabledLanguages`	Specifies a colon-separated list of languages to transla `abap`, `actionscript`, `apex`, `cfml`, `cobol`, `configurati` and `vb`. **Value type:** String **Default:** All languages in the specified source are tran `com.fortify.sca.DISabledLanguages` property. **Command-line option:** `-enable-language`
`com.fortify.sca.DisableCompilerName`	If set to true, OpenText SAST includes build script files translation as source files. **Value type:** Boolean **Default:** `false` **Command-line option:** `-disable-compiler-resolu`
`com.fortify.sca.ProjectRoot`	Specifies the directory to store intermediate files gene makes extensive use of intermediate files located in th performance for analysis by making sure this directory **Value type:** String (path) **Default (Windows):** `${win32.LocalAppdata}/Fort` **Note** `${win32.LocalAppdata}` is a variable tha shell folder. **Default (non-Windows):** `$home/.fortify` **Command-line option:** `-project-root` **Example:** `com.fortify.sca.ProjectRoot=C:\Users\`
Translation	

`com.fortify.sca.fileextensions.java`
`com.fortify.sca.fileextensions.cs`
`com.fortify.sca.fileextensions.js`
`com.fortify.sca.fileextensions.py`
`com.fortify.sca.fileextensions.rb`
`com.fortify.sca.fileextensions.aspx`
`com.fortify.sca.fileextensions.php`
 	
Note
This is a partial list. For the complete list, see the properties file.
	Specifies how to translate specific file name extension
	extension types are `ABAP`, `ACTIONSCRIPT`, `APEX`, `APEX_`
	`BSP`, `BYTECODE`, `CFML`, `COBOL`, `CSHARP`, `DART`, `DOCKERFIL`
	`JAVA_PROPERTIES`, `JAVASCRIPT`, `JINJA`, `JSON`, `JSP`, `JSP`
	`RUBY`, `RUBY_ERB`, `SCALA`, `SWIFT`, `SWC`, `SWF`, `TLD`, `SQL`, `TS(`
	and `YAML`.
	Value type: String (valid language type)
	Default: See the `fortify-sca.properties` file for th
	Examples:
	`com.fortify.sca.fileextensions.java=JAVA`
	`com.fortify.sca.fileextensions.cs=CSHARP`
	`com.fortify.sca.fileextensions.js=TYPESCRIPT`
	`com.fortify.sca.fileextensions.py=PYTHON`
	`com.fortify.sca.fileextensions.swift=SWIFT`
	`com.fortify.sca.fileextensions.razor=ASPNET`
	`com.fortify.sca.fileextensions.php=PHP`
	`com.fortify.sca.fileextensions.tf=HCL`
	You can also specify a value of `oracle:<path_to_scr`
	that accepts one command-line parameter of a file na
	valid OpenText SAST file type (see previous list) to std
	zero return code or the script does not exist, the file is
	file.
	Example:
	`com.fortify.sca.fileextensions.jsp=oracle:<pa`

`com.fortify.sca.compilers.javac=com.fortify.sca.util.compilers.JavacCompiler`
`com.fortify.sca.compilers.c++=com.fortify.sca.util.compilers.GppCompiler`
`com.fortify.sca.compilers.make=com.fortify.sca.util.compilers.TouchlessCompiler`
`com.fortify.sca.compilers.mvn=com.fortify.sca.util.compilers.MavenAdapter`
 	
Note
This is a partial list. For the complete list,
see the properties file.
	Specifies custom-named compilers.
	Value type: String (compiler)
	Default: See the Compilers section in the `fortify-sc`
	Example:
	To tell OpenText SAST that “my-gcc” is a gcc compiler
	`com.fortify.sca.compilers.my-gcc=com.fortify.`
	Notes:
	• Compiler names can begin or end with an asterisk
	• Execution of clang/clang++ is not supported with
	`com.fortify.sca.compilers.g++=`
	
	`com.fortify.sca.util.compilers.GppComp`

`com.fortify.sca.UseAntListener`	If set to true, OpenText SAST includes `com.fortify.d` **Value type:** Boolean **Default:**`false`
`com.fortify.sca.exclude`	Specifies one or more files to exclude from translation (non-Windows). See Specifying Files and Directories fc **Value type:** String **Default:** Not enabled **Command-line option:**`-exclude` **Example:**`com.fortify.sca.exclude=file1.x;file2`
`com.fortify.sca.InputFileEncoding`	Specifies the source file encoding type. OpenText SAS source files. To work with a multi-encoded project, you when OpenText SAST first reads the source code file. propagates it into the FVDL file. Typically, if you do not specify the encoding type, Ope `java.io.InputStreamReader` constructor with no enc parser), OpenText SAST defaults to `UTF-8`. **Value type:** String **Default:** (none) **Command-line option:**`-encoding` **Example:** `com.fortify.sca.InputFileEncoding=UTF-16`
`com.fortify.sca.RegExecutable`	On Windows platforms, specifies the path to the `reg.e` Cygwin syntax, even when you run OpenText SAST fro backslash. **Value type:** String (path) **Default:**`reg` **Example:** `com.fortify.sca.RegExecutable=C:\\Windows\\Sy`

`com.fortify.sca.xcode.TranslateAfterError`	Specifies whether the xcodebuild touchless adapter cc non-zero exit code. If set to false, translation stops aft OpenText SAST touchless build halts with the same ex translation of the build file identified prior to the xcode (unless some other error also occurs). Regardless of this setting, if xcodebuild exits with a nc written to the log file. **Value type:** Boolean **Default:**`false`
Scan	
`com.fortify.sca.AddImpliedMethods`	If set to true, OpenText SAST generates implied metho **Value type:** Boolean **Default:** `true`
`com.fortify.sca.alias.Enable`	If set to true, enables alias analysis. **Value type:** Boolean **Default:** `true`
`com.fortify.sca.analyzer.controlflow.EnableTimeOut`	Specifies whether to enable Control Flow Analyzer tim **Value type:** Boolean **Default:** `true`
`com.fortify.sca.BinaryName`	Specifies a subset of source files to scan. Only the sou included in the scan. **Value type:** String (path) **Default:** (none) **Command-line option:**`-bin` or `-binary-name`
`com.fortify.sca.DefaultAnalyzers`	Specifies a comma- or colon-separated list of the type `buffer`, `content`, `configuration`, `controlflow`, `data` **Value type:** String **Default:** This property is commented out and all anal **Command-line option:** `-analyzers`
`com.fortify.sca.DisableFunctionPointers`	If set to true, disables function pointers during the sca **Value type:** Boolean **Default:**`false`
`com.fortify.sca.EnableAnalyzer`	Specifies a comma- or colon-separated list of analyzer values for this property are `buffer`, `content`, `configu` `structural`. **Value type:** String **Default:** (none)

`com.fortify.sca.EnableSubtraceFiltering`	If set to true, filters out partial duplicates where issues
	For example, if the engine finds 2 similar issues with t
	`A -> B -> C -> D`
	`B -> C -> D`
	The second issue is removed as a subtrace duplicate c
	accurate one.
	Value type: Boolean
	Default: `true`

`com.fortify.sca.ExitCodeLevel`	Extends the default exit code options. See Exit Codes property.
`com.fortify.sca.FilterFile`	Specifies the path to a filter file for the scan. See Abou **Value type:** String (path) **Default:** (none) **Command-line option:** `-filter`
`com.fortify.sca.FilteredInstanceIDs`	Specifies a comma-separated list of IIDs to be filtered **Value type:** String **Default:** (none) **Example:** `com.fortify.sca.FilteredInstanceIDs=CA4E1623A`
`com.fortify.sca.FilteredRuleLanguages`	Specifies a comma- or colon-separated list of language `abap`, `actionscript`, `apex`, `cfml`, `cobol`, `configurati` and `vb`. **Value type:** String **Default:** (none) **Example:**`com.fortify.sca.FileredRuleLanguages=`
`com.fortify.sca.MaxPassthroughChainDepth`	Specifies the length of a taint path between input and **Value type:** Integer **Default:**`4`

`com.fortify.sca.MultithreadedAnalysis`	Specifies whether OpenText SAST runs in parallel anal **Value type:** Boolean **Default:**`true`
`com.fortify.sca.Phase0HigherOrder.Languages`	Specifies a comma-separated list of languages for whi the ability to track dataflow through higher-order code values are `python`, `swift`, `ruby`, `javascript`, and `typ` **Value type:** String **Default:** `python,ruby,swift,javascript,typescri`
`com.fortify.sca.Phase0HigherOrder.Timeout.Hard`	Specifies the total time (in seconds) for higher-order a immediately. OpenText recommends this timeout limit in case some that you set the hard timeout to about 50% longer tha soft timeout occurs first. **Value type:** Number **Default:** `2700`
`com.fortify.sca.PrecisionLevel`	Specifies the scan precision. Scans with a lower precis **Value type:** Number **Default:** (none) **Command-line option:** `-scan-precision` | `-p`
`com.fortify.sca.ProjectTemplate`	Specifies the issue template file to use for the scan. Th FPR to Application Security, it uses the issue template **Value type:** String **Default:** (none) **Command-line option:** `-project-template` **Example:** `com.fortify.sca.ProjectTemplate=test_issuetem`
`com.fortify.sca.QuickScanMode`	If set to true, OpenText SAST performs a quick scan. C `quickscan.properties`, instead of the `fortify-sca.` **Value type:** Boolean **Default:** (not enabled) **Command-line option:**`-quick`
`com.fortify.sca.ScanPolicy`	Specifies the scan policy for prioritizing reported vulne scan policy values are `classic`, `security`, and `devops` **Value type:** String **Default:**`security` **Command-line option:**`-sc` or `-scan-policy`
`com.fortify.sca.ThreadCount`	Specifies the number of threads for parallel analysis m threads used because of a resource constraint. If you reduction in the number of threads used might solve t **Value type:** Integer **Default:** (number of available processor cores)
`com.fortify.sca.TypeInferenceFunctionTimeout`	The amount of time (in seconds) that type inference c not specified. **Value type:** Long **Default:**`60`
`com.fortify.sca.TypeInferenceLanguages`	Comma- or colon-separated list of languages that use for dynamically-typed languages. **Value type:** String **Default:**`javascript,python,ruby,typescript`
`com.fortify.sca.TypeInferencePhase0Timeout`	Specifies the total amount of time (in seconds) that ty Unlimited if set to zero or is not specified. **Value type:** Long **Default:** `300`
`com.fortify.sca.UniversalBlacklist`	Specifies a colon-separated list of functions to hide fro **Value type:** String **Default:**`.*yyparse.*`

1.29.2.2. Regex analysis properties

The properties for the `fortify-sca.properties` file in the following table apply to regular expression analysis.

Property name	Description
`com.fortify.sca.regex.Enable`	If set to true, regular expression analysis is enabled. **Value type:** Boolean **Default:**`true`
`com.fortify.sca.regex.ExcludeBinaries`	If set to true, binary files are excluded from a regular expression analysis. **Value type:** Boolean **Default:**`true`
`com.fortify.sca.regex.MaxSize`	Specifies the maximum size (in megabytes) for files that are scanned in a regular expression analysis. Files that exceed this file size maximum are excluded from a regular expression analysis. **Value type:** Number **Default:**`10`

See Also

Regular Expression Analysis

1.29.2.3. LIM license properties

The properties for the `fortify-sca.properties` file in the following table apply to licensing with the LIM.

Property name	Description
`com.fortify.sca.lim.Url`	Specifies the LIM server API URL. Do not edit this value directly with a text editor. Use the command-line option to change this value. **Value type:** String **Default:** (none) **Command-line option:**`-store-license-pool-credentials` **Examples:** https://*<ip_address>*:*<port>*
`com.fortify.sca.lim.PoolName`	Specifies the LIM license pool name. Do not edit this value directly with a text editor. Use the command-line option to change this value. **Value type:** String **Default:** (none) **Command-line option:**`-store-license-pool-credentials`
`com.fortify.sca.lim.PoolPassword`	Specifies the LIM license pool password (encrypted). Do not edit this value directly with a text editor. Use the command-line option to change this value. **Value type:** String **Default:** (none) **Command-line option:**`-store-license-pool-credentials`
`com.fortify.sca.lim.ProxyUrl`	Specifies the proxy server used to connect to the LIM server. **Value type:** String **Default:** (none) **Examples:**`http://proxy.example.com:8080` `https://proxy.example.com` **Command-line option:**`-store-license-pool-credentials`
`com.fortify.sca.lim.ProxyUsername`	Specifies an encrypted user name for proxy authentication to connect to the LIM server. Do not edit this value directly with a text editor. Use the command-line option to change this value. **Value type:** String **Default:** (none) **Command-line option:**`-store-license-pool-credentials`
`com.fortify.sca.lim.ProxyPassword`	Specifies an encrypted password for proxy authentication to connect to the LIM server. Do not edit this value directly with a text editor. Use the command-line option to change this value. **Value type:** String **Default:** (none) **Command-line option:**`-store-license-pool-credentials`
`com.fortify.sca.lim.RequireTrustedSSLCert`	If set to true, any attempt to connect to the LIM server without a trusted certificate fails. If this property is set to false, a message displays when any attempt to connect to the LIM server without a trusted certificate occurs. **Value type:** Boolean **Default:**`true`
`com.fortify.sca.lim.WaitForInitialLicense`	If set to true and LIM license pool credentials are stored, OpenText SAST waits for a LIM license to become available before starting a translation or scan. If this property is set to false, OpenText SAST aborts if it cannot obtain a LIM license. **Value type:** Boolean **Default:**`true`

LIM License Directives

1.29.2.4. Rule properties

The properties for the `fortify-sca.properties` file in the following table apply to rules (and custom rules) and Rulepacks.

Property name	Description
`com.fortify.sca.DefaultRulesDir`	Sets the directory used to search for the OpenText provided encrypted rules files. **Value Type:** String (path) **Default:** `${com.fortify.Core}/config/rules`
`com.fortify.sca.RulesFile`	Specifies a custom Rulepack or directory. If you specify a directory, all of the files in the directory with the `.bin` and `.xml` extensions are included. **Value Type:** String (path) **Default:** (none) **Command-line option:** `-rules`
`com.fortify.sca.CustomRulesDir`	Sets the directory used to search for custom rules. **Value Type:** String (path) **Default:** `${com.fortify.Core}/config/customrules`
`com.fortify.sca.RulesFileExtensions`	Specifies a list of file extensions for rules files. Any files in `<sast_install_dir>/Core/config/rules` (or a directory specified with the `-rules` option) whose extension is in this list is included. The `.bin` extension is always included, regardless of the value of this property. The delimiter for this property is the system path separator. **Value Type:** String **Default:** `.xml`
`com.fortify.sca.NoDefaultRules`	If set to true, rules from the default Rulepacks are not loaded. OpenText SAST processes the Rulepacks for description elements and language libraries, but no rules are processed. **Value Type:** Boolean **Default:** (none) **Command-line option:** `-no-default-rules`
`com.fortify.sca.NoDefaultIssueRules`	If set to true, disables rules in default Rulepacks that lead directly to issues. OpenText SAST still loads rules that characterize the behavior of functions. This can be helpful when creating custom issue rules. **Value Type:** Boolean **Default:** (none) **Command-line option:** `-no-default-issue-rules`
`com.fortify.sca.NoDefaultSourceRules`	If set to true, disables source rules in the default Rulepacks. This can be helpful when creating custom source rules. **Note** Characterization source rules are not disabled. **Value Type:** Boolean **Default:** (none) **Command-line option:** `-no-default-source-rules`
`com.fortity.sca.NoDefaultSinkRules`	If set to true, disables sink rules in the default Rulepacks. This can be helpful when creating custom sink rules. **Note** Characterization sink rules are not disabled. **Value Type:** Boolean **Default:** (none) **Command-line option:** `-no-default-sink-rules`

1.29.2.5. Java and Kotlin properties

The properties for the `fortify-sca.properties` file in the following table apply to the translation of Java and Kotlin code.

Property name	Description
`com.fortify.sca.JavaClasspath`	Specifies the class path used to analyze Java or Kotlin source code. Separate multiple paths with semicolons (Windows) or colons (non-Windows). **Value type:** String (paths) **Default:** (none) **Command-line option:** `-cp` or `-classpath`
`com.fortify.sca.JdkVersion`	Specifies the Java source code version for Java or Kotlin translation. **Value type:** String **Default:** `11` **Command-line option:** `-jdk` or `-source`
`com.fortify.sca.CustomJdkDir`	Specifies a directory that contains a JDK version that is not included in the OpenText SAST installation (*<sast_install_dir>*/Core/bootcp/). **Value type:** String (path) **Default:** (none) **Command-line option:** `-custom-jdk-dir`
`com.fortify.sca.JavaSourcepath`	Specifies a semicolon- (Windows) or colon-separated (non-Windows) list of Java or Kotlin source file directories that are not included in the scan but are used for name resolution. The source path is similar to class path, except it uses source files rather than class files for resolution. **Value type:** String (paths) **Default:** (none) **Command-line option:** `-sourcepath`
`com.fortify.sca.Appserver`	Specifies the application server to process JSP files. The valid values are `weblogic` or `websphere`. **Value type:** String **Default:** (none) **Command-line option:** `-appserver`
`com.fortify.sca.AppserverHome`	Specifies the application server's home directory. For WebLogic, this is the path to the directory that contains `server/lib`. For WebSphere, this is the path to the directory that contains the `JspBatchCompiler` script. **Value type:** String (path) **Default:** (none) **Command-line option:** `-appserver-home`
`com.fortify.sca.AppserverVersion`	Specifies the version of the WebLogic or WebSphere application server. **Value type:** String **Default:** (none) **Command-line option:** `-appserver-version`
`com.fortify.sca.JavaExtdirs`	Specifies directories to include implicitly on the class path for WebLogic and WebSphere application servers. **Value type:** String **Default:** (none) **Command-line option:** `-extdirs`
`com.fortify.sca.JavaSourcepathSearch`	If set to true, OpenText SAST only translates Java source files that are referenced by the target file list. Otherwise, OpenText SAST translates all files included in the source path. **Value type:** Boolean **Default:** `true`
`com.fortify.sca.DefaultJarsDirs`	Specifies semicolon- or colon-separated list of directories of commonly used JAR files. JAR files located in these directories are appended to the end of the class path option (`-cp`). **Value type:** String **Default:** `default_jars`
`com.fortify.sca.DecompileBytecode`	If set to true, Java bytecode is decompiled for the translation. **Value type:** Boolean **Default:** `false`
`com.fortify.sca.jsp.UseSecurityManager`	If set to true, the JSP parser uses JSP security manager. **Value type:** Boolean **Default:** `true`
`com.fortify.sca.jsp.DefaultEncoding`	Specifies the encoding for JSPs. **Value type:** String (encoding) **Default:** `ISO-8859-1`

`com.fortify.sca.jsp.LegacyDataflow`	If set to true, enables additional filtering on JSP-related dataflow to reduce the amount of spurious false positives detected. **Value type:** Boolean **Default:** `false` **Command-line option:** `-legacy-jsp-dataflow`
`com.fortify.sca.KotlinJvmDefault`	Specifies the generation of the `DefaultImpls` class for methods with bodies in Kotlin interfaces. The valid values are: • `disable`—Specifies to generate the `DefaultImpls` class for each interface that contains methods with bodies. • `all`—Specifies to generate the `DefaultImpls` class if an interface is annotated with `@JvmDefaultWithCompatibility`. • `all-compatibility`—Specifies to generate the `DefaultImpls` class unless an interface is annotated with `@JvmDefaultWithoutCompatibility`. **Value type:** String **Default:** `disable`
`com.fortify.sca.ShowUnresolvedSymbols`	If set to true, displays any unresolved types, fields, and functions referenced in translated Java source files at the end of the translation. **Value type:** Boolean **Default:** `false` **Command-line option:** `-show-unresolved-symbols`

Analyzing Java, Kotlin and JSP projects

1.29.2.6. Visual Studio and MSBuild project properties

The properties for the `fortify-sca.properties` file in the following table apply to the translation of .NET projects and solutions.

Property name	Description

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

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

`com.fortify.sca.DisableASPExternalEntries`	If set to true, disables ASP external entries in the scan. **Value type:** Boolean **Default:** `false`

Translating Visual Studio and MSBuild Projects

1.29.2.7. JavaScript and TypeScript properties

The properties for the `fortify-sca.properties` file in the following table apply to the translation of JavaScript and TypeScript code.

Property name	Description
`com.fortify.sca.EnableDOMModeling`	If set to true, OpenText SAST generates JavaScript code to model the DOM tree that an HTMI during the translation phase and identifies DOM-related issues (such as cross-site scripting is property if the code you are translating includes HTML files that have embedded or referenc **Note** Enabling this property can increase the translation time. **Value type:** Boolean **Default:** `false`
`com.fortify.sca.DOMModeling.tags`	If you set the `com.fortify.sca.EnableDOMModeling` property to true, you can specify addit separated HTML tags names for OpenText SAST to include in the DOM modeling. **Value type:** String **Default:** `body`, `button`, `div`, `form`, `iframe`, `input`, `head`, `html`, and `p`. **Example:** `com.fortify.sca.DOMModeling.tags=ul,li`
`com.fortify.sca.JavaScript.src.domain.whitelist`	Specifies trusted domain names where OpenText SAST can download referenced JavaScript f Delimit the URLs with vertical bars. **Value type:** String **Default:** (none) **Example:** `com.fortify.sca.JavaScript.src.domain.whitelist=http://www.xyz.com|h`
`com.fortify.sca.DisableJavascriptExtraction`	If set to true, JavaScript code embedded in JSP, JSPX, PHP, and HTML files is not extracted an **Value type:** Boolean **Default:** `false`
`com.fortify.sca.EnableTranslationMinifiedJS`	If set to true, enables translation for minified JavaScript files. **Value type:** Boolean **Default:** `false`

`com.fortify.sca.skip.libraries.ES6`
`com.fortify.sca.skip.libraries.jQuery`
`com.fortify.sca.skip.libraries.javascript`
`com.fortify.sca.skip.libraries.typescript`
	Specifies a list of comma- or colon-separated JavaScript or TypeScript technology library files translated. You can use regular expressions in the file names. Note that the regular expressi `\d\.\d\.\d)?'` is automatically inserted before `.min.js` or `.js` for each file name included i `com.fortify.sca.skip.libraries.jQuery` property value.
	Value type: String
	Defaults:
	• ES6: `es6-shim.min.js,system-polyfills.js,shims_for_IE.js`
	• jQuery: `jquery.js,jquery.min.js, jquery-migrate.js,jquery-migrate.min.js, ui.js,jquery-ui.min.js, jquery.mobile.js,jquery.mobile.min.js, jquery.color.js,jquery.color.min.js, jquery.color.svg-names.js, jquery.co names.min.js, jquery.color.plus-names.js, jquery.color.plus-names.min.js, jquery.tools.min.js`
	• javascript: `bootstrap.js,bootstrap.min.js,typescript.js,typescriptServices.j`
	• typescript: `typescript.d.ts,typescriptServices.d.ts`

`com.fortify.sca.follow.imports`	If set to true, files included with an import statement are included in the translation. **Value type:** Boolean **Default:** `true`
`com.fortify.sca.exclude.node.modules`	If set to true, files in a node_modules directory are excluded from the analysis phase. **Value type:** Boolean **Default:** `true`
`com.fortify.sca.exclude.unimported.node.modules`	Specifies whether to exclude source code in a node_modules directory. If set to true, only im node_modules are included in the translation. **Note** This property is only applied if `com.fortify.sca.exclude.node.modules` is set false. **Value type:** Boolean **Default:** `true`

Translating JavaScript and TypeScript Code

1.29.2.8. Python properties

The properties for the `fortify-sca.properties` file in the following table apply to the translation of Python code.

Property name	Description
`com.fortify.sca.PythonPath`	Specifies a semicolon-separated (Windows) or colon-separated (non-Windows) list of additional import directories. OpenText SAST does not respect PYTHONPATH environment variable that the Python runtime system uses to find import files. Use this property to specify the additional import directories. **Value type:** String (path) **Default:** (none) **Command-line option:** `-python-path`
`com.fortify.sca.PythonVersion`	Specifies the Python source code version to scan. The valid values are `2` and `3`. **Value type:** Number **Default:** `3` **Command-line option:** `-python-version`
`com.fortify.sca.PythonNoAutoRootCalculation`	If set to true, disables the automatic calculation of a common root directory of all project files to use for importing modules and packages For more details, see Including Imported Modules and Packages. **Value type:** Boolean **Default:** `false` **Command-line option:** `-python-no-auto-root-calculation`
`com.fortify.sca.DjangoTemplateDirs`	Specifies semicolon-separated (Windows) or colon-separated (non-Windows) list of directories for Django templates. OpenText SAST does not use the TEMPLATE_DIRS setting from the Django `settings.py` file. **Value type:** String (paths) **Default:** (none) **Command-line option:** `-django-template-dirs`
`com.fortify.sca.DjangoDisableAutodiscover`	Specifies that OpenText SAST does not automatically discover Django templates. **Value type:** Boolean **Default:** (none) **Command-line option:** `-django-disable-autodiscover`
`com.fortify.sca.JinjaTemplateDirs`	Specifies semicolon-separated (Windows) or colon-separated (non-Windows) list of directories for Jinja2 templates. **Value type:** String (paths) **Default:** (none) **Command-line option:** `-jinja-template-dirs`
`com.fortify.sca.DisableTemplateAutodiscover`	Specifies that OpenText SAST does not automatically discover Django or Jinja2 templates. **Value type:** Boolean **Default:** (none) **Command-line option:** `-disable-template-autodiscover`

Translating Python Code

1.29.2.9. Go properties

The properties for the `fortify-sca.properties` file in the following table apply to the translation of Go code.

Property name	Description
`com.fortify.sca.gotags`	Specifies custom build tags for a Go project. This is equivalent to the `-tags` option for the `go` command. **Value type:** String **Default:** (none) **Command-line option:** -gotags
`com.fortify.sca.GOPATH`	Specifies the root directory of your project/workspace. **Value type:** String **Default:** (GOPATH system environment variable)
`com.fortify.sca.GOROOT`	Specifies the location of the Go installation. **Value type:** String **Default:** (GOROOT system environment variable)
`com.fortify.sca.GOPROXY`	Specifies one or more comma-separated proxy URLs. You can also specify `direct` or `off`. **Value type:** String **Default:** (GOPROXY system environment variable)

See Also

Translating Go Code

1.29.2.10. Ruby properties

The properties for the `fortify-sca.properties` file in the following table apply to the translation of Ruby code.

Property name	Description
`com.fortify.sca.RubyLibraryPaths`	Specifies one or more paths to directories that contain Ruby libraries. **Value type:** String (path) **Default:** (none) **Command-line option:** `-ruby-path`
`com.fortify.sca.RubyGemPaths`	Specifies one or more paths to RubyGems locations. Set this value if the project has associated gems to scan. **Value type:** String (path) **Default:** (none) **Command-line option:** `-rubygem-path`

Translating Ruby Code

1.29.2.11. COBOL properties

The properties for the `fortify-sca.properties` file in the following table apply to the translation of COBOL code.

Property name	Description
`com.fortify.sca.CobolCopyDirs`	Specifies one or more semicolon- or colon-separated directories where OpenText SAST looks for copybook files. **Value type:** String (path) **Default:** (none) **Command-line option:** `-copydirs`
`com.fortify.sca.CobolDialect`	Specifies the COBOL dialect. The valid values for dialect are `COBOL390` or `MICROFOCUS`. The dialect value is case-insensitive. **Value type:** String **Default:** COBOL390 **Command-line option:** `-dialect`
`com.fortify.sca.CobolCheckerDirectives`	Specifies one or more semicolon-separated COBOL checker directives. **Value type:** String **Default:** (none) **Command-line option:** `-checker-directives`
`com.fortify.sca.CobolLegacy`	If set to true, enables legacy COBOL translation. **Value type:** Boolean **Default:** `false` **Command-line option:** `-cobol-legacy`
`com.fortify.sca.CobolFixedFormat`	If set to true, specifies fixed-format COBOL to direct OpenText SAST to only look for source code between columns 8-72 in all lines of code (legacy COBOL translation only). **Value type:** Boolean **Default:** `false` **Command-line option:** `-fixed-format`
`com.fortify.sca.CobolCopyExtensions`	Specifies one or more semicolon- or colon-separated copybook file extensions (legacy COBOL translation only). **Value type:** String **Default:** (none) **Command-line option:** `-copy-extensions`

Translating COBOL Code

1.29.2.12. PHP properties

The properties for the `fortify-sca.properties` file in the following table apply to the translation of PHP code.

Property name	Description
`com.fortify.sca.PHPVersion`	Specifies the PHP version. For a list of valid versions, see Supported languages. **Value type:** String **Default:** `8.2` **Command-line option:** `-php-version`
`com.fortify.sca.PHPSourceRoot`	Specifies the PHP source root. **Value type:** Boolean **Default:** (none) **Command-line option:** `-php-source-root`

Translating PHP Code

1.29.2.13. ABAP properties

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

Property name	Description
`com.fortify.sca.AbapDebug`	If set to true, OpenText SAST adds ABAP statements to debug messages. **Value type:** Boolean **Default:** (none)
`com.fortify.sca.AbapIncludes`	When OpenText SAST encounters an ABAP 'INCLUDE' directive, it looks in the named directory. **Value type:** String (path) **Default:** (none)

1.29.2.14. Flex and ActionScript properties

The properties for the `fortify-sca.properties` file in the following table apply to the translation of Flex and ActionScript code.

Property name	Description
`com.fortify.sca.FlexLibraries`	Specifies a semicolon-separated (Windows) or colon-separated (non-Windows) of libraries to "link" to. This list must include `flex.swc`, `framework.swc`, and `playerglobal.swc` (which are usually located in the `frameworks/libs` directory in your Flex SDK root). Use this property primarily to resolve ActionScript. **Value type:** String (path) **Default:** (none) **Command-line option:** `-flex-libraries`
`com.fortify.sca.FlexSdkRoot`	Specifies the root location of a valid Flex SDK. The folder must contain a frameworks folder that contains a `flex-config.xml` file. It must also contain a `bin` folder that contains an `mxmlc` executable. **Value type:** String (path) **Default:** (none) **Command-line option:** `-flex-sdk-root`
`com.fortify.sca.FlexSourceRoots`	Specifies any additional source directories for a Flex project. Separate multiple directories with semicolons (Windows) or colons (non-Windows). **Value type:** String (path) **Default:** (none) **Command-line option:** `-flex-source-root`

1.29.2.15. ColdFusion (CFML) properties

The properties for the `fortify-sca.properties` file in the following table apply to the translation of CFML code.

Property name	Description
`com.fortify.sca.CfmlUndefinedVariablesAreTainted`	If set to true, OpenText SAST treats undefined variables in CFML pages as tainted. This serves as a hint to the Dataflow Analyzer to watch out for register-globals-style vulnerabilities. However, enabling this property interferes with dataflow findings where a variable in an included page is initialized to a tainted value in an earlier-occurring included page. **Value type:** Boolean **Default:** `false`
`com.fortify.sca.CaseInsensitiveFiles`	If set to true, make CFML files case-insensitive for applications developed using a case-insensitive file system and scanned on case-sensitive file systems. **Value type:** Boolean **Default:** (not enabled)
`com.fortify.sca.SourceBaseDir`	Specifies the base directory for ColdFusion projects. **Value type:** String (path) **Default:** (none) **Command-line option:** `-source-base-dir`

Translating ColdFusion Code

1.29.2.16. SQL properties

The properties for the `fortify-sca.properties` file in the following table apply to the translation of SQL code.

Property name	Description
`com.fortify.sca.SqlLanguage`	Specifies the SQL language variant. The valid SQL language type values are `PLSQL` (for Oracle PL/SQL) and `TSQL` (for Microsoft T-SQL). **Value type:** String **Default:** `TSQL` **Command-line option:** `-sql-language`

Translating SQL

1.29.2.17. Output properties

The properties for the `fortify-sca.properties` file in the following table apply to the analysis output.

Property name	Description
`com.fortify.sca.ResultsFile`	The file to which results are written. **Value type:** String **Default:** (none) **Command-line option:** `-f` **Example:** `com.fortify.sca.ResultsFile=MyResults.fpr`
`com.fortify.sca.Renderer`	Controls the output format. The valid values are `fpr`, `fvdl`, `text`, and `auto`. The default of `auto` selects the output format based on the extension of the file provided with the `-f` option. **Value type:** String **Default:** `auto` **Command-line option:** `-format`
`com.fortify.sca.OutputAppend`	If set to true, OpenText SAST appends results to an existing results file. **Value type:** Boolean **Default:** `false` **Command-line option:** `-append`
`com.fortify.sca.ResultsAsAvailable`	If set to true, OpenText SAST prints results as they become available. This is helpful if you do not specify the `-f` option (to specify an output file) and print to stdout. **Value type:** Boolean **Default:** `false`
`com.fortify.sca.BuildLabel`	Specifies a label for the scanned project. OpenText SAST does not use this label but includes it in the results. **Value type:** String **Default:** (none) **Command-line option:** `-build-label`
`com.fortify.sca.BuildProject`	Specifies a name for the scanned project. OpenText SAST does not use this name but includes it in the results. **Value type:** String **Default:** (none) **Command-line option:** `-build-project`
`com.fortify.sca.BuildVersion`	Specifies a version number for the scanned project. OpenText SAST does not use this version number but it is included in the results. **Value type:** String **Default:** (none) **Command-line option:** `-build-version`
`com.fortify.sca.MachineOutputMode`	Output information in a format that scripts or OpenText SAST tools can use rather than printing output interactively. Instead of a single line to display scan progress, a new line is printed below the previous one on the console to display updated progress. **Value type:** Boolean **Default:** (not enabled) **Command-line option:** `-machine-output`
`com.fortify.sca.SnippetContextLines`	Sets the number of lines of code to display surrounding an issue. Snippets always include the two lines of code on each side of the line where the error occurs. By default, five lines of code are displayed. **Value type:** Number **Default:** `2`
`com.fortify.sca.FVDLDisableDescriptions`	If set to true, excludes OpenText Application Security Content descriptions from the analysis results file (FVDL). **Value type:** Boolean **Default:** `false` **Command-line option:** `-fvdl-no-descriptions`
`com.fortify.sca.FVDLDisableEngineData`	If set to true, excludes engine data from the analysis results file (FVDL). **Value type:** Boolean **Default:** `false` **Command-line option:**`-fvdl-no-enginedata`
`com.fortify.sca.FVDLDisableLabelEvidence`	If set to true, excludes label evidence from the analysis results file (FVDL). **Value type:** Boolean **Default:** `false`

`com.fortify.sca.FVDLDisableProgramData`	If set to true, excludes the `ProgramData` section from the analysis results file (FVDL). **Value type:** Boolean **Default:** `false` **Command-line option:** `-fvdl-no-progdata`
`com.fortify.sca.FVDLDisableSnippets`	If set to true, excludes code snippets from the analysis results file (FVDL). **Value type:** Boolean **Default:** `false` **Command-line option:** `-fvdl-no-snippets`
`com.fortify.sca.FVDLStylesheet`	Specifies location of the style sheet for the analysis results. **Value type:** String (path) **Default:** `${com.fortify.Core}/resources/sca/fvdl2html.xsl`

1.29.2.18. Mobile build session (MBS) properties

The properties for the `fortify-sca.properties` file in the following table apply to MBS files.

Property name	Description
`com.fortify.sca.MobileBuildSessions`	If set to false, OpenText SAST does not copy source files into the build session directory. **Value type:** Boolean **Default:** `true`
`com.fortify.sca.ExtractMobileInfo`	If set to true, OpenText SAST extracts the build ID and the OpenText SAST version number from the mobile build session. **Note** OpenText SAST does not extract the mobile build with this property. **Value type:** Boolean **Default:** `false`

Mobile Build Sessions

1.29.2.19. Proxy properties

The properties for the `fortify-sca.properties` file in the following table apply to proxy settings.

Property name	Description
`com.fortify.sca.https.proxyHost`	Specifies a proxy host name. **Value type:** String **Default:** (none)
`com.fortify.sca.https.proxyPort`	Specifies a proxy port number. **Value type:** Number **Default:** (none)

1.29.2.20. Logging properties

The properties for the `fortify-sca.properties` file in the following table apply to log files.

Property name	Description
`com.fortify.sca.LogFile`	Specifies the default log file name and location. **Value type:** String (path) **Default:**`${com.fortify.sca.ProjectRoot}/log/sca.log`and `${com.fortify.sca.ProjectR` **Command-line option:** `-logfile`
`com.fortify.sca.LogLevel`	Specifies the minimum log level for both log files. The valid values are `DEBUG`, `INFO`, `WARN`, `ERR`(see Accessing Log Files and Configuring Log Files. **Value type:** String **Default:** `INFO`
`com.fortify.sca.ClobberLogFile`	If set to true, OpenText SAST overwrites the log file for each run of sourceanalyzer. **Value type:** Boolean **Default:** `false` **Command-line option:** `-clobber-log`
`com.fortify.sca.PrintPerformanceDataAfterScan`	If set to true, OpenText SAST writes performance-related data to the OpenText SAST Support l This value is automatically set to true when in debug mode. **Value type:** Boolean **Default:** `false`

Configuring Log Files

1.29.2.21. Debug properties

The properties for the `fortify-sca.properties` file in the following table apply to debug settings.

Property name	Description
`com.fortify.sca.Debug`	Includes debug information in the OpenText SAST Support log file, which is only useful for Customer Support to help troubleshoot. **Value type:** Boolean **Default:** `false` **Command-line option:** `-debug`
`com.fortify.sca.DebugVerbose`	This is the same as the `com.fortify.sca.Debug` property, but it includes more details, specifically for parse errors. **Value type:** Boolean **Default:** (not enabled) **Command-line option:** `-debug-verbose`
`com.fortify.sca.Verbose`	If set to true, includes verbose messages in the OpenText SAST Support log file. **Value type:** Boolean **Default:** `false` **Command-line option:** `-verbose`
`com.fortify.sca.DebugTrackMem`	If set to true, additional performance information is written to the OpenText SAST Support log. **Value type:** Boolean **Default:** (not enabled) **Command-line option:** `-debug-mem`
`com.fortify.sca.CollectPerformanceData`	If set to true, enables additional timers to track performance. **Value type:** Boolean **Default:** (not enabled)
`com.fortify.sca.Quiet`	If set to true, disables the command-line progress information. **Value type:** Boolean **Default:** `false` **Command-line option:** `-quiet`
`com.fortify.sca.MonitorSca`	If set to true, OpenText SAST monitors its memory use and warns when JVM garbage collection becomes excessive. **Value type:** Boolean **Default:** `true`

1.29.3. fortify-sca-quickscan.properties

OpenText SAST offers a less in-depth scan known as a quick scan. This option scans the project in quick scan mode, using the property values in the `fortify-sca-quickscan.properties` file. By default, a quick scan reduces the depth of the analysis and applies the Quick View filter set. The Quick View filter set provides only critical and high priority issues.

Note

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

The following table provides two sets of default values: the default value for quick scans and the default value for normal scans. If only one default value is shown, the value is the same for both normal scans and quick scans.

Property name	Description
`com.fortify.sca.CtrlflowMaxFunctionTime`	Sets the time limit (in milliseconds) for Control Flow analysis on a single function. **Value type:** Integer **Quick scan default:** `30000` **Default:** `600000`
`com.fortify.sca.DisableAnalyzers`	Specifies a comma- or colon-separated list of analyzers to disable during a scan. The valid analyzer names are `buffer`, `content`, `configuration`, `controlflow`, `dataflow`, `nullptr`, `semantic`, and `structural`. **Value type:** String **Quick scan default:** `controlflow:buffer` **Default:** (none)
`com.fortify.sca.FilterSet`	Specifies the filter set to use. You can use this property with an issue template to filter at scan-time instead of post-scan. See `com.fortify.sca.ProjectTemplate` described in Translation and Analysis Phase Properties to specify an issue template that contains the filter set to use. When set to `Quick View`, this property runs rules that have a potentially high impact and a high likelihood of occurring and rules that have a potentially high impact and a low likelihood of occurring. Filtered issues are not written to the FPR and therefore this can reduce the size of an FPR. For more information about filter sets, see the *OpenText™ Fortify Audit Workbench User Guide*. **Value type:** String **Quick scan default:** `Quick View` **Default:** (none)
`com.fortify.sca.FPRDisableMetatable`	Disables the creation of the metatable, which includes information for the Function view in Fortify Audit Workbench. This metatable enables right-click on a variable in the source window to show the declaration. If C/C++ scans take an extremely long time, setting this property to true can potentially reduce the scan time by hours. **Value type:** Boolean **Quick scan default:** `true` **Default:** `false` **Command-line option:** `-disable-metatable`
`com.fortify.sca.FPRDisableSourceBundling`	Disables source code inclusion in the FPR file. Prevents OpenText SAST from generating marked-up source code files during a scan. If you plan to upload FPR files that are generated as a result of a quick scan to Application Security, you must set this property to `false`. **Value type:** Boolean **Quick scan default:** `true` **Default:** `false` **Command-line option:** `-disable-source-bundling`
`com.fortify.sca.NullPtrMaxFunctionTime`	Sets the time limit (in milliseconds) for Null Pointer analysis for a single function. The standard default is five minutes. If this value is set to a shorter limit, the overall scan time decreases. **Value type:** Integer **Quick scan default:** `10000` **Default:** `300000`
`com.fortify.sca.TrackPaths`	Disables path tracking for Control Flow analysis. Path tracking provides more detailed reporting for issues, but requires more scan time. To disable this for JSP only, set it to `NoJSP`. Specify `None` to disable all functions. **Value type:** String **Quick scan default:** (none) **Default:** `NoJSP`
`com.fortify.sca.limiters.ConstraintPredicateSize`	Specifies the size limit for complex calculations in the Buffer Analyzer. Skips calculations that are larger than the specified size value in the Buffer Analyzer to improve scan time. **Value type:** Integer **Quick scan default:** `10000` **Default:** `500000`

`com.fortify.sca.limiters.MaxChainDepth`	Controls the maximum call depth through which the Dataflow Analyzer tracks tainted data. Increase this value to increase the coverage of dataflow analysis, which results in longer scan times. **Note** Call depth refers to the maximum call depth on a dataflow path between a taint source and sink, rather than call depth from the program entry point, such as `main()`. **Value type:** Integer **Quick scan default:** `3` **Default:** `5`
`com.fortify.sca.limiters.MaxFunctionVisits`	Sets the number of times taint propagation analyzer visits functions. **Value type:** Integer **Quick scan default:** `5` **Default:** `50`
`com.fortify.sca.limiters.MaxPaths`	Controls the maximum number of paths to report for a single dataflow vulnerability. Changing this value does not change the results that are found, only the number of dataflow paths displayed for an individual result. **Note** OpenText does not recommend setting this property to a value larger than `5` because it might increase the scan time. **Value type:** Integer **Quick scan default:** `1` **Default:** `5`
`com.fortify.sca.limiters.MaxTaintDefForVar`	Sets a complexity limit for the Dataflow Analyzer. Dataflow incrementally decreases precision of analysis on functions that exceed this complexity metric for a given precision level. This value controls how much taint is tracked for a variable chain. **Value type:** Integer **Quick scan default:** `250` **Default:** `1000`
`com.fortify.sca.limiters.MaxTaintDefForVarAbort`	Sets a hard limit for function complexity. If complexity of a function exceeds this limit at the lowest precision level, the analyzer skips analysis of the function. **Value type:** Integer **Quick scan default:** `500` **Default:** `4000`

1.29.4. fortify-rules.properties

This topic describes the properties available for use in the `fortify-rules.properties` file.

Improving Results

Use these properties to modify behavior of scan results, either enabling new sets of rules, filtering rules, or enabling correlation of results with OpenText DAST.

Property name	Description
`com.fortify.sca.rules.EnableRuleComments`	If set to true, enables the ability to prevent issues appearing in results using the `// FortifyRemove()` comments. For more information, see Filtering using FortifyRemove comments **Value Type:** Boolean **Default**: `true`
`com.fortify.sca.rules.IsLibrary`	If set to true, enables new entrypoint rules in code that adds WEB ,XSS, and PRIVATE taint to every public function variable (certain exclusions apply). (Currently only Java and JVM languages apply) **Value type:** Boolean **Default:** `false`
`com.fortify.sca.rules.enablePQCRules`	If set to true, enables rules to identify issues related to Post-Quantum Cryptographic threats. See security content updates and documentation for more details, including which languages and libraries are supported. **Value type:** Boolean **Default:** `false`

DAST Correlation & Verification

Property name	Description
`com.fortify.sca.rules.enable_wi_correlation`	If set to true and OpenText SAST scans an application with a supported framework, produces a results file to be imported into OpenText™ Dynamic Application Security Testing to improve results. **Value type:** Boolean **Default:** `false`

Google Cloud Function Integration

Scanning Google Cloud Functions either provide a JSON or YAML cloud build config file or set the properties in the below table to optimize results.

Property name	Description
`com.fortify.sca.rules.GCPFunctionName`	Name of the serverless function called when no JSON/YAML cloud build config file exists. **Value type:** String **Default:** (none)
`com.fortify.sca.rules.GCPHttpTrigger`	If set to true, the scanned cloud function is an HTTP trigger. **Value type:** Boolean **Default:** `false`

Properties to Customize Regular Expressions

Although many techniques are used to identify vulnerabilities in code, some rules have to rely upon regular expressions to try to find identifiers in code, and these can often be configured by properties. The following table describes a list of properties that can be used to modify the regular expressions used by the rules.

It is advised to set these within the `fortify-rules.properties` file instead of directly on the command line to prevent clashes between regular expression and shell syntax.

Property name	Description

com.fortify.sca.rules.password_regex.global	The regular expression to match password identifiers across all languages unless a language-specific rules property is set. **Value type:** String **Default:** (?i)(s|_)?(user|usr|member|admin|guest|login|default| new|current|old|client|server|proxy|sqlserver| my|mysql|mongo|mongodb|db|database|ldap|smtp| email|email(_)?smtp)?(_|\.)?(pass(wd|word|phrase)|secret)
com.fortify.sca.rules.password_regex.abap	Regular expression to match password identifiers in ABAP code. Setting this property overrides the global regex password rules property. **Value type:** String **Default:** (value for com.fortify.sca.rules.password_regex.global)
com.fortify.sca.rules.password_regex.actionscript	Regular expression to match password identifiers in ActionScript code. Setting this property overrides the global regex password rules property. **Value type:** String **Default:** (value for com.fortify.sca.rules.password_regex.global)
com.fortify.sca.rules.password_regex.apex	Regular expression to match password identifiers in Salesforce Apex code. Setting this property overrides the global regex password rules property. **Value type:** String **Default:** (value for com.fortify.sca.rules.password_regex.global)
com.fortify.sca.rules.password_regex.cfml	Regular expression to match password identifiers in ColdFusion (CFML) code. Setting this property overrides the global regex password rules property. **Value type:** String **Default:** (none)
com.fortify.sca.rules.password_regex.cobol	Regular expression to match password identifiers in COBOL code. Setting this property overrides the global regex password rules property. **Value type:** String **Default:** (value for com.fortify.sca.rules.password_regex.global)
com.fortify.sca.rules.password_regex.config	Regular expression to match password identifiers in XML. Setting this property overrides the global regex password rules property. Do not use regular expression modifiers. The value is case-insensitive. **Value type:** String **Default:** (s|_)?(user|usr|member|admin|guest|login|default| new|current|old|client|server|proxy|sqlserver| my|mysql|mongo|mongodb|db|database|ldap|smtp| email|email(_)?smtp)?(_|\.)?pass(wd|word|phrase)
com.fortify.sca.rules.password_regex.cpp	Regular expression to match password identifiers in C and C++ code. Setting this property overrides the global regex password rules property. **Value type:** String **Default:** (value for com.fortify.sca.rules.password_regex.global)
com.fortify.sca.rules.password_regex.dart	Regular expression to match password identifiers in Dart code. Setting this property overrides the global regex password rules property. **Value type:** String **Default:** (value for com.fortify.sca.rules.password_regex.global)
com.fortify.sca.rules.password_regex.dotnet	Regular expression to match password identifiers in .NET code. Setting this property overrides the global regex password rules property. **Value type:** String **Default:** (value for com.fortify.sca.rules.password_regex.global)
com.fortify.sca.rules.password_regex.docker	Regular expression to match password identifiers in Dockerfiles. Setting this property overrides the global regex password rules property. **Value type:** String **Default:** .*pass(wd|word|phrase).*
com.fortify.sca.rules.password_regex.golang	Regular expression to match password identifiers in Go code. Setting this property overrides the global regex password rules property. **Value type:** String **Default:** (value for com.fortify.sca.rules.password_regex.global)
com.fortify.sca.rules.password_regex.java	Regular expression to match password identifiers in Java code. Setting this property overrides the global regex password rules property. **Value type:** String **Default:** (value for com.fortify.sca.rules.password_regex.global)

com.fortify.sca.rules.password_regex.javascript	Regular expression to match password identifiers in JavaScript and TypeScript code. Setting this property overrides the global regex password rules property. **Value type:** String **Default:** (value for com.fortify.sca.rules.password_regex.global)
com.fortify.sca.rules.password_regex.json	Regular expression to match password identifiers in JSON. Setting this property overrides the global regex password rules property. **Value type:** String **Default:** (?i).*pass(wd|word|phrase).*
com.fortify.sca.rules.password_regex.jsp	Regular expression used to match password identifiers in JSP code. Setting this property overrides the global regex password rules property. **Value type:** String **Default:** (value for com.fortify.sca.rules.password_regex.global)
com.fortify.sca.rules.password_regex.objc	Regular expression to match password identifiers in Objective-C and Objective-C++ code. Setting this property overrides the global regex password rules property. **Value type:** String **Default:** (?i)(s|_)?(user|usr|member|admin|guest|login|default| new|current|old|client|server|proxy|sqlserver| my|mysql|mongo|mongodb|db|database|ldap|smtp| email|email(_)?smtp)?(_|\.)?(token|pin|pass(wd|word|phrase))
com.fortify.sca.rules.password_regex.php	Regular expression to match password identifiers in PHP code. Setting this property overrides the global regex password rules property. **Value type:** String **Default:** (value for com.fortify.sca.rules.password_regex.global)
com.fortify.sca.rules.password_regex.powershell	Regular expression to match password identifiers in PowerShell files. Setting this property overrides the global regex password rules property. **Value type:** String **Default:** (?i)([a-z_]*|\{.*)(pass(wd|word|phrase)|pwd)(.*\}|[a-z_]*)
com.fortify.sca.rules.password_regex.properties	Regular expression to match password identifiers in Properties files. Setting this property overrides the global regex password rules property. **Value type:** String **Default:** (value for com.fortify.sca.rules.password_regex.global)
com.fortify.sca.rules.password_regex.python	Regular expression to match password identifiers in Python code. Setting this property overrides the global regex password rules property. **Value type:** String **Default:** (value for com.fortify.sca.rules.password_regex.global)
com.fortify.sca.rules.password_regex.ruby	Regular expression to match password identifiers in Ruby code. Setting this property overrides the global regex password rules property. **Value type:** String **Default:** (value for com.fortify.sca.rules.password_regex.global)
com.fortify.sca.rules.password_regex.sql	Regular expression to match password identifiers in SQL code. Setting this property overrides the global regex password rules property. **Value type:** String **Default:** (value for com.fortify.sca.rules.password_regex.global)
com.fortify.sca.rules.password_regex.swift	Regular expression to match password identifiers in Swift code. Setting this property overrides the global regex password rules property. **Value type:** String **Default:** (?i)(s|_)?(user|usr|member|admin|guest|login|default| new|current|old|client|server|proxy|sqlserver| my|mysql|mongo|mongodb|db|database|ldap|smtp| email|email(_)?smtp)?(_|\.)?(token|pin|pass(wd|word|phrase))
com.fortify.sca.rules.password_regex.vb	Regular expression to match password identifiers in VB6 code. Setting this property overrides the global regex password rules property. **Value type:** String **Default:** (value for com.fortify.sca.rules.password_regex.global)
com.fortify.sca.rules.password_regex.yaml	Regular expression to match password identifiers in YAML. Setting this property overrides the global regex password rules property. **Value type:** String **Default:** (?i).*pass(wd|word|phrase).*

`com.fortify.sca.rules.key_regex.global`	The regular expression to match key identifiers across all languages unless a language-specific regex key rules property is set. **Value type:** String **Default:** `(?i)((enc|dec)(ryption|rypt)?|crypto|secret|private)(_)?key`
`com.fortify.sca.rules.key_regex.abap`	Regular expression to match key identifiers in ABAP code. Setting this property overrides the global regex key rules property. **Value type:** String **Default:** (value for `com.fortify.sca.rules.key_regex.global`)
`com.fortify.sca.rules.key_regex.actionscript`	Regular expression to match key identifiers in ActionScript code. Setting this property overrides the global regex key rules property. **Value type:** String **Default:** (value for `com.fortify.sca.rules.key_regex.global`)
`com.fortify.sca.rules.key_regex.cfml`	Regular expression to match key identifiers in CFML code. Setting this property overrides the global regex key rules property. **Value type:** String **Default:** (value for `com.fortify.sca.rules.key_regex.global`)
`com.fortify.sca.rules.key_regex.cpp`	Regular expression to match key identifiers in C and C++ code. Setting this property overrides the global regex key rules property. **Value type:** String **Default:** (value for `com.fortify.sca.rules.key_regex.global`)
`com.fortify.sca.rules.key_regex.golang`	Regular expression to match key identifiers in Go code. Setting this property overrides the global regex key rules property. **Value type:** String **Default:** (value for `com.fortify.sca.rules.key_regex.global`)
`com.fortify.sca.rules.key_regex.java`	Regular expression to match key identifiers in Java code. Setting this property overrides the global regex key rules property. **Value type:** String **Default:** (value for `com.fortify.sca.rules.key_regex.global`)
`com.fortify.sca.rules.key_regex.javascript`	Regular expression to match key identifiers in JavaScript and TypeScript code. Setting this property overrides the global regex key rules property. **Value type:** String **Default:** (value for `com.fortify.sca.rules.key_regex.global`)
`com.fortify.sca.rules.key_regex.jsp`	Regular expression to match key identifiers in JSP code. Setting this property overrides the global regex key rules property. **Value type:** String **Default:** (value for `com.fortify.sca.rules.key_regex.global`)
`com.fortify.sca.rules.key_regex.objc`	Regular expression used to match key identifiers in Objective-C and Objective-C++ code. Setting this property overrides the global regex key rules property. **Value type:** String **Default:** (value for `com.fortify.sca.rules.key_regex.global`)
`com.fortify.sca.rules.key_regex.php`	Regular expression to match key identifiers in PHP code. Setting this property overrides the global regex key rules property. **Value type:** String **Default:** (value for `com.fortify.sca.rules.key_regex.global`)
`com.fortify.sca.rules.key_regex.python`	Regular expression to match key identifiers in Python code. Setting this property overrides the global regex key rules property. **Value type:** String **Default:** (value for `com.fortify.sca.rules.key_regex.global`)
`com.fortify.sca.rules.key_regex.ruby`	Regular expression used to match key identifiers in Ruby code. Setting this property overrides the global regex key rules property. **Value type:** String **Default:** (value for `com.fortify.sca.rules.key_regex.global`)
`com.fortify.sca.rules.key_regex.sql`	Regular expression to match key identifiers in SQL code. Setting this property overrides the global regex key rules property. **Default:** (value for `com.fortify.sca.rules.key_regex.global`)
`com.fortify.sca.rules.key_regex.swift`	Regular expression used to match key identifiers in Swift code. Setting this property overrides the global regex key rules property. **Value type:** String **Default:** (value for `com.fortify.sca.rules.key_regex.global`)
`com.fortify.sca.rules.key_regex.vb`	Regular expression to match key identifiers in Visual Basic 6 code. Setting this property overrides the global regex key rules property. **Value type:** String **Default:** (value for `com.fortify.sca.rules.key_regex.global`)

1.30. Command-line tools

OpenText SAST command-line tools enable you to manage OpenText Application Security Content, perform post-installation configurations, and monitor scans. These tools are located in `<sast_install_dir>/bin`. The tools for Windows are provided as `.bat` or `.cmd` files. The following table describes the command-line tools installed with OpenText SAST.

Note

By default, log files for OpenText SAST tools are written to the following directory:

- Windows: `C:\Users\<username>\AppData\Local\Fortify\<tool_name>-<version>\log`
- Non-Windows: `<userhome>/.fortify/<tool_name>-<version>/log`

Tool	Description	More information
fortifyupdate	Compares installed security content to the current version and makes any required updates	About updating OpenText Application Security Content
FPRUtility	With this tool you can: • Merge audited projects • Verify FPR signatures • Display information from an FPR file • Combine or split source code files and audit projects into FPR files • Alter an FPR	*OpenText™ Application Security Tools Guide*
scapostinstall	This tool enables you to migrate properties files from a previous version of OpenText SAST, specify a locale, and specify a proxy server for security content updates and for Application Security.	Running the post-install tool
SCAState	Provides state analysis information on the JVM during the analysis phase	Checking the scan status with SCAState

This section contains the following topics:

1.30.1. About updating OpenText Application Security Content

You can use the fortifyupdate command-line tool to download the latest Fortify Secure Coding Rulepacks and metadata from OpenText.

The fortifyupdate tool gathers information about the existing security content in your OpenText SAST installation and contacts the Fortify Rulepack update server with this information. The server returns new or updated security content, and removes any obsolete security content from your OpenText SAST installation. If your installation is current, a message is displayed to that effect.

This section contains the following topics:

1.30.1.1. Updating OpenText Application Security Content

Use the fortifyupdate command-line tool to either download security content or import a local copy of the security content. This tool is located in the `<sast_install_dir>/bin` directory.

The default read timeout for this tool is 180 seconds. To change the timeout setting, add the `rulepackupdate.SocketReadTimeoutSeconds` property in the `server.properties` configuration file. For more information, see the *OpenText™ Application Security Tools Guide*.

The basic command-line syntax for fortifyupdate is shown in the following example:

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

To update your OpenText SAST installation with the latest Fortify Secure Coding Rulepacks and external metadata from the Fortify Rulepack update server, type the following command:

```
fortifyupdate
```

To update security content from the local system:

```
fortifyupdate -import <my_local_rules>.zip
```

To update security content from a Application Security server using credentials:

```
fortifyupdate -url <ssc_url> -sscUser <username> -sscPassword <password>
```

1.30.1.2. fortifyupdate command-line options

The following table describes the fortifyupdate options.

fortifyupdate option	Description
`-acceptKey`	Specifies to accept the public key. When this is specified, you are not prompted to provide a public key. Use this option to accept the public key if you update OpenText Application Security Content from a non-standard location with the `-url` option.
`-acceptSSLCertificate`	Specifies to use the SSL certificate provided by the server.
`-import <file>.zip`	Imports the ZIP file that contains security content. By default, Rulepacks are imported into the `<sast_install_dir>/Core/config/rules` directory.
`-coreDir <dir>`	Specifies a core directory where fortifyupdate stores the update. If this is not specified, the fortifyupdate performs the update in the *<sast_install_dir>*. **Important** Make sure that you copy the contents of the `<sast_install_dir>/config/keys` folder and paste it to a `config/keys` folder in this directory before you run `fortifyupdate`.
`-includeMetadata`	Specifies to only update external metadata.
`-includeRules`	Specifies to only update Rulepacks.
`-locale <locale>`	Specifies a locale. English is the default if no security content exists for the specified locale. The valid values are: • `en` (English) • `es` (Spanish) • `ja` (Japanese) • `ko` (Korean) • `pt_BR` (Brazilian Portuguese) • `zh_CN` (Simplified Chinese) • `zh_TW` (Traditional Chinese) **Note** The values are **not** case-sensitive. Alternatively, you can specify a default locale for security content updates in the `fortify.properties` configuration file. For more information, see the *OpenText™ Application Security Tools Guide*.
`-proxyhost <host>`	Specifies a proxy server network name or IP address.
`-proxyport <port>`	Specifies a proxy server port number.
`-proxyUsername <username>`	Specifies a user name if the proxy server requires authentication.
`-proxyPassword <password>`	Specifies the password if the proxy server requires authentication.
`-showInstalledRules`	Displays the currently installed Rulepacks including any custom rules and custom metadata.
`-showInstalledExternalMetadata`	Displays the currently installed external metadata.
`-url <url>`	Specifies a URL from which to download the security content. The default URL is `https://update.fortify.com` or the value set for the `rulepackupdate.server` property in the `server.properties` configuration file. For more information about the `server.properties` configuration file, see the *OpenText™ Application Security Tools Guide*. You can download the security content from a Application Security server by providing a Application Security URL.
Specify one of the following types of credentials if you update security content from Application Security with the `-url` option:	
`-sscUsername` `-sscPassword`	Specifies a Application Security user account by user name and password.
`-sscAuthToken`	Specifies a Application Security authentication token of type UnifiedLoginToken, CIToken, or ToolsConnectToken.

1.30.2. Checking the scan status with SCAState

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

To check the state:

1. Start a scan.
2. Open another command window.
3. Type the following at the command prompt:

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

See Also

SCAState command-line options

1.30.2.1. SCAState command-line options

The following table describes the SCAState options.

SCAState option	Description
`-a` | `--all`	Displays all available information.
`-debug`	Displays information that is useful to debug SCAState behavior.
`-ftd` | `--full-thread-dump`	Prints a thread dump for every thread.
`-h` | `--help`	Displays the help information for the SCAState tool.
`-hd <filename>` | `--heap-dump <filename>`	Specifies the file to which the heap dump is written. The file is interpreted relative to the remote scan's working directory; this is not necessarily the same directory where you are running SCAState.
`-liveprogress`	Displays the ongoing status of a running scan. This is the default. If possible, this information is displayed in a separate terminal window.
`-nogui`	Causes the OpenText SAST state information to display in the current terminal window instead of in a separate window.
`-pi` | `--program-info`	Displays information about the source code being scanned, including how many source files and functions it contains.
`-pid <process_id>`	Specifies the currently running OpenText SAST process ID. Use this option if there are multiple OpenText SAST processes running simultaneously. To obtain the process ID on Windows systems: 1. Open a command window. 2. At the command prompt, type `tasklist`. A list of processes is displayed. 3. Find the `java.exe` process in the list and note its PID. To find the process ID on Linux systems: • At the command prompt, type `ps aux | grep sourceanalyzer`.
`-progress`	Displays scan information up to the point at which the command is issued. This includes the elapsed time, the current phase of the analysis, and the number of results already obtained.
`-properties`	Displays configuration settings (this does not include sensitive information such as passwords).
`-scaversion`	Displays the OpenText SAST version number for the sourceanalyzer that is currently running.
`-td` | `--thread-dump`	Prints a thread dump for the main scanning thread.
`-timers`	Displays information from the timers and counters that are instrumented in OpenText SAST.
`-version`	Displays the SCAState version.
`-vminfo`	Displays the following statistics that JVM standard MXBeans provides: ClassLoadingMXBean, CompilationMXBean, GarbageCollectorMXBeans, MemoryMXBean, OperatingSystemMXBean, RuntimeMXBean, and ThreadMXBean.
<none>	Displays scan progress information (this is the same as `-progress`).

Note

OpenText SAST writes Java process information to the location of the TMP system environment variable. On Windows systems, the TMP system environment variable location is `C:\Users\<username>\AppData\Local\Temp`. If you change this TMP system environment variable to point to a different location, SCAState cannot locate the `sourceanalyzer` Java process and does not return the expected results. To resolve this issue, change the TMP system environment variable to match the new TMP location. OpenText recommends that you run SCAState as an administrator on Windows.

opentext™