OpenText™ Fortify SAST

User Guide

Version 26.4

© Copyright 2026 Open Text

Table of Contents

User Guide 10
1 Change log 10
2 Introduction 15
2.1 OpenText Fortify SAST 15
2.1.1 About the analyzers 16
2.2 Licensing 18
2.3 Renewing an expired license 18
2.4 OpenText Application Security Content 19
2.5 Fortify ScanCentral SAST 19
2.6 OpenText Application Security Tools 19
2.7 Sample projects 20
2.8 Related documents 20
3 System requirements 22
3.1 Hardware requirements 22
3.1.1 Sample scans 24
3.2 Supported platforms and architectures 26
3.3 Software requirements 26
3.4 AI-powered SAST requirements 28
3.4.1 Configurable LLMs 28
3.4.2 Database requirements 29
3.5 Language compatibility 30
3.5.1 Libraries, frameworks, and technologies 33
3.5.2 Language compatibility for AI-powered SAST 39
3.6 Supported build tools 40
3.7 Supported compilers 41
3.8 OpenText Application Security Content 42
3.9 Virtual Machine support 42
3.10 Acquiring software 42
3.11 Verifying software downloads 46
4 Installing Fortify SAST 46
4.1 About installing Fortify SAST 46
4.1.1 Installing Fortify SAST 47
4.1.2 Installing Fortify SAST silently 48
4.1.3 Installing Fortify SAST in text-based mode on non-Windows platforms 50
4.1.4 Manually installing OpenText Application Security Content 51
4.2 Using Docker to install and run Fortify SAST 51
4.2.1 Creating a Dockerfile to install OpenText SAST 51
4.2.2 Running the container 52
4.3 Upgrading Fortify SAST 53
4.4 About uninstalling Fortify SAST 53
4.4.1 Uninstalling Fortify SAST 54
4.4.2 Uninstalling Fortify SAST silently 54

4.4.3 Uninstalling Fortify SAST in text-based mode on non-Windows platforms 54
4.5 Post-installation tasks 55
4.5.1 Running the post-install tool 55
4.5.2 Migrating properties files 55
4.5.3 Specifying a locale 55
4.5.4 Configuring Fortify Security Content updates 56
4.5.5 Configuring the connection to Application Security 56
4.5.6 Removing proxy server settings 57
4.5.7 Adding trusted certificates 57
5 Analysis process overview 57
5.1 Scanning Basics 58
5.2 Translation phase 58
5.3 Analysis phase 59
5.4 Translation and analysis phase verification 59
5.5 Ranking issue severity 60
6 Analyzing using AI-powered SAST 61
6.1 Configuring the LLM 61
6.1.1 Using AWS Bedrock for AI-powered SAST 61
6.1.2 Using Azure for AI-powered SAST 62
6.1.3 Using Google Cloud for AI-powered SAST 63
6.1.4 Using OpenAI APIs for AI-powered SAST 63
6.2 Connecting to the database 64
6.3 Using the dbTool 66
6.4 Sample analysis using AI-powered SAST 67
6.5 AI-powered SAST configuration options 67
6.6 Rate limiting 81
6.7 Using the pwtool to encrypt sensitive values 81
7 Analyzing Java, Kotlin and JSP projects 82
7.1 Integrating with Gradle 82
7.1.1 Using Gradle integration 82
7.1.2 Troubleshooting Gradle integration 83
7.1.3 Using the Gradle plugin 83
7.2 Integrating with Maven 86
7.2.1 Installing and updating the Fortify Maven Plugin 86
7.2.2 Testing the Fortify Maven Plugin installation 86
7.2.3 Using the Fortify Maven Plugin 87
7.3 Integrating with Bazel 88
7.3.1 Java Bazel integration examples 88
7.4 Integrating with Ant 89
7.5 Manual Java and Kotlin translation syntax 89
7.5.1 Java, Kotlin and JSP command-line options 90
7.5.2 Java command-line examples 93
7.5.3 Kotlin command-line examples 93
7.6 Analyzing Kotlin scripts 94
7.7 Kotlin and Java translation interoperability 94

7.8 Handling Java warnings 94
7.9 Analyzing Jakarta EE (Java EE) applications 94
7.9.1 Translating Java files 95
7.9.2 Translating JSP projects, configuration files, and deployment descriptors 95
7.9.3 Jakarta EE (Java EE) translation warnings 95
7.10 Analyzing Java bytecode 95
7.11 Troubleshooting Java, Kotlin, and JSP translation and analysis issues 96
8 Analyzing Android projects 97
8.1 Android project translation prerequisites 97
8.2 Android code analysis command-line syntax 97
8.3 Filtering issues detected in Android layout files 97
9 Analyzing Groovy code 97
9.1 Groovy analysis prerequisites 98
9.2 Groovy translation syntax 98
10 Analyzing Scala code 98
11 Analyzing Visual Studio projects 99
11.1 Visual Studio project translation prerequisites 99
11.2 Visual Studio Project command-line syntax 99
11.3 Handling special cases for translating Visual Studio projects 100
11.3.1 Running translation from a script 100
11.3.2 Translating plain .NET and ASP.NET projects 100
11.3.3 Translating C/C++ and Xamarin projects 101
11.3.4 Translating projects with settings containing spaces 101
11.3.5 Translating a single project from a Visual Studio solution 101
11.3.6 Analyzing projects that build multiple executable files 101
11.4 Alternative ways to translate Visual Studio projects 101
11.4.1 Alternative translation options for Visual Studio solutions 101
11.4.2 Translating without explicitly running Fortify SAST 102
12 Analyzing JavaScript and TypeScript code 103
12.1 Translating pure JavaScript projects 104
12.2 Excluding dependencies 104
12.3 Excluding NPM Dependencies 104
12.4 NPM dependencies 105
12.4.1 Examples of excluding NPM dependencies 105
12.5 Translating JavaScript projects with HTML files 106
13 Analyzing Dart and Flutter code 107
13.1 Dart and Flutter translation prerequisites 107
13.2 Dart and Flutter command-line syntax 107
13.3 Dart and Flutter command-line examples 107
14 Analyzing Python and Jupyter Notebooks 108
14.1 Integrating with Bazel 108
14.1.1 Python Bazel integration examples 108
14.2 Python translation command-line syntax 109
14.2.1 Python command-line options 109
14.2.2 Python command-line examples 111

14.3 Translating Python in a virtual environment 111
14.4 Including imported modules and packages 112
14.5 Including namespace packages 112
14.6 Translating Django and Flask 112
15 Analyzing iOS and Xcode projects 113
15.1 iOS project translation prerequisites 113
15.2 iOS code analysis command-line syntax 113
16 Analyzing C and C++ code 114
16.1 C and C++ Code translation prerequisites 114
16.2 Integrating with Make 114
16.3 Integrating with CMake 115
16.4 Integrating with Gradle 115
16.5 Manual C and C++ translation syntax 116
16.6 C/C++ Precompiled Header Files 116
16.7 Using other compilers 116
17 Analyzing Rust code 117
17.1 Rust analysis prerequisites 118
17.2 Rust translation syntax 118
18 Analyzing Fortran code 118
18.1 Fortran analysis prerequisites 118
18.2 Fortran translation syntax 118
19 Analyzing AI-agent instructions files 119
19.1 AI-agent instructions files analysis prerequisites 119
19.2 AI-agent instructions files translation syntax 119
20 Analyzing Go code 119
20.1 Go command-line syntax 119
20.2 Go command-line options 120
20.3 Including custom Go build tags 122
20.4 Resolving dependencies 122
21 Analyzing PHP code 122
21.1 PHP command-line options 123
22 Analyzing Perl code 123
22.1 Perl analysis prerequisites 123
22.2 Perl translation syntax 123
23 Analyzing Ruby code 124
23.1 Ruby translation syntax 124
24 Analyzing Ada code 124
24.1 Ada analysis prerequisites 125
24.2 Ada translation syntax 125
25 Analyzing Delphi code 125
25.1 Delphi analysis prerequisites 125
25.2 Delphi translation syntax 125
26 Analyzing Elixir code 126
26.1 Elixir analysis prerequisites 126
26.2 Elixir translation syntax 126

27 Analyzing Erlang code 126
27.1 Erlang analysis prerequisites 127
27.2 Erlang translation syntax 127
28 Analyzing Lua code 127
28.1 Lua analysis prerequisites 127
28.2 Lua translation syntax 127
29 Analyzing Kornshell code 128
29.1 Kornshell analysis prerequisites 128
29.2 Kornshell translation syntax 128
30 Analyzing Salesforce Apex and Visualforce code 128
30.1 Apex and Visualforce translation prerequisites 128
30.2 Apex and Visualforce command-line syntax 129
31 Analyzing ABAP code 129
31.1 About downloading source files 130
31.1.1 INCLUDE processing 130
31.2 Importing the transport request 130
31.3 Adding Fortify SAST to your Favorites list 131
31.4 Running the Fortify ABAP Extractor 131
31.5 Uninstalling the Fortify ABAP Extractor 134
32 Analyzing COBOL code 134
32.1 Preparing COBOL source and copybook files for translation 135
32.2 COBOL command-line syntax 136
32.2.1 Translating COBOL source files without file extensions 136
32.2.2 Translating COBOL source files with arbitrary file extensions 136
32.2.3 COBOL command-line options 136
32.3 Using Legacy COBOL translation (deprecated) 137
32.3.1 Legacy COBOL translation command-line options 137
33 Analyzing SQL 138
33.1 PL/SQL command-line example 139
33.2 T-SQL command-line example 139
34 Analyzing Infrastructure as Code (IaC) 139
35 Analyzing JSON 141
36 Analyzing YAML 141
37 Analyzing Dockerfiles 141
38 Analyzing Bash code 141
38.1 Bash analysis prerequisites 142
38.2 Bash translation syntax 142
39 Analyzing PowerShell code 142
39.1 PowerShell analysis prerequisites 142
39.2 PowerShell translation syntax 142
40 Analyzing R code 143
40.1 R analysis prerequisites 143
40.2 R translation syntax 143
41 Analyzing Solidity code 143
42 Analyzing other languages and configurations 144

42.1 Analyzing Flex and ActionScript 144
42.1.1 Flex and ActionScript command-line options 144
42.1.2 ActionScript command-line examples 145
42.1.3 Handling resolution warnings 146
42.2 Analyzing ColdFusion code 146
42.2.1 ColdFusion command-line syntax 147
42.2.2 ColdFusion (CFML) command-line options 147
42.3 Analyzing ASP/VBScript virtual roots 147
42.4 Classic ASP command-line example 149
42.5 VBScript command-line example 149
43 Analyzing Library code 149
44 Scanning for Secrets 150
44.1 Regular Expression Analysis 150
45 Scanning for PQC issues 151
46 Optimizing results 151
46.1 Applying a scan policy to the analysis 151
46.2 Excluding issues with filter files 153
46.2.1 Filter file example 156
46.3 Using filter sets to exclude issues 157
46.4 Filtering using FortifyRemove comments 158
46.5 Fortify Java annotations 160
46.5.1 Dataflow annotations 161
46.5.2 Field and variable annotations 162
46.5.3 Other annotations 163
47 Optimizing performance 163
47.1 Antivirus software 164
47.2 Hardware considerations 164
47.3 Tuning options 165
47.4 Quick scan 166
47.5 Configuring scan speed with speed dial 167
47.6 Breaking down codebases 168
47.7 Limiting analyzers and languages 169
47.7.1 Disabling analyzers 169
47.7.2 Disabling languages 169
47.8 Optimizing FPR files 169
47.8.1 Using filter files 170
47.8.2 Using filter sets 170
47.8.3 Excluding source code from the FPR 170
47.8.4 Reducing the FPR file size 171
47.8.5 Opening large FPR files 172
47.9 Monitoring long running scans 172
47.9.1 Using the SCAState tool 173
47.9.2 Using JMX tools 173
47.9.2.1 Using JConsole 173
47.9.2.2 Using Java VisualVM 173

48 Using mobile build sessions 173
48.1 Mobile build session version compatibility 174
48.2 Creating a mobile build session 174
48.3 Importing a mobile build session 174
49 Troubleshooting 175
49.1 Exit codes 175
49.2 Memory tuning 176
49.2.1 Java heap exhaustion 176
49.2.2 Native heap exhaustion 177
49.2.3 Stack overflow 177
49.3 Scanning complex functions 177
49.3.1 Dataflow Analyzer limiters 178
49.3.2 Control Flow and Null Pointer analyzer limiters 179
49.4 Issue non-determinism 179
49.5 Locating the log files 179
49.6 Configuring log files 180
49.7 Reporting issues and requesting enhancements 181
50 Command-line reference 181
50.1 Specifying files and directories 181
50.2 Directives 183
50.2.1 LIM license directives 184
50.3 Translation options 186
50.4 Analysis options 189
50.5 Output options 193
50.6 Other options 196
51 Configuration options 198
51.1 Properties files 199
51.1.1 Properties file format 199
51.1.2 Overriding settings 199
51.2 fortify-sca.properties 200
51.2.1 Translation and analysis phase properties 201
51.2.2 Regex analysis properties 211
51.2.3 LIM license properties 212
51.2.4 Rule properties 214
51.2.5 Java and Kotlin properties 216
51.2.6 Visual Studio and MSBuild project properties 219
51.2.7 JavaScript and TypeScript properties 220
51.2.8 Python properties 222
51.2.9 Go properties 224
51.2.10 Ruby properties 224
51.2.11 COBOL properties 225
51.2.12 PHP properties 227
51.2.13 ABAP properties 227
51.2.14 Flex and ActionScript properties 227
51.2.15 ColdFusion (CFML) properties 228

51.2.16 SQL properties 229
51.2.17 Output properties 229
51.2.18 Mobile build session (MBS) properties 232
51.2.19 Proxy properties 232
51.2.20 Logging properties 233
51.2.21 Debug properties 233
51.3 fortify-sca-quickscan.properties 235
51.4 fortify-rules.properties 238
52 Command-line tools 247
52.1 About updating OpenText Application Security Content 248
52.1.1 Updating OpenText Application Security Content 248
52.1.2 fortifyupdate command-line options 249
52.2 Checking the scan status with SCAState 251
52.2.1 SCAState command-line options 251
53 Installation and upgrade notes 253

User Guide

This section provides instructions for using OpenText™ Fortify SAST to scan code on most major programming platforms. This section is intended for people responsible for security audits and secure coding.

1. Change log

The following table lists the changes made to this guide. Revisions are published between software releases when the changes affect product functionality.

Software release / Document version	Changes
26.4.0	Added: • New AlmaLinux, SUSE Linux Enterprise Server, and macOS versions. For more information, see Supported platforms and architecture • New Net (Core), C#, Go, Kotlin, Swift, and T-SQL server versions. For more information, see Language compatibility • Support for new version for gcc, Swift, LLVM Clang. For more information, see Supported compilers • New MSBuild, dotnet, and xcode versions. For more information, see Supported build tools • KornShell as a supported language for AI-powered SAST. For more information, see Analyzing Kornshell code • The `-regex-scan-minified` option and the `com.fortify.sca.regex.ScanMinifiedJS` property to scan minified JavaScript and TypeScript files during regex analysis. For more information, see Regex analysis properties • Support for configuring database connection properties through environment variables. For more information, see Connecting to the database • The `com.fortify.sca.ai.cache.missOnNewModel` property to regenerate cached LLM results when a different model is used. For more information, see AI-powered SAST configuration options Updated: • To use the legacy COBOL analyzer, specify the `-cobol-legacy` option or set the `com.fortify.sca.CobolLegacy=true` property. For more information, see Legacy COBOL translation • Google Cloud authentication guidance for AI-powered SAST to include Application Default Credentials (ADC). For more information, see Using Google Cloud for AI-powered SAST • Incorporated product name changes Removed: • Support for macOS 15, 26.3, 26.4 and CPU arch x64 • Support for .Net Framework 4.6, Java (with android) 7, and T-SQL server versions 2005, 2008, and 20012 • Support for Maven 3.8.x, xcode 16.x, Bazel 6.x • Support for compiler gcc 14 • Support for compiler LLVM Clang 17.0.6

26.3.0	Added: • New Claude models via Microsoft Azure for AI-powered SAST (see Supported LLMs) • Added Google Cloud as a supported provider for AI-powered SAST (see Supported LLMs and Using Google Cloud for AI-powered SAST) • Added support for running scans on Amazon EC2 instances that use IAM roles with the required permissions and specifying a Bedrock bearer token for AWS authentication (see Using AWS Bedrock for AI-powered SAST) • Added support for configuring the Azure OpenAI endpoint for Microsoft Foundry deployments by using the Azure Foundry resource name (see Using Azure for AI-powered SAST) • Added support for new Claude model names via Microsoft Azure for `com.fortify.sca.ai.model` property (see AI-powered SAST configuration options) • Added new `com.fortify.sca.ai.aws.inferenceProfile` and `com.fortify.sca.ai.aws.bearerTokenBedrock` properties in AWS vendor configuration properties (see AI-powered SAST configuration options) • Added new `com.fortify.sca.ai.azure.resource` property in Azure vendor configuration properties (see AI-powered SAST configuration options) • Added new versions for macOS® (see Supported platforms and architectures) • Added a section for file configuration properties (see AI-powered SAST configuration options) • Added new versions for Go and Swift (see Language compatibility) • Added new versions for Gradle (build integration) and xcodebuild (see Supported build tools) • Added new compiler versions for LLMV Clang, Apple Clang, and Swiftc (see Supported compilers) • Added page on using custom compilers when analyzing C/C++ code (see Using other compilers) • Added new extension types for `com.fortify.sca.fileextensions.<language>` property (see Translation and analysis phase properties) • Added new `com.fortify.LegacyHiddenGlobSyntax` property to avoid including files and directories whose names start with a period are included by default, even on platforms that treat these entries as hidden. (see

	Translation and analysis phase properties and Specifying files and directories) Updated: • Incorporated product name changes • Changed all mentions of *OpenText SAST* to Fortify SAST • Updated description for specifying an AWS inference profile ID or inference profile ARN in the `com.fortify.sca.ai.model` property (see AI-powered SAST configuration options) • Updated description for `-cp <paths>` | `-classpath <paths>` property (see Java, Kotlin and JSP command-line options) Removed: • Removed CentOS, Red Hat® Enterprise Linux® 7.2, Ubuntu® 20.04.1 LTS, Microsoft Windows® Server 2019 • Removed the `com.fortify.sca.FilterSet` property from fortify-sca-quickscan.properties

26.2.0	Added: • New LLM providers for AI-powered SAST (Supported LLMs) • Added support for analyzing COBOL, Fortran, and AI-agent instructions files using AI-powered SAST (see Language compatibility for AI-powered SAST) • Added Analyzing COBOL code, Analyzing Fortran code, and Analyzing AI-agent rules files sections. • A section for database requirements for using AI-powered SAST (Database requirements) • A section for Using Azure for AI-Powered SAST • A section for Using OpenAI APIs for AI-Powered SAST • Azure and OpenAI vendor configuration properties (see AI-powered SAST configuration options) • Azure and OpenAI sensitive properties in Using the pwtool to encrypt sensitive values • Added new versions for C++, Kotlin, Swift, PHP (see Language compatibility) • Added new versions for MSBuild and xcodebuild (see Supported build tools) • Added new versions for Microsoft Windows® and macOS® (see Supported platforms and architectures) Updated: • Added a caution related to Akka license file (see Analyzing Scala code) Removed: • Removed Microsoft Windows® version 10, macOS® version 14, and IBM® AIX® version 7.1 • Removed .NET (Core) versions 2.x - 4.x • Removed .NET Framework versions 2.0 - 4.5 • Removed C# version 5 • Removed Swift versions 5.10 - 6.0 • Removed MSBuild version 14.x • Removed xcodebuild versions 15.3, 15.4 • Removed local translation for Ruby (see Analyzing Ruby code) • The `com.fortify.sca.ruby.legacy.enabled` property was removed from the `fortify-sca.properties` file (see Ruby properties) • The topics *Ruby command-line options*, *Adding libraries*, and *Adding gem paths were removed from this document* The topic *Including external JavaScript or HTML in the translation* was removed from this document

	• The `com.fortify.sca.FilterSet` property in the `fortify-sca-quickscan.properties` file is deprecated and will be removed from the `fortify-sca-quickscan.properties` file in the next release
26.1.0	Added: • AI-powered SAST (see Analyzing using AI) • Added new Python version (see Language compatibility) • Added new xcodebuild version (see Supported build tools) • Added frameworks for languages supported for AI-assisted analysis (see Libraries, frameworks, and technologies) Removed: • Removed Ant version 1.9.x and Xcode build versions 15.3 - 15.4 (see Supported build tools)

2. Introduction

This section contains the following topics:

- OpenText Fortify SAST
- Licensing
- Renewing an expired license
- OpenText Application Security Content
- Fortify ScanCentral SAST
- OpenText Application Security Tools
- Sample projects
- Related documents

2.1. OpenText Fortify SAST

Fortify 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. Fortify 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 Fortify 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 Fortify Software Security Center. 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.

2.1.1. About the analyzers

Fortify 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, Fortify 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 Fortify 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.

Analyzer	Description
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.

2.2. Licensing

Fortify 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 Fortify 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 Fortify SAST. With a LIM managed concurrent license, multiple installations of Fortify SAST can share a single license. For information about how to set up the LIM with licenses for Fortify SAST, see *OpenText™ Fortify License and Infrastructure Manager Installation and Usage Guide*. For more information about managing your LIM license from Fortify SAST, see LIM license directives.

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

2.3. Renewing an expired license

The license for Fortify SAST expires annually.

To update an expired license:

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

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

2.4. OpenText Application Security Content

Fortify 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 Fortify SAST (see Installing Fortify 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 Fortify 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™ Fortify SAST also includes the Fortify SAST Rules Skill, which assists with creating, modifying, and validating custom rules and rule packs. The skill supports tasks such as developing dataflow, structural, control flow, configuration, and taint analysis rules, as well as authoring rule XML.

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.

2.5. Fortify ScanCentral SAST

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

You can analyze your code in one of two ways:

- If your application is written in a language supported for Fortify ScanCentral SAST translation, you can offload the translation and analysis (scan) phase of the analysis to Fortify ScanCentral SAST.
- Perform the translation phase on a local build machine and generate a mobile build session (MBS). Start the scan with Fortify 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 Fortify ScanCentral SAST, see the *OpenText™ Fortify ScanCentral SAST Installation, Configuration, and Usage Guide*.

2.6. OpenText Application Security Tools

OpenText provides applications and tools (including Secure Code Plugins) that integrate with Fortify SAST, Fortify ScanCentral SAST, and Fortify Software Security Center. 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 Fortify Software Security Center 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 Fortify ScanCentral SAST) and then optionally upload the results to Fortify Software Security Center.
BIRTReportGenerator ReportGenerator	Command-line tools to generate issue reports (BIRT) and legacy reports from FPR files.

2.7. 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 Fortify 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.

2.8. 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.

Fortify SAST

The following documents provide information about Fortify 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™ Fortify SAST User Guide sast-ugd-<*version*>.pdf	This document describes how to install and use Fortify SAST to scan code on many of the major programming platforms. It is intended for people responsible for security audits and secure coding.
OpenText™ Fortify SAST Custom Rules Guide sast-cr-ugd-<*version*>.zip	This document provides the information that you need to create custom rules for Fortify 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	

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 Fortify 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 Fortify Software Security Center.
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.

3. System requirements

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

- Hardware requirements
- Supported platforms and architectures
- Software requirements
- AI-powered SAST requirements
- Language compatibility
- Supported build tools
- Supported compilers
- OpenText Application Security Content
- Virtual Machine support
- Acquiring software
- Verifying software downloads

3.1. 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

Fortify SAST takes advantage of all CPU cores available on your system to reduce the scan time of large projects. When you run Fortify SAST, avoid running other CPU intensive processes during the Fortify 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 Fortify 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.

3.1.1. Sample scans

These sample scans metrics were collected using Fortify SAST version 26.3 on dedicated virtual machines with the Classic Scan Policy. These scans were run with OpenText Application Security Content 26.1 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	00:55	14:37	632	31843	Windows Server 2022 4 CPU cores, 32 GB RAM
ABAP	abap2UI5	00:11	00:48	11	59141	Linux (alma 9) 8 CPU cores, 64 GB RAM
C/C++	nasm 0.98.38	00:34	02:36	796	35997	Linux (Centos 7) with 8 CPUs and 32 GB of RAM
Java	WebGoat 8	00:14	01:05	457	23662	Linux (alma 9) 8 CPU cores, 64 GB RAM
Java	WordPress for Android	00:07	01:24	777	35377	Linux (alma 9) 8 CPU cores, 64 GB RAM
JavaScript	three.js	04:39	11.22	288	636355	Linux (Alma 9) with 4 CPU cores and 32 GB RAM
PHP	CakePHP	00:15	4:54	5190	136621	Linux (alma 9) 8 CPU cores, 64 GB RAM
PHP	phpBB3	01:03	03:04	1495	206802	Linux (alma 9) 8 CPU cores, 64 GB RAM
Python 3	numpy-1.13.3	1:56	07:14	430	563487	Linux (alma 9) 8 CPU cores, 64 GB RAM
Swift	MediaBrowser	00:09.39	00:20.2	25	17699	macOS with 4 CPUs and 16 GB of RAM

TypeScript	rxjs-7.8.1	02:14	07:19	56	204587	Linux (Alma 9) with 4 CPU cores and 32 GB RAM

3.2. Supported platforms and architectures

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

Operating system	Platforms	Distributions and versions	Notes
Microsoft Windows®	x64	Windows 11 Windows Server 2022, 2025	
Linux®	x64 ARM	• AlmaLinux 8.6, 9.x, 10.x • Red Hat® Enterprise Linux® 8.x (8.2 or later), 9.x, 10.x • SUSE® Linux® Enterprise Server 15, 16 • Ubuntu®22.04.1 LTS	
macOS®	M series ARM	26, 27	Xcode versions 26.2 and above require macOS® 15.6 and above.
IBM® AIX®	Power ISA	7.3	**Important** You must have the IBM XL C/C++ for AIX 16.1 Runtime environment package installed.

3.3. Software requirements

The Fortify SAST installation includes an embedded OpenJDK/JRE version 21, which the software requires. You do not need to install Java 21.

Note

Fortify SAST is tested and supported with the JRE bundled in the installation. While Fortify SAST may be compatible with other Java 21 JRE implementations, OpenText does not guarantee compatibility and does not provide support for issues that directly result from using a non-bundled JRE.

OpenText recognizes that organizations may need to upgrade their JRE to meet internal policies and security requirements. If you encounter issues when using a non-bundled JRE, revert to the bundled JRE to confirm whether the issue is specific to the JRE configuration before you contact Customer Support.

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

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

Language	Software	Operating systems	Translation Requirement	Scan Requirement
Visual Studio, MSBuild, or .NET projects	.NET Framework 4.8 or later (MSBuild only)	Windows	Yes	No
	.NET SDK 10.0	Windows, Linux	Yes	No
ABAP®/BSP	Fortify ABAP Extractor is supported on a system running ABAP Platform 2023 / ABAP Version 7.58.	All	No	No
Bicep	.NET SDK10.0	Windows, Linux	No	Yes
Scala	The Akka compiler plugin is available in the Maven Central Repository.	All	Yes	No
Solidity	The relevant Solidity compiler version If the scanning machine has an Internet connection, Fortify SAST will attempt to download the necessary Solidity compilers. In such cases, the Solidity compiler is not required on the scanning machine prior to the scan.	All	No	Yes

To use AI-assisted analysis, you must have a PostgreSQL database version 12.0 or later. For more information, see AI-powered SAST requirements.

3.4. AI-powered SAST requirements

This section describes the requirements for AI-powered SAST.

Requirements for using LLMs hosted by AWS Bedrock

- You must have an Amazon Web Services (AWS) account.
- The AWS IAM user or role that is used to make requests to AWS Bedrock must have permissions to call the following APIs:
  - `Bedrock Runtime: InvokeModel`
  - `Bedrock: GetInferenceProfile` (Only required if you use an application inference profile as the model ID)

For more information on AWS Identity and Access Management, see *AWS Documentation*.

Requirements for using LLMs hosted by Microsoft Foundry on Azure

- You must have an Azure account.
- You must create a Microsoft Foundry deployment.
- (Optional) If you are using Microsoft Entra ID authentication, the Microsoft Entra ID principal should be assigned the `Azure AI User` role for a scope that includes your Microsoft Foundry deployment.
  For more information on Azure and Microsoft Foundry, see *Azure Documentation*.

Requirements for using LLMs hosted by Agent Platform on Google Cloud

- You must have a Google Cloud account.
- You must give the relevant principle the `Agent Platform User` role.
- You must enable the desired model in Agent Platform's Model Garden.
- You must allow the `structured_outputs` feature for your desired model for the `vertexai.allowedPartnerModelFeatures` constraint.

Requirements for using LLMs hosted by OpenAI

- You must have an OpenAI account.
- You must create an API key.

Other requirements

- To enable caching of LLM results, you must have a PostgreSQL database version 12.0 or later.

> **Note**
>
> If you do not use a database for caching results, it could lead to increased LLM costs.

For more information on database requirements, see Database requirements.

- (Optional) Ensure you have the pwtool that installs with OpenText Application Security Tools if you want to obfuscate sensitive values before storing them in the `fortify-sca.properties` file or using them on the command line. For more information, see

Supported LLMs

Database requirements

3.4.1. Configurable LLMs

You may provide one of the following LLMs to Fortify SAST to analyze the source code files.

Provider	Model
AWS	Claude Sonnet 4.6 (verified) Claude Opus 4.6
Azure	Claude Sonnet 4.6 (verified) Claude Opus 4.6 GPT-6.0 GPT-5.4 GPT-5.2 GPT-5.2
Google Cloud	Claude Sonnet 4.6 (verified)
OpenAI	GPT-6.0 GPT-5.4 GPT-5.2

Important

Models labeled *verified* have been tested to meet OpenText quality requirements and ensure adequate result quality.

3.4.2. Database requirements

Consider the following database requirements.

- You must have a PostgreSQL database version 12.0 or later.
- You must create a PostgreSQL database and user.
- You can optionally create a schema or use the default `public` schema.
- The database user requires the following privileges:
  - CONNECT on the database
  - CREATE, USAGE on the database schema.

    Note

    The default schema is `public`. You can specify the schema through the `currentSchema` parameter in the JDBC connection URL.

    Example

    `jdbc:postgresql://host:5432/db?currentSchema=myschema`.

  - SELECT, INSERT, UPDATE, DELETE on all the tables in the schema.
  - USAGE on all the sequences in the schema.

Example: create user, database, and grant privileges

Consider the following example.

- Create a database user

Example

```
CREATE USER <user> WITH PASSWORD '<password>';
```

- Create and connect to the database

Example

```
CREATE DATABASE <database>;
\c <database>
```

- Optionally, create a custom schema or use the default `public` schema

Example

```
CREATE SCHEMA <schema>;
```

- Grant the required privileges

Example

```
GRANT CONNECT ON DATABASE <database> TO <user>;
GRANT CREATE, USAGE ON SCHEMA <schema> TO <user>;
GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA <schema> TO <user>;
GRANT USAGE ON ALL SEQUENCES IN SCHEMA <schema> TO <user>;
```

- If multiple database users connect to the database, configure default privileges so that each user can access objects created by other users

Note

```
ALTER DEFAULT PRIVILEGES FOR ROLE <other_user> IN SCHEMA <schema>
GRANT SELECT, INSERT, UPDATE, DELETE ON TABLES TO <user>;
ALTER DEFAULT PRIVILEGES FOR ROLE <other_user> IN SCHEMA <schema>
GRANT USAGE ON SEQUENCES TO <user>;
```

3.5. Language compatibility

Fortify 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 Fortify 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.

In addition to the list below, there is a set of languages that can utilize AI-powered analysis, which is compatible with all versions and frameworks for each language. See Language compatibility for AI-assisted analysis for further details.

Language / framework	Verified Compatibility
.NET (Core)	7.x-11.x (Preview)
.NET Framework	4.6.2–4.8.1
ABAP/BSP	6.x, 7.x
ActionScript	3.0
Apex	55–61
Bicep	0.12.x–0.15.31
C#	6–15 (Preview)
C	C11, C17, C23 (see Compilers)
C++	C++11, C++14, C++17, C++20, C++23 (see Compilers)
Classic ASP (with VBScript)	2.0, 3.0
COBOL	All dialects and versions supported via AI-Powered SAST. See Language compatibility for AI-assisted analysis for more information.
ColdFusion	8–10
Dart™	2.12-3.8
Docker® (Dockerfiles)	any
Go™ programming language	Up to 1.27
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)	1.8–25
JavaScript	ECMAScript® 2015–2024
JSON	ECMA-404
JSP	1.2, 2.1
Kotlin	1.3–2.4
MXML (Flex®)	4
Objective-C/C++	2.0 (see Compilers)
PHP	7.3–8.5
PL/SQL	8–23
Python®	2.6–3.14 Python in Jupyter Notebooks can also be scanned.
Ruby	All versions supported with AI-assisted analysis. See Language compatibility for AI-assisted analysis for more information.
Scala	2.11–2.13, 3.3–3.6
Solidity	0.4.12–0.8.21
Swift®	6.1 - 6.4 (see Compilers for supported swiftc versions)
T-SQL	SQL Server 2014, 2016, 2017, 2019, 2022, 2025
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

3.5.1. Libraries, frameworks, and technologies

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

> **Note**
> For AI-powered SAST languages, support for all libraries and frameworks is included by default.

Java

Adobe Flex Blaze DS	Apache Slide	iBatis	Mozilla Rhino	Spring AI
Ajanta	Apache Spring Security (Acegi)	IBM MQ	MyBatis	Spring MVC
Amazon Web Services (AWS) SDK	Apache Struts	IBM WebSphere	MyBatis-Plus	Spring Boot
Android	Apache Tapestry	Jackson	Netscape LDAP API	Spring Data Commons
Android Jetpack	Apache Tomcat	Jakarta Activation	OkHttp	Spring Data JPA
Apache Axiom	Apache Torque	Jakarta EE Core	OpenCSV	Spring Data MongoDB
Apache Axis	Apache Util	Jakarta EE (Java EE)	Oracle Application Development Framework (ADF)	Spring Data Redis
Apache Beam	Apache Velocity	Jasypt	Oracle BC4J	Spring for GraphQL
Apache Beehive NetUI	Apache Wicket	Java Annotations	Oracle JDBC	Spring HATEOAS
Apache Catalina	Apache Xalan	Java Core	Oracle OA Framework	Spring JMS
Apache Cocoon	Apache Xerces	Java Excel API	Oracle tcDataSet	Spring JMX
Apache Commons	ATG Dynamo	JavaMail	Oracle XML Developer Kit (XDK)	Spring Messaging
Apache ECS	Azure SDK	JAX-RS	OWASP Enterprise Security API (ESAPI)	Spring Security
Apache Hadoop	Bouncy Castle	JAXB	OWASP HTML Sanitizer	Spring Webflow
Apache HttpComponents	Castor	Jaxen	OWASP Java Encoder	Spring WebSockets
Apache Jasper	Display Tag	JBoss	Plexus Archiver	Spring WS
Apache Log4j	Dom4j	JDesktop	Realm	Stripes
Apache Lucene	Fortify Java Annotations	JDOM	Restlet	Sun JavaServer Faces (JSF)
Apache MyFaces	GDS AntiXSS	Jetty	SAP Web Dynpro	Tungsten
Apache OGNL	Google Cloud	JGroups	Saxon	Weblogic
Apache ORO	Google Dataflow	json-simple	SnakeYAML	WebSocket
Apache POI	Google Guava	JTidy Servlet	Spring	XStream
Apache SLF4J	Google Web Toolkit	JXTA		YamlBeans
	gRPC	JYaml		ZeroTurnaround ZIP
	Gson	J2EE Core		Zip4J
	Hibernate	Liferay Portal		
		MongoDB		

Kotlin

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

Kotlin Core

Scala

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

Akka HTTP Scala Core	Scala Play Scala Slick

.NET

.NET Framework, .NET Core, and .NET Standard ADO.NET Entity Framework ADODB Amazon Web Services (AWS) SDK ASP.NET Core ASP.NET MVC ASP.NET SignalR ASP.NET Web API Azure SDK	Castle ActiveRecord CsvHelper Dapper DB2 .NET Provider DotNetZip Entity Framework Core fastJSON gRPC Hot Chocolate	IBM Informix .NET Provider Json.NET Log4Net Microsoft .NET WebSockets Microsoft ApplicationBlocks Microsoft My Framework Microsoft Practices EnterpriseLibrary Microsoft Web Protection Library MongoDB	MySql .Net Connector 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) C Core	CURL Library GLib JNI	MySQL Netscape LDAP ODBC	OpenSSL POSIX Threads SQLite	Sun RPC WinAPI

C++

CPPCore Boost Smart Pointers MFC	STL WMI

SQL

SQLCore Microsoft T-SQL	Oracle ModPLSQL Oracle PL/SQL Core

PHP

ADOdb Advanced PHP Debugging CakePHP PHPCore 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 AngularJS Anthropic Apollo Server Bluebird child-process-promise	Express GraphQL.js Google Gemini Google Vertex AI Handlebars Helmet	iOS JavaScript Bridge JavaScript (ECMAScript)/HTML5 Core 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 Vue

Python & Jupyter Notebooks

aiopg Amazon Web Services (AWS) Lambda Amazon SageMaker Anthropic AutoGen Azure Functions Boto3 Django Faiss FastAPI	FastAPI-FastMCP FastMCP Flask Google Cloud Graphene gRPC httplib2 Hugging Face Jinja3 LangChain	libxml2 lxml memcache-client _mysql MySQL Connector/Python MySQLdb OpenAI oslo.config Pandas Paramiko	psycopg2 pycrypto PyCryptodome pycurl pylibmc PyMongo PySpark Python Core PyYAML requests	simplejson six Starlette TensorFlow Twisted Mail urllib3 WebKit

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 Network Apple os Apple Security Apple Social Apple UIKit Apple UserNotifications	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 CryptoKit Apple Foundation Apple HealthKit Apple LocalAuthentication Apple MessageUI Apple Network	Apple os Apple Security Apple Social Apple SwiftUI Apple System Apple UIKit Apple UserNotifications	Apple WatchConnectivity Apple WatchKit Apple WebKit Realm SQLite SSZipArchive Swift Core	Zip ZipArchive ZIPFoundation ZipUtilities ZipZap

Go

Go Core Gin GORM	logrus gRPC

Dart

Dart SDK Flutter

Salesforce Apex/Visualforce

Salesforce Apex/Visualforce Core

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			

3.5.2. Language compatibility for AI-powered SAST

The following languages are supported for AI-powered SAST.

- AI-agent rules file
- AI Instructions file
- Ada
- Bash
- COBOL*
- Delphi
- Elixir

- Erlang
- Fortran
- Groovy
- KornShell
- Lua
- Perl
- PowerShell
- R
- Ruby**
- Rust

*AI-powered SAST and traditional SAST are both available. OpenText recommends using AI-powered SAST for better results.

**AI-powered SAST has replaced traditional local SAST analysis.

3.6. Supported build tools

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

Build tool	Versions	Notes
Apache Ant™	1.10.x	
Bazel	7.x	Bazel integration supports Java and Python.
dotnet	7.x - 11.x (Preview)	
Gradle (build integration)	6.6–9.x	Fortify SAST Gradle integration supports Java, Kotlin, and C/C++.
Gradle (Gradle plugin)	6.6–8.5	Fortify SAST Gradle Plugin supports Java and Kotlin.
Apache Maven™ Software	3.9.x	
MSBuild	15.x–18.x	Fortify SAST MSBuild 18 integration is compatible with .NET 7.0 or later and .NET Framework 4.7.2 or later. While these versions have been tested, Fortify SAST is designed with flexibility in mind and may successfully scan versions of MSBuild not explicitly verified. We encourage users to upgrade to the latest version of Fortify SAST and attempt scans to determine compatibility. If you encounter issues with a newer or unverified version of MSBuild, please reach out to OpenText Support for assistance.
xcodebuild	26.0-27.0	

3.7. Supported compilers

Fortify SAST supports the compilers listed in the following table.

Compiler	Versions	Operating systems
gcc	GNU gcc 6.x–14	Windows, Linux, macOS, AIX
gcc	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
LLVM Clang	19.1.4, 19.1.5, 21.1.6	Windows, Linux, macOS, AIX
Apple Clang	17.0.0, 21.0.0	macOS
Swiftc	6.1 - 6.4	macOS

1Fortify SAST supports applications built in the following Xcode versions:

26.0-27.0

.

3.8. OpenText Application Security Content

Fortify Secure Coding Rulepacks are backward compatible with all supported Fortify SAST versions. This ensures that Rulepack updates do not break Fortify SAST installations that are currently supported.

3.9. 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.

3.10. Acquiring software

Fortify 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	Fortify SAST package for Windows This package includes: • Fortify SAST installer, which includes the following components • Fortify License and Infrastructure Manager installer • Fortify 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 Fortify SAST Windows package
OpenText_SAST_Fortify_Linux-ARM_<*version*>.tar.gz	Fortify SAST package for Linux on ARM This package includes: • Fortify SAST installer, which includes the following components • Fortify 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 Fortify SAST Linux on ARM package

OpenText_SAST_Fortify_Linux_<*version*>.tar.gz	Fortify SAST package for Linux This package includes: • Fortify SAST installer, which includes the following components • Fortify 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 Fortify SAST Linux package
OpenText_SAST_Fortify_Mac-ARM_<*version*>.tar.gz	Fortify SAST package for macOS-ARM This package includes: • Fortify SAST installer, which includes the following components • Fortify 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. • Contact Customer Support if you require an x86 installer.
OpenText_SAST_Fortify_Mac-ARM_<*version*>.tar.gz.sig	Signature file for the Fortify SAST macOS-ARM package
OpenText_SAST_Fortify_AIX_<*version*>.tar.gz	Fortify SAST package for AIX This package includes: • Fortify SAST installer • Fortify SAST Custom Rules Guide bundle • About OpenText Application Security Software Documentation
OpenText_SAST_Fortify_AIX_<*version*>.tar.gz.sig	Signature file for the Fortify SAST AIX package
OpenText_SAST_Fortify_Samples_<version>.zip	Code samples to help you learn to use Fortify SAST

OpenText_SAST_Fortify_Samples_<version>.zip.sig	Signature file for Fortify SAST code samples

3.11. 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

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>`

4. Installing Fortify SAST

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

- About installing Fortify SAST
- Using Docker to install and run Fortify SAST
- Upgrading Fortify SAST
- About uninstalling Fortify SAST
- Post-installation tasks

4.1. About installing Fortify SAST

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

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

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

> **Note**
> On non-Windows systems, you must install Fortify SAST as a user that has a home directory with write permission. Do not install Fortify 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 Fortify SAST by updating the installed configuration files. For information about the configuration options for Fortify SAST, see Configuration Options.

4.1.1. Installing Fortify SAST

To install Fortify SAST:

1. Run the installer file for your operating system to start the Fortify 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-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 Fortify SAST, and then click **Next**.
   If you selected to include Fortify 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 Fortify 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 Fortify SAST performs a task that requires a license, the application will attempt to acquire a LIM lease from the license pool. If Fortify 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 Fortify Software 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 Fortify Software Security Center 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 Fortify SAST artifact files. For more information, see About upgrading Fortify 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 **Fortify 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 Fortify SAST, any selected components, and Fortify Software 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.

4.1.2. Installing Fortify 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 Fortify SAST in the same directory where OpenText™ Application Security Tools is installed.

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

To install Fortify 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 Fortify Software 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 Fortify Software Security Content download, add the following lines:

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

    5. To enable download of Fortify Software 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 Fortify Software Security Center server and proxy information to obtain Fortify Software Security Content, a request to migrate from a previous release, and the location of the Fortify 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`

4.1.3. Installing Fortify 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 Fortify SAST in the same directory where OpenText™ Application Security Tools is installed.

To perform a text-based installation of Fortify 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`

4.1.4. 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.

4.2. Using Docker to install and run Fortify SAST

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

> **Note**
> You can only run Fortify SAST in Docker on supported Linux platforms.

- Creating a Dockerfile to install OpenText SAST
- Running the container

4.2.1. Creating a Dockerfile to install OpenText SAST

This topic describes how to create a Dockerfile to install Fortify 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 Fortify 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 Fortify 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 Fortify SAST silently.
3. Run the Fortify SAST installer using the RUN instruction.
   You must run the installer in unattended mode. For more information, see Installing Fortify SAST silently.
4. Run fortifyupdate to install the Fortify Software Security Content using the RUN instruction.

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

5. To configure the image so you can run Fortify 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 Fortify 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> "."
```

4.2.2. Running the container

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

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

To run the Fortify 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 Fortify 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 Fortify 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 Fortify 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 Fortify 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.

4.3. Upgrading Fortify SAST

To upgrade Fortify 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 Fortify 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.

4.4. About uninstalling Fortify SAST

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

This section contains the following topics:

- Uninstalling Fortify SAST

- Uninstalling Fortify SAST silently
- Uninstalling Fortify SAST in text-based mode on non-Windows platforms

4.4.1. Uninstalling Fortify SAST

To uninstall Fortify 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 Fortify SAST that you are uninstalling.
     The Fortify SAST (`sca<version>`) application settings folder is not removed.
   - Click **No** to retain the application settings on your system.

4.4.2. Uninstalling Fortify SAST silently

To uninstall Fortify 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 Fortify SAST that you are uninstalling.

4.4.3. Uninstalling Fortify SAST in text-based mode on non-Windows platforms

To uninstall Fortify 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`

4.5. Post-installation tasks

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

- Running the post-install tool
- Migrating properties files
- Specifying a locale
- Configuring Fortify Security Content updates
- Configuring the connection to Application Security
- Removing proxy server settings
- Adding trusted certificates

4.5.1. Running the post-install tool

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

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`.

4.5.2. Migrating properties files

To migrate properties files from a previous version of Fortify 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.

4.5.3. Specifying a locale

English is the default locale for an Fortify SAST installation.

To change the locale for your Fortify 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)

4.5.4. Configuring Fortify Security Content updates

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

To specify settings for Fortify 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 Fortify Software 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`).

4.5.5. Configuring the connection to Application Security

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

To specify settings for connecting to Fortify Software Security Center:

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 Fortify Software Security Center 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 Fortify Software Security Content from your Fortify Software Security Center server (option `6`)
   - The Fortify Software Security Center user name (option `7`)

4.5.6. Removing proxy server settings

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

To remove the proxy settings for obtaining Fortify Software Security Content updates or connecting to Fortify Software Security Center:

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.

4.5.7. Adding trusted certificates

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

- Fortify 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 Fortify Software Security Content) to update Fortify Software Security Content.
- Fortify SAST configured as a Fortify ScanCentral SAST sensor uses an HTTPS connection to communicate with the Controller.

When using HTTPS, Fortify 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 Fortify SAST needs to trust connections where the server presents a certificate issued by this CA, you must configure Fortify SAST to trust the CA. Otherwise, the use of HTTPS connections might fail.

You must add the trusted certificate of the CA to the Fortify SAST keystore. The Fortify 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 Fortify 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**.

5. Analysis process overview

- Scanning Basics
- Translation phase

- Analysis phase
- Translation and analysis phase verification
- Ranking issue severity

5.1. Scanning Basics

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

1. Remove all existing Fortify 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™ Fortify ScanCentral SAST Installation, Configuration, and Usage Guide*.

5.2. 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 Fortify SAST using the `sourceanalyzer` command. Fortify 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. Fortify 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 Fortify 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.

5.3. 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, Fortify 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
```

To analyze code using AI-assisted analysis, you should set up a PostgreSQL database to configure the LLM connection and configure the database in the `fortify-sca.properties` file. Alternatively, these properties can be specified at the command line, but it may incur additional costs as caching of results is avoided. For more information on AI-assisted analysis, see Analyzing using AI.

If you are analyzing languages that require AI-assisted analysis and do not configure the LLM or database connection, those files will only be scanned via Secret Scanning.

5.4. 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 Fortify 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
- Fortify 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*.

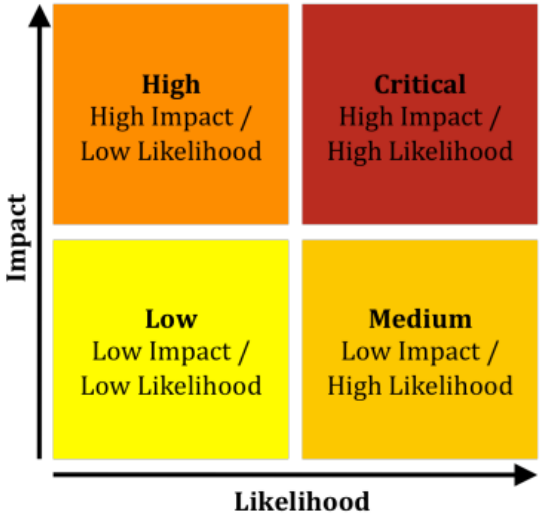
5.5. Ranking issue severity

Fortify SAST uses the Fortify Priority Order to calculate which severity each issue should be calculated.

Fortify SAST uses the following default severity buckets:

- Critical
- High
- Medium
- Low

Fortify SAST applies a four-quadrant model that evaluates issues based on their Impact and Likelihood, as illustrated in the following diagram.



An issue is classified as **Critical, High, Medium, or Low** based on the **Impact** and **Likelihood**.

Impact is a score assigned by the expert research team, reflecting the current security landscape.

Likelihood is calculated using the following formula:

`Likelihood = (Accuracy × Probability × Confidence) / 25`

To understand how Likelihood is determined, it is important to understand **Accuracy**, **Probability**, and **Confidence**:

Accuracy

The research team assigns an accuracy score to each rule when it is created. This score may increase or decrease in later versions as new techniques for identifying vulnerabilities are introduced.
Accuracy ranges from `1.0 to 5.0`.

Probability

The probability score reflects the answer to:
"If this issue is real, how likely is it that an attacker could exploit it?"
This value is provided by the research team and may change over time as new exploitation methods or defensive mechanisms emerge in the industry.
Probability ranges from `1.0 to 5.0`.

Confidence

The confidence score is generated by Fortify SAST to indicate how reliable the modeling is and how many assumptions were needed to identify the issue, assuming the rules are accurate. The calculation method varies across analyzers and may evolve as analysis engines improve. For more information, see About the analyzers.
Confidence ranges from `1.0 to 5.0`.

6. Analyzing using AI-powered SAST

Fortify SAST uses AI-powered SAST to analyze source files of a subset of languages by passing them to a configured Large Language Model (LLM) along with guidance and rules to identify dozens of vulnerabilty categories.

AI-powered SAST supplements existing analysis and enables scanning of languages that would otherwise require significantly more time to provide native support.

Off-cloud users must provision their own LLM instance and configure it through a PostgreSQL database that they maintain independently. For more information, see AI-assisted Analysis Requirements and Supported LLMs.

This section contains the following topics:

- Configuring the LLM
- Connecting to the database
- Using the dbTool
- Sample analysis using AI-powered SAST
- AI-powered SAST configuration options
- Rate limiting
- Using the pwtool to encrypt sensitive values

6.1. Configuring the LLM

Enable AI-powered SAST by specifying the provider and model.

```
com.fortify.sca.ai.provider=<provider>
com.fortify.sca.ai.model=<model, deployment, or inference profile identifier>
```

Example

```
com.fortify.sca.ai.provider=aws
com.fortify.sca.ai.model=us.anthropic.claude-sonnet-4-5-20250929-v1:0
```

For more information about supported LLM providers and models, see LLM request configuration and Supported LLMs.

For more detailed information on configuring access to specific LLM providers, see the following topics:

- Using AWS Bedrock for AI-powered SAST
- Using Azure for AI-powered SAST
- Using Google Cloud for AI-powered SAST
- Using OpenAI APIs for AI-powered SAST

6.1.1. Using AWS Bedrock for AI-powered SAST

Fortify SAST uses the AWS SDK for Java (version 2) to send API requests to AWS Bedrock. The AWS SDK requires credentials to authenticate and authorize these requests. You can provide credentials in several ways.

- **SDK Default Provider Chain**
  The SDK will attempt to automatically discover credentials using the default credential provider chain. Credentials can be provided through any of the credential provider methods in the default credential provider chain. For more information, see *AWS Documentation*.

Credential provider methods include but are not limited to:

  - **Credentials file:**
    You can specify the AWS access key, AWS secret key, and optional AWS session token in the AWS `credentials` file. The `credentials` file is located at `<HOME_DIR>/.aws/credentials` on Linux or macOS, or at `C:\Users\USERNAME\.aws\credentials` on Windows. For more information on configuring the `credentials` file, see *AWS Documentation*.
  - **Configure environment variables**
    You can specify the AWS access key, AWS secret key, and optional AWS session token as system environment variables. For more information on configuring the environment variables, see *AWS Documentation*.
  - **IAM Role for EC2 Instances**
    You can run the scan on an EC2 instance with an IAM role that has the appropriate permissions.
- **Bedrock bearer token**
  You can provide a Bedrock bearer token (API key) using the `com.fortify.sca.ai.aws.bearerTokenBedrock` property or by setting the `AWS_BEARER_TOKEN_BEDROCK` system environment variable.
- **Store the credentials in the PostgreSQL database**

You can store the AWS access key, AWS secret key, and optional AWS session token directly in the provided database by using the dbTool. For more information, see Using the dbTool.

- **Add the credentials in the `fortify-sca.properties` file**

The AWS access key, AWS secret key, and optional AWS session token can be explicitly set as Fortify properties in the `fortify-sca.properties` file. You can also specify the property on the command line with the `-D` option. For more information about the specific properties, see AWS configuration properties.

-

6.1.2. Using Azure for AI-powered SAST

Fortify SAST makes HTTP requests to Azure.

Authentication

There are multiple ways to authenticate.

- **Microsoft Entra ID**
  Fortify SAST uses the `DefaultAzureCredentialBuilder` to automatically resolve credentials, so authentication can be configured via any provider method supported by the `DefaultAzureCredentialBuilder`.
  For more information, see *Azure Entra Documentation*.
- **API key**
  You can specify an API key for your Microsoft Foundry deployment using any of the following methods:
  - Store the `com.fortify.sca.ai.azure.apiKey` property in the database using the dbTool. For more information, see Using the dbTool.
  - Set the `FORTIFY_AI_AZURE_API_KEY` system environment variable.
  - Set the `com.fortify.sca.ai.azure.apiKey` property in the `fortify-sca.properties` file. You can also specify the property on the command line with the `-D` option.

Specify the Azure OpenAI endpoint

You must specify the Azure OpenAI endpoint for your Microsoft Foundry deployment using either the base URL or the resource:

- Specify the base URL using any of the following methods:
  - Store the `com.fortify.sca.ai.azure.baseUrl` property in the database using the dbTool. For more information, see Using the dbTool.
  - Set the `FORTIFY_AI_AZURE_BASE_URL` system environment variable.
  - Set the `com.fortify.sca.ai.azure.baseUrl` property in the `fortify-sca.properties` file. You can also specify the property on the command line with the `-D` option.

- Specify the resource using any of the following methods:
  - Store the `com.fortify.sca.ai.azure.resource` property in the database using the dbTool. For more information, see Using the dbTool.
  - Set the `FORTIFY_AI_AZURE_RESOURCE` system environment variable.
  - Set the `com.fortify.sca.ai.azure.resource` property in the `fortify-sca.properties` file. You can also specify the property on the command line with the `-D` option.
-

6.1.3. Using Google Cloud for AI-powered SAST

The following properties can be set in the database using the dbTool, in the `fortify-sca.properties` file, or on the command-line using the `-D` option.

- Set the `com.fortify.sca.ai.google.project` property to the ID of your Google Cloud project.
- Set the `com.fortify.sca.ai.google.location` property to the Google Cloud region you selected for the model you are using.

AI-powered SAST uses the Application Default Credentials (ADC) chain to automatically discover Google Cloud credentials, so you can provide credentials in any way that ADC supports. Alternatively, you can specify the location of your Google credentials file using the `com.fortify.sca.ai.google.credentialsFile` property.

6.1.4. Using OpenAI APIs for AI-powered SAST

Fortify SAST makes HTTP requests to OpenAI.

Authentication

You must specify an API key using any of the following methods:

- Store the `com.fortify.sca.ai.openai.apiKey` property in the database using the dbTool. For more information, see Using the dbTool.
- Set the `OPENAI_API_KEY` system environment variable.
- Set the `com.fortify.sca.ai.openai.apiKey` property in the `fortify-sca.properties` file. You can also specify the property on the command line with the `-D` option.

(Optional) Specify the base URL

You can specify the base URL using any of the following methods:

- Store the `com.fortify.sca.ai.openai.baseUrl` property in the database using the dbTool. For more information, see Using the dbTool.
- Set the `OPENAI_BASE_URL` system environment variable.
- Set the `com.fortify.sca.ai.openai.baseUrl` property in the `fortify-sca.properties` file. You can also specify the property on the command line with the `-D` option.

(Optional) Specify the organization ID

You can specify the organization ID using any of the following methods:

- Store the `com.fortify.sca.ai.openai.organization` property in the database using the dbTool. For more information, see Using the dbTool.
- Set the `OPENAI_ORG_ID` system environment variable.
- Set the `com.fortify.sca.ai.openai.organization` property in the `fortify-sca.properties` file. You can also specify the property on the command line with the `-D` option.

(Optional) Specify the project ID

You can specify the project ID using any of the following methods:

- Store the `com.fortify.sca.ai.openai.project` property in the database using the dbTool. For more information, see Using the dbTool.
- Set the `OPENAI_PROJECT_ID` system environment variable.

- Set the `com.fortify.sca.ai.openai.project` property in the `fortify-sca.properties` file. You can also specify the property on the command line with the `-D` option.

6.2. Connecting to the database

We recommend configuring Fortify SAST to connect to a PostgreSQL database to enable the following features:

- Caching LLM results to reduce cost and scan time.
- Storing centralized configuration properties used to connect to the LLM and for configuring the behavior of AI-powered SAST.
- Enabling distributed rate limiting across multiple instances of `sourceanalyzer`.
- Storing LLM token usage data.

To connect to the database, provide the following JDBC connection properties in the fortify-sca.properties file or as command-line arguments.

Property	Description
com.fortify.sca.ai.db.url	Specifies the JDBC connection URL of the database. **Example** `jdbc:postgresql:my.domain.com:5432/ai_sast_db?currentSchema=ai_sast_schema` You may specify the PostgreSQL schema using the `currentSchema` query parameter in the URL. If no schema is specified the default schema `public` will be used. Alternatively, you can set the `FORTIFY_AI_DB_URL` system environment variable.
com.fortify.sca.ai.db.prop.username	Specifies the username to use to connect to the database. Alternatively, you can set the `FORTIFY_AI_DB_PROP_USERNAME` system environment variable.
com.fortify.sca.ai.db.prop.password	Specifies the password to use to connect to the database. This is a sensitive property. Users can choose to obfuscate the value using the `pwtool` to avoid storing it in plain text in the properties file or as a command line argument. Alternatively, you can set the `FORTIFY_AI_DB_PROP_PASSWORD` system environment variable.

com.fortify.sca.ai.db.prop.*	Any PostgreSQL JDBC connection properties can be set by prefixing them with "`com.fortify.sca.ai.db.prop.`".
	
	For example, set the following properties to enable authentication using SSL:
	
	`com.fortify.ai.db.prop.ssl=true`
	`com.fortify.ai.db.prop.sslmode=verify-full`
	`com.fortify.ai.db.prop.sslcert=my/dir/postgresql.crt`
	`com.fortify.ai.db.prop.sslkey=my/dir/postgresql.pk8`
	`com.fortify.ai.db.prop.sslrootcert=my/dir/root.crt`
	
	For more information about the available Postgres JDBC connection properties, see the *PostgreSQL Documentation*.
	
	Alternatively, you can set the `FORTIFY_AI_DB_PROP_SSL_CERT`, `FORTIFY_AI_DB_PROP_SSL_KEY`, and `FORTIFY_AI_DB_PROP_SSL_ROOT_CERT` system environment variables to provide SSL keys and certificates.

For more information, see Database configuration properties

6.3. Using the dbTool

The dbTool is used to provision and update configuration values in the database.

The dbTool is installed with Fortify SAST and located in the `<sast_install_directory>/bin` directory.

When you invoke the dbTool, you must provide the necessary database connection properties.

Example

`dbTool --com.fortify.sca.ai.db.url=<ai-db-url> --com.fortify.sca.ai.db.prop.username=<ai-db-username> --com.fortify.sca.ai.db.prop.password=<ai-db-password>`

Use the dbTool to store property values in the database using the `--store=<property>=<value>` argument.

Example

`--store="com.fortify.sca.ai.provider=aws" --store="com.fortify.sca.ai.model=us.anthropic.claude-sonnet-4-5-20250929-v1:0"`

Use the dbTool to store a collection of properties from a properties file using the `--storeFile=<path to .properties file>` argument.

Example

`--storeFile=/path/to/my.properties`

The properties file should have one property on each line using a `key=value` format.

Example

```
com.fortify.sca.ai.provider=aws
com.fortify.sca.ai.model=us.anthropic.claude-sonnet-4-5-20250929-v1:0
```

You may provide a base64 encoded AES 256 encryption key to encrypt values of sensitive properties before they are stored in the database. The encryption key must be provided to `sourceanalyzer` when scanning files in order to decrypt the encrypted property values.
You can use the following property values to specify an encryption key or an encryption key file:

- `com.fortify.sca.ai.config.encryptionKey`
- `com.fortify.sca.ai.config.encryptionKeyFile`

Example

`--com.fortify.sca.ai.config.encryptionKey="ZWm2HqvK3Jjrqb12ne1Wu/pU8qIa/CHQcSBxKa/7qrU="`

Note

If you do not specify an encryption key or an encryption key file, the dbTool will obfuscate sensitive property values to ensure the sensitive values are not stored in plain text in the database.

Here is an example usage of the dbTool:

Example

```
dbTool --com.fortify.sca.ai.db.url=jdbc:postgresql:my.domain.com:5432/sast-ai-db --
com.fortify.sca.ai.db.prop.username=ai-db-username --com.fortify.sca.ai.db.prop.password=ai-
db-password --store="com.fortify.sca.ai.provider=aws" --
store="com.fortify.sca.ai.model=us.anthropic.claude-sonnet-4-5-20250929-v1:0" --
storeFile="/path/to/file.properties" --
com.fortify.sca.ai.config.encryptionKey="ZWm2HqvK3Jjrqb12ne1Wu/pU8qIa/CHQcSBxKa/7qrU="
```

6.4. Sample analysis using AI-powered SAST

See the relevant language section to understand how to enable source files for AI-powered SAST.

During the scan phase, if you have configured your LLM properties inside a database, the basic command-line syntax is as follows:

Example

```
sourceanalyzer -b <build_id> -scan <database_connection_properties> <other_scan_options>
```

If you configure the database connection properties in the `<SAST_INSTALL_DIR>/Core/config/fortify-sca.properties` file, you do not need to specify the database connection properties during the scan.

Alternatively, you can specify all the LLM properties during the scan and fetch the credentials using the default credential provider chain or set the relevant credentials in the `fortify-sca.properties` file or on the command line.

Providing configuration for connecting to a database is not strictly required to use AI-powered SAST. Use the following basic command-line syntax when you analyze using AI-powered SAST without a database:

Example

```
sourceanalyzer -b <build_id> -scan <AI_configuration_properties> <other_scan_options>
```

Caution

OpenText does not recommended using AI-powered SAST without a database as it prevents result caching which is likely to incur additional costs.

The following example shows the basic command-line syntax to analyze a sample Erlang source file using AI-powered SAST:

Example

```
sourceanalyzer -b myProject -clean
sourceanalyzer -b myProject <file_path>/sample.erl
sourceanalyzer -b myProject -scan -Dcom.fortify.sca.ai.provider=aws -
Dcom.fortify.sca.ai.model="us.anthropic.claude-sonnet-4-5-20250929-v1:0" -f my_AI_results.fpr
```

6.5. AI-powered SAST configuration options

The following sections describe the properties available for configuring AI-powered SAST. Store a property in the database using the dbTool, set a property in the `fortify-sca.properties` file, or specify the property on the command line with the `-D` option.

LLM request configuration

Property	Description
com.fortify.sca.ai.enabled	If set to false, disables AI-powered SAST. **Important** Set this property in the `fortify-sca.properties` or pass it as a command-line argument. If you configure the property in the database using dbTool, it has no effect. **Default**: true
com.fortify.sca.ai.provider	Required. Specifies the LLM vendor. **Default**: null **Supported values:** • azure • openai • aws

com.fortify.sca.ai.model	Required. Specifies the LLM. **Default**: null **For Azure:** Accepts the model name: • claude-sonnet-4-6 • claude-opus-4-6 • claude-sonnet-4-5 • gpt-5.4 • gpt-5.2 (verified) • gpt-5.2-Codex • gpt-5.1 • gpt-5.1-Codex • gpt-5 • gpt-5-Codex **For Google Cloud:** Accepts the model name. Any Anthropic Claude model is accepted. For available models, see the Google Cloud Documentation. Example: claude-sonnet-4-6 **For OpenAI:** Accepts the model name: • gpt-5.4 • gpt-5.2 (verified) • gpt-5.2-Codex • gpt-5.1 • gpt-5.1-Codex • gpt-5 • gpt-5-Codex **For AWS**: Accepts an inference profile ID or inference profile ARN for a system inference profile or application inference Profile. You may alternatively set the `com.fortify.sca.ai.aws.inferenceProfile` property instead of the `com.fortify.sca.ai.model` property. If you are using a custom application inference profile and have not provided the appropriate permissions to resolve the underlying model of an inference profile (requires permissions to call the Bedrock `GetInferenceProfile` API), then you may set the `com.fortify.sca.ai.model` property to the underyling model ID and the `com.fortify.sca.ai.aws.inferenceProfile` property to the application inference profile ARN. The given inference profile should use one of the following foundation models: • Claude Sonnet 4.6 (anthropic.claude-sonnet-4-6)

- Claude Opus 4.6 (anthropic.claude-opus-4-6-v1)
- Claude Sonnet 4.5 (anthropic.claude-sonnet-4-20250514-v1:0)
- Claude Sonnet 4 (anthropic.claude-sonnet-4-5-20250929-v1:0)

Supported AWS system inference profiles:

- Claude Sonnet 4.6
  - global.anthropic.claude-sonnet-4-6
  - us.anthropic.claude-sonnet-4-6
  - au.anthropic.claude-sonnet-4-6
  - eu.anthropic.claude-sonnet-4-6
  - jp.anthropic.claude-sonnet-4-6
- Claude Opus 4.6
  - global.anthropic.claude-opus-4-6-v1
  - us.anthropic.claude-opus-4-6-v1
  - au.anthropic.claude-opus-4-6-v1
  - eu.anthropic.claude-opus-4-6-v1
- Claude Sonnet 4.5 (verified)
  - global.anthropic.claude-sonnet-4-5-20250929-v1:0
  - us.anthropic.claude-sonnet-4-5-20250929-v1:0
  - au.anthropic.claude-sonnet-4-5-20250929-v1:0
  - eu.anthropic.claude-sonnet-4-5-20250929-v1:0
  - jp.anthropic.claude-sonnet-4-5-20250929-v1:0
- Claude Sonnet 4
  - global.anthropic.claude-sonnet-4-20250514-v1:0
  - us.anthropic.claude-sonnet-4-20250514-v1:0
  - apac.anthropic.claude-sonnet-4-20250514-v1:0
  - eu.anthropic.claude-sonnet-4-20250514-v1:0

Note

System inference profiles must be used in an AWS region that is supported by the system inference profile.

Example

`us.anthropic.claude-sonnet-4-5-20250929-v1:0` may be used in the `us-east-1` region but not in the `eu-west-1` region.

com.fortify.sca.ai.llm.parallelism	The maximum number of concurrent LLM requests. **Default**: 4 x available processors
com.fortify.sca.ai.llm.rateLimiter.enabled	Specifies whether the LLM request rate limiter is enabled. **Default**: true
com.fortify.sca.ai.llm.rateLimiter.updateDistributedLimits	Specifies whether the provided `requestsPerMinute` and `tokensPerMinute` should update existing limits in the database. If set to true, the limits in the database will be updated. If set to false, the existing limits will not be updated and the provided limits will be ignored. **Default**: false OpenText recommends to use the `dbTool` to set the distributed rate limits in the database.
com.fortify.sca.ai.llm.requestsPerMinute	Specifies the maximum number of LLM requests allowed per minute. This property sets the total limit across all instances of `sourceanalyzer` connected to the same database. If there are no database configured, this property sets the maximum limit for a single `sourceanalyzer` instance. **Default**: 200
com.fortify.sca.ai.llm.tokensPerMinute	Specifies the maximum number of LLM request input and output tokens allowed per minute. This property sets the total limit across all instances of `sourceanalyzer` connected to the same database. If there are no database configured, this property sets the maximum limit for a single `sourceanalyzer` instance. **Default**: 200,000
com.fortify.sca.ai.llm.maxRequestCount	The maximum number of LLM requests that may be made in a single scan. This property ensures users to avoid incurring a large bill due to accidentally scanning a large project. Before making any LLM requests, the number of necessary requests is calculated. If the number of required requests exceeds the limit, the scan is aborted without making any LLM requests. **Default**: 10,000

com.fortify.sca.ai.fileChunkSize	The maximum number of characters allowed in a request to a LLM. If a file exceeds this amount, the file will be split into multiple file chunks. If the chunk size is changed, it may cause some cache entries to become stale if the associated file content changes due to the chunk size changing. New LLM requests will be required for the affected file chunks. **Default**: 50,000

Vendor configuration

Property	Description
AWS configuration properties	
com.fortify.sca.ai.aws.inferenceProfile	Specifies the Bedrock inference profile using the inference profile's ID or ARN. This can be set instead of the `com.fortify.sca.ai.model` property. If you are using a custom application inference profile and have not provided the appropriate permissions to resolve the underlying model of an inference profile (requires permissions to call the Bedrock `GetInferenceProfile` API), then you may set the `com.fortify.sca.ai.model` property to the underyling model ID and the `com.fortify.sca.ai.aws.inferenceProfile` property to the application inference profile ARN.
com.fortify.sca.ai.aws.region	Specifies the AWS region in which to make the LLM request. **Default**: null Alternatively may be provided by the following: • Environment variable: `AWS_REGION` • AWS config or credentials file: `region`
com.fortify.sca.ai.aws.profile	Specifies which profile in the AWS `config` or `credentials` file to use to authenticate with AWS. **Default**: null **Environment variable**: `AWS_PROFILE`
com.fortify.sca.ai.aws.accessKeyId	Specifies the AWS access key to use when authenticating with AWS. If set, must also set `com.fortify.sca.ai.aws.secretAccessKey` and `com.fortify.sca.ai.aws.sessionToken` (if you are using short-term credentials). **Default**: null Alternatively may be provided by the following: • Environment variable: `AWS_ACCESS_KEY_ID` • AWS config or credentials file: `aws_access_key_id`

com.fortify.sca.ai.aws.secretAccessKey	Specifies the AWS secret access key to use when authenticating with AWS. If set, must also set `com.fortify.sca.ai.aws.secretAccessKey` and `com.fortify.sca.ai.aws.sessionToken` (if you are using short-term credentials). **Default**: null Alternatively may be provided by the following: • Environment variable: `AWS_SECRET_ACCESS_KEY` • AWS config or credentials file: `aws_secret_access_key` This is a sensitive property. The value will be masked in logs. Users can choose to obfuscate the value using `pwtool` to avoid storing it in plain text in the properties file or as a command line argument.
com.fortify.sca.ai.aws.sessionToken	Specifies the AWS session token to use when authenticating with AWS. If set, must also set `com.fortify.sca.ai.aws.accessKeyId` and `com.fortify.sca.ai.aws.secretAccessKey.` Alternatively may be provided by the following: • Environment variable: `AWS_SESSION_TOKEN` • AWS config or credentials file: `aws_session_token` This is a sensitive property. The value will be masked in logs. Users can choose to obfuscate the value using `pwtool` to avoid storing it in plain text in the properties file or as a command line argument.
com.fortify.sca.ai.aws.bearerTokenBedrock	Specifies the Bedrock bearer token that can be used to authenticate with AWS. Alternatively may be provided by setting the `AWS_BEARER_TOKEN_BEDROCK` system environment variable. This is a sensitive property. The value will be masked in logs. Users can choose to obfuscate the value using `pwtool` to avoid storing it in plain text in the properties file or as a command line argument.
Azure configuration properties	
com.fortify.sca.ai.azure.deployment	Specifies the name of the Microsoft Foundry deployment. Only required if the deployment does not have the same name as the model provided for `com.fortify.sca.ai.model`.

com.fortify.sca.ai.azure.resource	Specifies the Azure Foundry resource name. Specify either the `com.fortify.sca.ai.azure.resource` or `com.fortify.sca.ai.azure.baseUrl` but not both. Alternatively, may be provided by setting the `FORTIFY_AI_AZURE_RESOURCE` system environment variable.
com.fortify.sca.ai.azure.baseUrl	Specifies the Azure OpenAI endpoint for your Microsoft Foundry deployment. Specify either the `com.fortify.sca.ai.azure.baseUrl` or `com.fortify.sca.ai.azure.resource` but not both. Alternatively, may be provided by setting the `FORTIFY_AI_AZURE_BASE_URL` system environment variable.
com.fortify.sca.ai.azure.apiKey	Specifies the secret API key for authenticating with your Microsoft Foundry deployment. **Environment variable**: `FORTIFY_AI_AZURE_API_KEY`
Google Cloud configuration properties	
com.fortify.sca.ai.google.project	Specifies the Google Cloud project ID.
com.fortify.sca.ai.google.location	Specifies the Google Cloud region or location where the model is deployed. **Example**: us-east5 or global.
com.fortify.sca.ai.google.credentialsFile	Specifies the path to a Google Cloud credentials JSON file. Optional, only set if your credentials are not discoverable through the standard Application Default Credentials (ADC) chain. In most environments, leave this unset and let ADC resolve credentials automatically.
OpenAI configuration properties	
com.fortify.sca.ai.openai.apiKey	Specifies the secret API key for authenticating with OpenAI. **Environment variable:** `OPENAI_API_KEY`
com.fortify.sca.ai.openai.organization	Specifies the identifier of the OpenAI organization. Only required if you belong to multiple organizations. **Environment variable**: `OPENAI_ORG_ID`

com.fortify.sca.ai.openai.project	Specifies the identifier of the OpenAI project. Only required if you access projects through a legacy user API key. **Environment variable**: `OPENAI_PROJECT_ID`
com.fortify.sca.ai.openai.baseUrl	Specifies the base URL used to connect to the OpenAI service. Only required if you are not connecting directly to OpenAI servers. Defaults to the production environment: https://api.openai.com/v1. **Environment variable**: `OPENAI_BASE_URL`

Property	Description
Database connection properties	
com.fortify.sca.ai.db.url	Specifies the JDBC connection URL of the database. **Example** `jdbc:postgresql:my.domain.com:5432/ai_sast_db?currentSchema=ai_sast_schema` Required to connect to database for LLM result caching, centralized configuration, distributed rate limiting, and LLM request usage data. You may specify the PostgreSQL schema using the `currentSchema` query parameter in the URL. If no schema is specified the default schema `public` will be used. For more information, see the *PostgreSQL Documentation*. **Default**: null
com.fortify.sca.ai.db.prop.username	Specifies the username to use to connect to the database. Required to connect to database for LLM result caching, centralized configuration, distributed rate limiting, and LLM request usage data. **Default**: null
com.fortify.sca.ai.db.prop.password	Specifies the password to use to connect to the database. This is a sensitive property. The value will be masked in logs. Users can choose to obfuscate the value using the `pwtool` to avoid storing it in plain text in the properties file or as a command line argument. **Default**: null

com.fortify.sca.ai.db.prop.*	Any PostgreSQL JDBC connection properties can be set by prefixing them with "`com.fortify.sca.ai.db.prop.`".
	Default: null
	For example, set the following properties to enable authentication using SSL:
	`com.fortify.ai.db.prop.ssl=true`
	`com.fortify.ai.db.prop.sslmode=verify-full`
	`com.fortify.ai.db.prop.sslcert=my/dir/postgresql.crt`
	`com.fortify.ai.db.prop.sslkey=my/dir/postgresql.pk8`
	`com.fortify.ai.db.prop.sslrootcert=my/dir/root.crt`
	For more information about the available Postgres JDBC connection properties, see *PostgreSQL Documentation*

Database behavior properties	
com.fortify.sca.ai.cache.enabled	Specifies whether the LLM result cache should be enabled. If disabled, both cache reads and writes will be disabled. **Default**: true The more specific properties `com.fortify.sca.ai.cache.read.enabled` and `com.fortify.sca.ai.cache.write.enabled` take precedence over this property if set.
com.fortify.sca.ai.cache.read.enabled	Specifies whether reading from the LLM result cache should be enabled. **Default**: true
com.fortify.sca.ai.cache.write.enabled	Specifies whether writing to the LLM result cache should be enabled. **Default**: true
com.fortify.sca.ai.cache.missOnNewModel	Specifies whether to consider an LLM result in the cache to be a cache miss if the LLM model used for the cached result is different from the model for the current request. Effectively, setting this to true will make it so that cached results are evicted if a new model is used. Default: false
com.fortify.sca.ai.cache.missOnNewPrompt	Specifies whether to consider an LLM result in the cache to be a cache miss if the LLM prompt used in the cached result is different from the prompt for the current request. Effectively, setting this to true will make it so that cached results are evicted if the relevant rule pack is updated. **Default**: true

com.fortify.sca.ai.cache.maxSize	The maximum number of LLM results cached before LRU eviction. **Default**: 1,000,000
com.fortify.sca.ai.cache.ttl.day	The number of days after which LLM results that have not been used will be evicted from the cache. **Default**: 30
com.fortify.sca.ai.remoteConfig.enabled	Specifies whether configuration properties may be obtained from the connected database. **Default**: true
com.fortify.sca.ai.config.encryptionKey	The optional base64 encoded AES-256 symmetric encryption key that is used to encrypt and decrypt values of sensitive properties that are stored in the database. The encryption key must be distributed to all `sourceanalyzer` instances in order for them to retrieve sensitive configuration from the database. You can distribute encryption keys using KMS, OS key store, password manager, and so on. **Caution** If the encryption key is lost, the configuration values that were encrypted using the key will be unrecoverable. If no encryption key is provided through either `com.fortify.sca.ai.config.encryptionKey` or `com.fortify.sca.ai.config.encryptionKeyFile`, the values of sensitive properties that are stored in the database will be obfuscated so they are not stored in plain text. This is a sensitive property. The value will be masked in logs. Users can choose to obfuscate the value using the `pwtool` to avoid storing it in plain text in the properties file or as a command line argument. **Default**: null

com.fortify.sca.ai.config.encryptionKeyFile	The optional path to the file that contains the base64 encoded AES-256 symmetric encryption key that is used to encrypt and decrypt values of sensitive properties that are stored in the database. The encryption key must be distributed to all `sourceanalyzer` instances in order for them to retrieve sensitive configuration from the database. You can distribute encryption keys using KMS, OS key store, password manager, and so on. **Caution** WARNING: If the encryption key is lost, the configuration values that were encrypted using the key will be unrecoverable. If no encryption key is provided through either `com.fortify.sca.ai.config.encryptionKey` or `com.fortify.sca.ai.config.encryptionKeyFile`, the values of sensitive properties that are stored in the database will be obfuscated so they at least are not stored in plain text. **Default**: null
com.fortify.sca.ai.usage.enabled	Specifies whether writing to the usage table is enabled. **Default**: true

File configuration

Property	Description
com.fortify.sca.ai.exclude.directories	Excludes any files nested within a directory whose name matches one of the given semi-colon-separated regex patterns. Default value: `test;tests;__tests__;testing;testcases;spec;example;examples;sample;samples;demo;demos;doc;docs;documentation`
com.fortify.sca.ai.exclude.fileNames	Excludes any files whose name matches one of the given semi-colon-separated regex patterns. Default value: `.*Test\..*;.*_test\..*;.*\.test\..*;.*\.spec\..*;.*Tests\..*;.*_spec\..*`

pwtool configuration

Property	Description
com.fortify.sca.ai.pwtool.encryptionKeyFile	The path to the encryption key file generated by pwtool. Required if the user provides values for sensitive properties that were obfuscated using pwtool. **Default**: null

6.6. Rate limiting

LLM providers enforce request rate limits and token rate limits to ensure service reliability and fair resource allocation. Exceeding these limits can result in throttling. When a request is throttled, Fortify SAST temporarily reduces the request rate and retries each throttled request up to a maximum number of attempts. Occasional throttling should have minimal impact on scan results, but heavy throttling can cause increased scan times and missing results due to some files not being analyzed.

You can resolve throttling issues in one of the following ways:

- You can submit a request to the LLM provider to increase your request rate limits and token rate limits for the model you use.
- You can set client-side rate limits to adjust the rate at which Fortify SAST makes requests to the LLM provider. If you are using a connected database, use the `dbTool` to set the `com.fortify.sca.ai.llm.requestsPerMinute` and `com.fortify.sca.ai.llm.tokensPerMinute` properties to enable distributed rate limiting across all instances of Fortify SAST that are connected to the database.

  Alternatively, set `com.fortify.sca.ai.llm.rateLimiter.updateDistributedLimits` to true (if you are using a connected database) and set the `com.fortify.sca.ai.llm.requestsPerMinute` and `com.fortify.sca.ai.llm.tokensPerMinute` properties in the `fortify-sca.properties` file or on the command line with the `-D` option.

6.7. Using the pwtool to encrypt sensitive values

You can use the pwtool to generate encrypted values for sensitive property values and use the encrypted values in place of the plain text values. This allows you to avoid storing sensitive values in plain text in the `fortify-sca.properties` file or to avoid passing sensitive values in plain text as command-line arguments.

You can encrypt the following properties using the pwtool:

- `com.fortify.sca.ai.azure.apiKey`
- `com.fortify.sca.ai.openai.apiKey`
- `com.fortify.sca.ai.db.prop.password`
- `com.fortify.sca.ai.aws.secretAccessKey`
- `com.fortify.sca.ai.aws.sessionToken`
- `com.fortify.sca.ai.config.encryptionKey`

To encrypt a sensitive property value:

1. At the command prompt, run the following command:

   ```
   <tools_install_dir>/bin/pwtool <pwtool_keys_file>
   ```

2. When prompted, type the sensitive value and press **Enter**.
3. The pwtool generates a new key stored in the file on the path specified in step 1 or reuses an existing file on the specified path. The pwtool uses the key to encrypt the typed sensitive value.
4. Copy the encrypted secret, and paste it as the value for the property in the `fortify-sca.properties` file or as a command-line argument.
5. To encrypt more sensitive property values, repeat steps 1 through 4 for each property value you want to encrypt.

6. Set the following property in the `fortify-sca.properties` file or pass as a command-line argument: `com.fortify.sca.ai.pwtool.encryptionKeyFile=<pwtool_keys_file>`

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.

Fortify 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.

- Integrating with Gradle
- Integrating with Maven
- Integrating with Bazel
- Integrating with Ant
- Manual Java and Kotlin translation syntax
- Analyzing Kotlin scripts
- Kotlin and Java translation interoperability
- Handling Java warnings
- Analyzing Jakarta EE (Java EE) applications
- Analyzing Java bytecode
- Troubleshooting Java, Kotlin, and JSP translation and analysis issues

7.1. Integrating with Gradle

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

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

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, Fortify SAST translates the source files as they are compiled. Alternatively, you can use the Fortify SAST Gradle Plugin to perform the analysis from within your Gradle build script (see Using the Fortify 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 Fortify 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 Fortify SAST installations, make sure that the version you want to use for your Gradle projects is defined before all other Fortify 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 Fortify 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
```

7.1.2. Troubleshooting Gradle integration

If you use configuration caching (`--configuration-cache` option) in your Gradle build with Fortify 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 Fortify 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
```

7.1.3. Using the Gradle plugin

The Fortify SAST installation includes a Gradle plugin located in `<sast_install_dir>/plugins` gradle. To use the Fortify 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 Fortify SAST tasks for the analysis: sca.clean, sca.translate, and sca.scan. See Build tools for platforms and languages supported specifically for Fortify SAST Gradle plugin.

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

To configure the Fortify 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 '26.4'
   ```

   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 "26.4"
...
}
```

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 Fortify 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 Fortify SAST has already translated the project using the Fortify 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 Fortify 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")
...
}
}
```

7.2. Integrating with Maven

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

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

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

- Installing and updating the Fortify Maven Plugin
- Testing the Fortify Maven Plugin installation
- Using the Fortify Maven Plugin

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.

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.
   Fortify 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.

7.2.3. Using the Fortify Maven Plugin

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

- In an Fortify SAST build integration
  In this method, prepend the maven command used to build your project with the `sourceanalyzer` command and any Fortify SAST options. To analyze your files as part of an Fortify 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 Fortify 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:26.4: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:26.4: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`

7.3. Integrating with Bazel

To integrate with Bazel builds, Fortify 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
```

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
```

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

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, Fortify 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
```

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, Fortify SAST loads classes in the order they appear in the class path. If there are multiple classes with the same name in the list, Fortify SAST uses the first loaded class. In the following example, if both `A.jar` and `B.jar` include a class called `MyData.class`, Fortify 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.
  Fortify 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, Fortify SAST can additionally emulate the compiler to help integrate more easily into custom build scripts.

To have Fortify SAST emulate the compiler, type:

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

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 similar to javac and consists of a semicolon-separated (Windows) or colon-separated (Linux/macOS) list of directories and files.
	The `-classpath` or `-cp` behave similarly to the way `-classpath` works with javac and java:
	• `-cp "<dir>"` searches the specified directory for `.class` files
	• `-cp "<dir>/*"` | `-cp "<dir>/*.jar"` searches the specified directory for `.jar` files
	Caution
	To avoid unintended shell expansion, enclose classpath values in quotation marks.
	You can use Fortify SAST file specifiers as shown in the following example:
	`-cp "build/classes:lib/*.jar"`
	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 Fortify 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

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
```

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"
```

7.6. Analyzing Kotlin scripts

Fortify 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. Fortify SAST translates `*.kts` files but does not apply these templates.

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 Fortify 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.

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 Fortify 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.

7.9. Analyzing Jakarta EE (Java EE) applications

To translate Jakarta EE applications, Fortify 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.

- Translating Java files
- Translating JSP projects, configuration files, and deployment descriptors
- Jakarta EE (Java EE) translation warnings

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".

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.

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.

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 Fortify SAST processes `.class` and `.jar` files.
2. Do one of the following:
   - Request that Fortify 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 Fortify 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"
  ```

7.11. Troubleshooting Java, Kotlin, and JSP translation and analysis issues

The following sections provide troubleshooting information for Java, Kotlin, and JSP analysis.

Many Java source files were not translated

When translating Java projects if some files are missing from the scan results, you can disable multi-file Java parsing by adding the following property to the `fortify-sca.properties` file:

`com.fortify.sca.JavaParseMultipleFiles=false`

Unable to translate some JSPs

Fortify 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 Fortify 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 fixing those
<list_of_jsp_files>
```

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

- The web application is not laid out in a proper deployable WAR directory format
- 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 Fortify 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 Fortify 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 Fortify 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`.

8. Analyzing Android projects

This section describes how to translate Java source code for Android applications. You can use Fortify 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*.

- Android project translation prerequisites
- Android code analysis command-line syntax
- Filtering issues detected in Android layout files

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

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.

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, Fortify 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.

9. Analyzing Groovy code

This section describes how to analyze Groovy projects. For projects combined with other files, see the relevant section for those languages.

This section contains the following topics:

- Groovy analysis prerequisites
- Groovy translation syntax

9.1. Groovy analysis prerequisites

Currently, Groovy code is only compatible with AI-powered SAST. For more information about configuring scans for AI-powered SAST, see Analyzing using AI-powered SAST.

If AI-powered SAST is not configured, Groovy code will only be scanned via Regular Expression Analysis and users may see suboptimal results.

9.2. Groovy translation syntax

To include Groovy code for analysis using AI-powered SAST, include all source files that you want to analyze.

Use the following basic command-line syntax to analyze Groovy code:

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

See Specifying files and directories for more information.

Example

```
sourceanalyzer -b <build_id> "**/*.groovy"
```

Important

Supported file extensions for the Groovy source files are: `.groovy`, `.gvy`, `.gy`, `.gsh`.

10. 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 Fortify SAST installation in the `<sast_install_dir>/plugins/lightbend` directory.

Caution

After upgrading to a new version, your existing Akka license file may expire or no longer be valid.

If you encounter such issues after an upgrade:

1. Contact Customer Support to request a license for the new version.
2. When you receive the updated license, remove the old license file and install the new one provided by Support.

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.

11. Analyzing Visual Studio projects

Fortify 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.

- Visual Studio project translation prerequisites
- Visual Studio Project command-line syntax
- Handling special cases for translating Visual Studio projects
- Alternative ways to translate Visual Studio projects

11.1. 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.

11.2. 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 Fortify SAST translation command. This starts a build integration that analyzes your solution and project files and automatically executes the appropriate translation steps. Fortify SAST translates all the projects contained in the Visual Studio solution (`.sln` or `.slnx`). You can also translate one or more specific projects by providing a semicolon-separated list of projects.

> **Important**
>
> To ensure that the build integration correctly pulls in all of the appropriate project dependencies and resources, you must run the Fortify 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, Fortify SAST translates all the projects contained in the Visual Studio solution `Sample.sln`.

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 Fortify 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 Fortify 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
```

11.3. Handling special cases for translating Visual Studio projects

- Running translation from a script
- Translating plain .NET and ASP.NET projects
- Translating C/C++ and Xamarin projects
- Translating projects with settings containing spaces
- Translating a single project from a Visual Studio solution
- Analyzing projects that build multiple executable files

11.3.1. 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 Fortify SAST translation:

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

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

11.3.2. 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.

11.3.3. 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.

11.3.4. 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
```

11.3.5. 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
```

11.3.6. 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.

11.4. Alternative ways to translate Visual Studio projects

This section describes alternative methods of translating Visual Studio projects.

- Alternative translation options for Visual Studio solutions
- Translating without explicitly running Fortify SAST

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

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

  Note that Fortify 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.

11.4.2. Translating without explicitly running Fortify SAST

You have the option to translate your Visual Studio project without invoking Fortify 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 Fortify 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 Fortify SAST build ID for translation. Make sure that you set this value. This is equivalent to the Fortify SAST `-b` option.	None
FORTIFY_MSBUILD_DEBUG	Enables debug mode. This is equivalent to the Fortify SAST `–debug` option.	False
FORTIFY_MSBUILD_DEBUG_VERBOSE	Enables verbose debug mode. This is equivalent to the Fortify 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 Fortify SAST log file. This is equivalent to the Fortify SAST `-logfile` option.	`%LOCALAPPDATA%/Fortify/sca/log/sca.log`

12. 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. Fortify SAST does not support adding new files to the file list associated with the build ID on subsequent invocations.

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

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

- Translating pure JavaScript projects
- Excluding dependencies
- Excluding NPM Dependencies
- NPM dependencies
- Translating JavaScript projects with HTML files

12.1. 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>`.

Analyzing Vue 3 projects (preview)

To translate and scan a Vue 3 project, include the `-flight-js` and `-vue3` options in the command-line instructions for both translation and scanning.

```
sourceanalyzer -b <build_id> -flight-js -vue3  <vue_files_or_dir>
sourceanalyzer -b <build_id> -flight-js -vue3 -scan -f results.fpr
```

12.2. 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 Fortify SAST does not translate the file shown in the previous example.

You can use regular expressions for the file names. Note that Fortify 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).

12.3. Excluding NPM Dependencies

By default, Fortify 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 Fortify 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 Fortify 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.

12.4. NPM dependencies

By default, Fortify 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.

- Examples of excluding NPM dependencies

12.4.1. 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
```

12.5. 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, Fortify 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 Fortify SAST to include in the DOM modeling with the `com.fortify.sca.DOMModeling.tags` property. By default, Fortify 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
```

13. Analyzing Dart and Flutter code

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

- Dart and Flutter translation prerequisites
- Dart and Flutter command-line syntax
- Dart and Flutter command-line examples

13.1. 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

13.2. 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>
```

13.3. 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"
```

14. Analyzing Python and Jupyter Notebooks

Fortify 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. Fortify SAST supports translation of Jupyter notebooks and the Django and Flask frameworks.

- Integrating with Bazel
- Python translation command-line syntax
- Translating Python in a virtual environment
- Including imported modules and packages
- Including namespace packages
- Translating Django and Flask

14.1. Integrating with Bazel

To integrate with Bazel builds, Fortify 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
```

14.1.1. 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
```

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

14.2. 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. Fortify SAST does not support adding new files to the file list associated with the build ID on subsequent invocations.

14.2.1. 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. Fortify 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. Fortify 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 Fortify 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. Fortify 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`

Python option	Description
-disable-template-autodiscover	Specifies that Fortify SAST does not automatically discover Django or Jinja2 templates. **Equivalent property name:** com.fortify.sca.DisableTemplateAutodiscover

Python Properties

14.2.2. 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
```

14.3. 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/
```

14.4. Including imported modules and packages

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

Fortify 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 Fortify 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.
   Fortify SAST includes a subset of modules from the standard Python library (module "builtins", all modules originally written in C, and others) in the translation. Fortify SAST first searches for a standard Python library module in the set included with Fortify SAST and then in the paths specified with the `-python-path` option. If your Python code imports any module that Fortify 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 Fortify SAST translates a `PrimaryDir/project1/a.py`, the directory `PrimaryDir/project1` is added as the last directory to search for imported modules and packages.

14.5. 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.

14.6. Translating Django and Flask

By default, Fortify 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 Fortify 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
```

15. Analyzing iOS and Xcode projects

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

- iOS project translation prerequisites
- iOS code analysis command-line syntax

15.1. 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. Fortify 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.

15.2. 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 Fortify 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
```

> **Note**
> The `-exclude` option does not exclude Swift source files generated or downloaded during Xcode build integration (for example, files under `**/DerivedData/**`), even when the files are explicitly specified in the exclude pattern.

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
```

16. Analyzing C and C++ code

This section describes how to translate C and C++ code. Fortify 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.

- C and C++ Code translation prerequisites
- Integrating with Make
- Integrating with CMake
- Integrating with Gradle
- Manual C and C++ translation syntax
- C/C++ Precompiled Header Files
- Using other compilers

16.1. 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. Fortify SAST translation does not require object files and static/dynamic library files.

16.2. Integrating with Make

To integrate Fortify 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, Fortify SAST automatically combines each of the separately-translated files into a single translated project.

16.3. Integrating with CMake

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

To integrate Fortify 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
   ```

16.4. 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

16.5. 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 Fortify 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 Fortify 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 Fortify 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 Fortify 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
```

Fortify 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 Fortify SAST `-nc` option to disable the compiler execution.

16.6. C/C++ Precompiled Header Files

Some C/C++ compilers support Precompiled Header Files, which can improve compilation performance. Some compilers' implementations of this feature have subtle side-effects. When the feature is enabled, the compiler might accept erroneous source code without warnings or errors. This can result in a discrepancy where Fortify 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.

16.7. Using other compilers

The unsupported C compiler adapter provides an alternative when you cannot use standard compilers such as GCC or MSVC.

Configuration

To register a custom compiler and linker, add the following entries to your `fortify-sca.properties` file:

```
com.fortify.sca.compilers.mycompiler = com.fortify.sca.util.compilers.ConfigurableCompiler
com.fortify.sca.compilers.mylinker = com.fortify.sca.util.compilers.LdCompiler
```

Replace `mycompiler` and `mylinker` with the actual names of your compiler and linker executables.

Running an analysis

After you configure the adapter, run `sourceanalyzer` with your standard command.

The adapter probes the compiler for system include directories by using a GCC-style probe. If the probe fails, you see the following warning:

```
No system includes found for compiler mycompiler.
Translation will not be able to find system include directories.
```

This means the engine cannot locate system headers such as `iostream`, `vector`, or `stdio.h`.

To resolve this, specify the include directories manually by using the `com.fortify.sca.ctran.clang-args` property:

```
-Dcom.fortify.sca.ctran.clang-args="-isystem /my/system/dir -isystem /my/other/system/dir"
```

Handling non-standard syntax

Some non-standard C/C++ constructs can cause the engine to stop translation.

If your code uses custom preprocessor directives that are not recognized, wrap them in a conditional block using the `FORTIFYSCA` macro:

```
#ifdef FORTIFYSCA
#include "myinclude.h"
#else
#includetag* "myinclude.h"
#endif // FORTIFYSCA
```

Then pass the macro definition via the `com.fortify.sca.ctran.clang-args` property:

```
-Dcom.fortify.sca.ctran.clang-args="-DFORTIFYSCA"
```

Fortify SAST uses the standard `#include` directive, while your compiler continues to use its native syntax.

Clang arguments reference

The `com.fortify.sca.ctran.clang-args` property passes arguments directly to the Clang frontend used by the engine during C/C++ translation.

For a full list of supported arguments, see the Clang Command Line Reference.

17. Analyzing Rust code

This section describes how to analyze Rust projects. For projects combined with other files, see the relevant section for those languages.

This section contains the following topics:

- Rust analysis prerequisites
- Rust translation syntax

17.1. Rust analysis prerequisites

Currently, Rust code is only compatible with AI-powered SAST. For more information about configuring scans for AI-powered SAST, see Analyzing using AI-powered SAST.

If AI-powered SAST is not configured, Rust code will only be scanned via Regular Expression Analysis and users may see suboptimal results.

17.2. Rust translation syntax

To include Rust code for analysis using AI-powered SAST, include all source files that you want to analyze.

Use the following basic command-line syntax to analyze Rust code:

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

See Specifying files and directories for more information.

Example

```
sourceanalyzer -b <build_id> "**/*.rs"
```

18. Analyzing Fortran code

This section describes how to analyze Fortran projects. For projects combined with other files, see the relevant section for those languages.

This section contains the following topics:

- Fortran analysis prerequisites
- Fortran translation syntax

18.1. Fortran analysis prerequisites

Currently, Fortran code is only compatible with AI-powered SAST. For more information about configuring scans for AI-powered SAST, see Analyzing using AI-powered SAST.

If AI-powered SAST is not configured, Fortran code will only be scanned via Regular Expression Analysis and users may see suboptimal results.

18.2. Fortran translation syntax

To include Fortran code for analysis using AI-powered SAST, include all source files that you want to analyze.

Use the following basic command-line syntax to analyze Fortran code:

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

See Specifying files and directories for more information.

Example

```
sourceanalyzer -b <build_id> "**/*.f"
```

Important
Supported file extensions for the Fortran source files are: `.f`, `.for`, `.ftn`, `.f90`, `.f95`, `.f03`, `.f08`, `.f15`, and `.f18`.

19. Analyzing AI-agent instructions files

This section describes how to analyze AI-agent instructions files.

This section contains the following topics:

- AI-agent instructions files analysis prerequisites
- AI-agent instructions files translation syntax

19.1. AI-agent instructions files analysis prerequisites

Currently, analyzing AI-agent instructions files is only compatible with AI-powered SAST. For more information about configuring scans for AI-powered SAST, see Analyzing using AI-powered SAST.

19.2. AI-agent instructions files translation syntax

To include AI-agent instructions files for analysis using AI-powered SAST, include all the AI-agent instructions files that you want to analyze.

Use the following basic command-line syntax to analyze AI-agent instructions files:

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

See Specifying files and directories for more information.

Example

```
sourceanalyzer -b <build_id> "**/*.md"
```

Important
Supported file extensions for the AI-agent instructions files are: `.md`, `.mdc`, `.cursorrules`, `.windsurfrules`.

20. Analyzing Go code

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

- Go command-line syntax
- Go command-line options
- Including custom Go build tags
- Resolving dependencies

20.1. 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 Fortify Software Security Center. 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.

20.2. 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 Fortify 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
	Fortify 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 Fortify SAST uses the Go compiler included in the Fortify SAST installation. **Equivalent property name:** com.fortify.sca.GOROOT

Go option	Description
`-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 Fortify SAST uses `https://proxy.golang.org,direct`. **Equivalent property name:** `com.fortify.sca.GOPROXY`

Go properties

20.3. 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 Fortify SAST translation using the `-gotags` option. For example:

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

The Fortify 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.

20.4. Resolving dependencies

Fortify 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.
  Fortify SAST downloads all required dependencies using the native Go toolchain. If access to the internet is restricted on the machine where you run Fortify 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. For more information, see Go command-line syntax.

21. 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. Fortify SAST does not support adding new files to the file list associated with the build ID on subsequent invocations.

- PHP command-line options

21.1. 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`

22. Analyzing Perl code

This section describes how to analyze Perl projects. For projects combined with other files, see the relevant section for those languages.

This section contains the following topics:

- Perl analysis prerequisites
- Perl translation syntax

22.1. Perl analysis prerequisites

Currently, Perl code is only compatible with AI-powered SAST. For more information about configuring scans for AI-powered SAST, see Analyzing using AI-powered SAST.

If AI-powered SAST is not configured, Perl code will only be scanned via Regular Expression Analysis and users may see suboptimal results.

22.2. Perl translation syntax

To include Perl code for analysis using AI-powered SAST, include all source files that you want to analyze.

Use the following basic command-line syntax to analyze Perl code:

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

See Specifying files and directories for more information.

Example

```
sourceanalyzer -b <build_id> "**/*.pl"
```

Important

Supported file extensions for the Perl source files are: `.pl`, `.pm`.

23. Analyzing Ruby code

This section describes how to analyze Ruby code. For projects combined with other files, see the relevant section for those languages.

- Ruby translation syntax

23.1. Ruby translation syntax

Ruby is only compatible with AI-powered SAST.

Note

If AI-Powered SAST is not configured, Ruby code will only be scanned via Regular Expression Analysis and users may see suboptimal results.

Use the following basic command-line syntax to analyze Ruby code using AI-powered SAST:

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

For example, specifying files individually:

```
sourceanalyzer -b <build_id> file1.rb file2.rb file3.erb
```

Alternatively use wildcards or directories to specify multiple files more easily:

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

For more information about configuring scans for AI-powered SAST, see the following topics:

- AI-powered SAST requirements
- Analyzing using AI-powered SAST

24. Analyzing Ada code

This section describes how to analyze Ada projects. For projects combined with other files, please see the relevant section for those languages.

This section contains the following topics:

- Ada analysis prerequisites
- Ada translation syntax

24.1. Ada analysis prerequisites

Currently, Ada code is only compatible with AI-powered SAST. For more information about configuring scans for AI-powered SAST, see Analyzing using AI-powered SAST.

If AI-powered SAST is not configured, Ada code will only be scanned via Regular Expression Analysis and users may see suboptimal results.

24.2. Ada translation syntax

To include Ada code for analysis using AI-powered SAST, include all source files that you want to analyze.

Use the following basic command-line syntax to analyze Ada code:

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

See Specifying files and directories for more information.

Example

```
sourceanalyzer -b <build_id> "**/*.ada"
```

Important

Supported file extensions for the Ada source files are: `.ada` , `.adb` , `.ads` .

25. Analyzing Delphi code

This section describes how to analyze Delphi projects. For projects combined with other files, see the relevant section for those languages.

This section contains the following topics:

- Delphi analysis prerequisites
- Delphi translation syntax

25.1. Delphi analysis prerequisites

Currently, Delphi code is only compatible with AI-powered SAST. For more information about configuring scans for AI-powered SAST, see Analyzing using AI-powered SAST.

If AI-powered SAST is not configured, Delphi code will only be scanned via Regular Expression Analysis and users may see suboptimal results.

25.2. Delphi translation syntax

To include Delphi code for analysis using AI-powered SAST, include all source files that you want to analyze.

Use the following basic command-line syntax to analyze Delphi code:

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

See Specifying files and directories for more information.

Example

```
sourceanalyzer -b <build_id> "**/*.pas"
```

Important

Supported file extensions for the Delphi source files are: `.pas`, `.dpr`, `.dpk`.

26. Analyzing Elixir code

This section describes how to analyze Elixir projects. For projects combined with other files, see the relevant section for those languages.

This section contains the following topics:

- Elixir analysis prerequisites
- Elixir translation syntax

26.1. Elixir analysis prerequisites

Currently, Elixir code is only compatible with AI-powered SAST. For more information about configuring scans for AI-powered SAST, see Analyzing using AI-powered SAST.

If AI-powered SAST is not configured, Elixir code will only be scanned via Regular Expression Analysis and users may see suboptimal results.

26.2. Elixir translation syntax

To include Elixir code for analysis using AI-powered SAST, include all source files that you want to analyze.

Use the following basic command-line syntax to analyze Elixir code:

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

See Specifying files and directories for more information.

Example

```
sourceanalyzer -b <build_id> "**/*.ex"
```

Important

Supported file extensions for the Elixir source files are: `.ex`, `.exs`.

27. Analyzing Erlang code

This section describes how to analyze Erlang projects. For projects combined with other files, see the relevant section for those languages.

This section contains the following topics:

- Erlang analysis prerequisites
- Erlang translation syntax

27.1. Erlang analysis prerequisites

Currently, Erlang code is only compatible with AI-powered SAST. For more information about configuring scans for AI-powered SAST, see Analyzing using AI-powered SAST.

If AI-powered SAST is not configured, Erlang code will only be scanned via Regular Expression Analysis and users may see suboptimal results.

27.2. Erlang translation syntax

To include Erlang code for analysis using AI-powered SAST, include all source files that you want to analyze.

Use the following basic command-line syntax to analyze Erlang code:

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

See Specifying files and directories for more information.

Example

```
sourceanalyzer -b <build_id> "**/*.erl"
```

Important

Supported file extensions for the Erlang source files are: `.erl`, `.hrl`.

28. Analyzing Lua code

This section describes how to analyze Lua projects. For projects combined with other files, see the relevant section for those languages.

This section contains the following topics:

- Lua analysis prerequisites
- Lua translation syntax

28.1. Lua analysis prerequisites

Currently, Lua code is only compatible with AI-powered SAST. For more information about configuring scans for AI-powered SAST, see Analyzing using AI-powered SAST.

If AI-powered SAST is not configured, Lua code will only be scanned via Regular Expression Analysis and users may see suboptimal results.

28.2. Lua translation syntax

To include Lua code for analysis using AI-powered SAST, include all source files that you want to analyze.

Use the following basic command-line syntax to analyze Lua code:

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

See Specifying files and directories for more information.

Example

```
sourceanalyzer -b <build_id> "**/*.lua"
```

29. Analyzing Kornshell code

This section describes how to analyze Kornshell projects. For projects combined with other files, see the relevant section for those languages.

This section contains the following topics:

- Kornshell analysis prerequisites
- Kornshell translation syntax

29.1. Kornshell analysis prerequisites

Currently, Kornshell code is only compatible with AI-powered SAST. For more information about configuring scans for AI-powered SAST, see Analyzing using AI-powered SAST.

If AI-powered SAST is not configured, Kornshell code will only be scanned via Regular Expression Analysis and users may see suboptimal results.

29.2. Kornshell translation syntax

To include Kornshell code for analysis using AI-powered SAST, include all source files that you want to analyze.

Use the following basic command-line syntax to analyze Bash code:

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

See Specifying files and directories for more information.

Example

```
sourceanalyzer -b <build_id> "**/*.ksh"
```

30. Analyzing Salesforce Apex and Visualforce code

- Apex and Visualforce translation prerequisites
- Apex and Visualforce command-line syntax

30.1. 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 Fortify 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 Fortify 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.

30.2. 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`.

31. 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.

- About downloading source files
- Importing the transport request
- Adding Fortify SAST to your Favorites list
- Running the Fortify ABAP Extractor
- Uninstalling the Fortify ABAP Extractor

31.1. About downloading source files

To translate ABAP code, the Fortify ABAP Extractor program downloads source files to the presentation server, and optionally, starts Fortify 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 `LIKE programname` is used to extract rows.

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

ABAP Properties

31.1.1. 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.

31.2. 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 Fortify 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 Fortify SAST can scan the ABAP code.

31.3. Adding Fortify SAST to your Favorites list

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

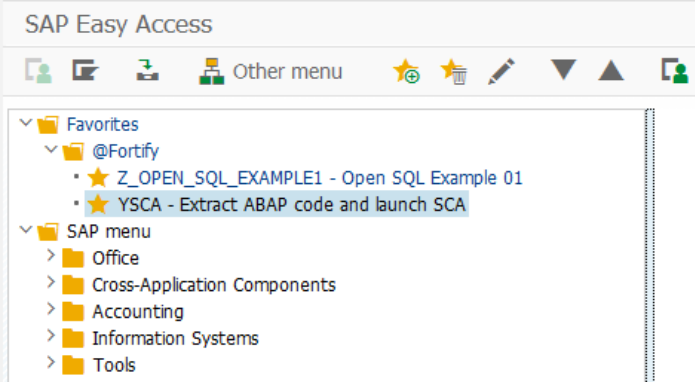
To add Fortify SAST to your Favorites list:

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.

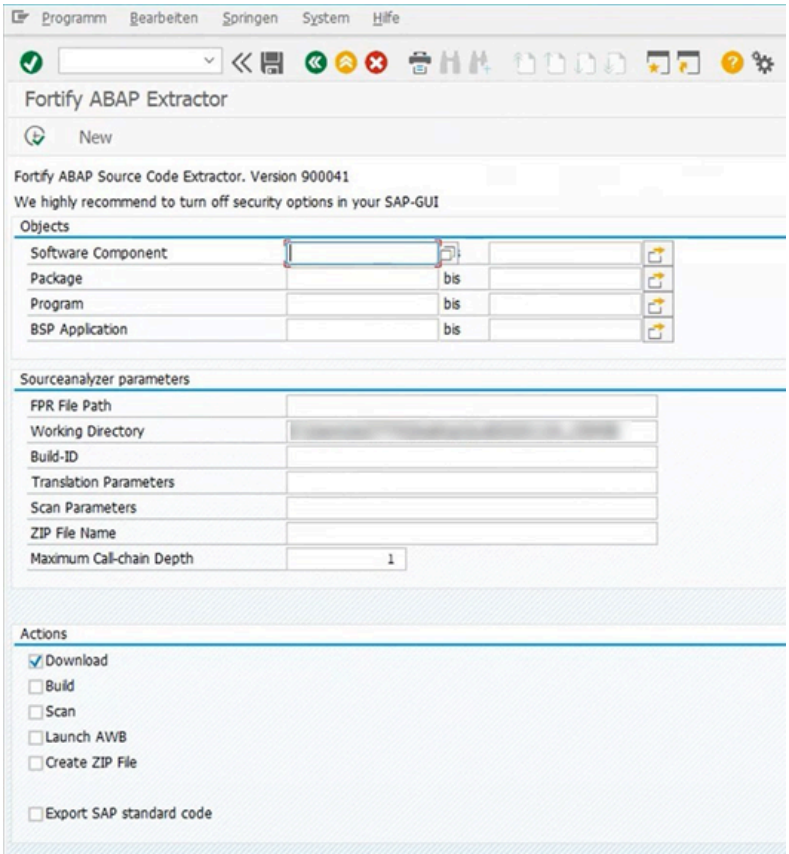
31.4. Running the Fortify ABAP Extractor

To run the Fortify ABAP Extractor:

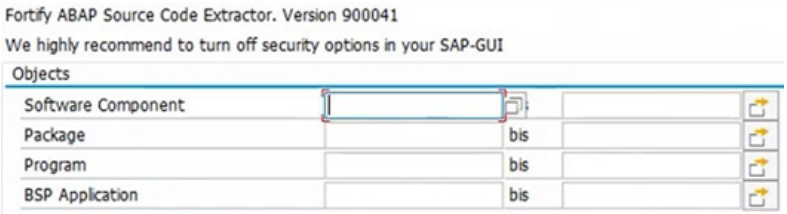
1. Start the Fortify ABAP Extractor from the **Favorites** link, the transaction code, or manually start the Extractor object.



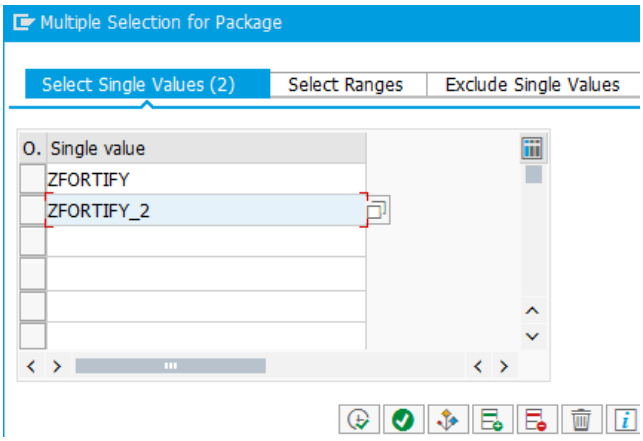
This opens the Fortify ABAP Extractor.



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.



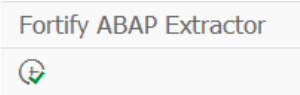
3. Provide the Fortify SAST-specific parameters described in the following table.

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. Fortify 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 Fortify 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 Fortify 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 Fortify SAST download the source code extracted from your SAP database.
Build	Select the **Build** check box to have Fortify SAST translate all downloaded ABAP code and store it using the specified build ID. This action requires that you have an installed version of Fortify 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 Fortify SAST is installed.
Scan	Select the **Scan** check box to have Fortify 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 Fortify 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 Fortify 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**.



31.5. 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

32. Analyzing COBOL code

COBOL can be scanned by using AI-powered SAST or traditional SAST. If AI-powered SAST and a database connection is not specified, Fortify SAST automatically reverts to performing local analysis of COBOL source files.

> **Note**
> AI-Powered SAST is single-file analysis and other differences with LLM-based analysis may mean that users may not receive the same results if scanning with AI-Powered SAST compared to traditional local scanning.

If you have AI-powered SAST enabled for other files and want to ensure local analysis of COBOL source files is still performed, you must make sure the following property is set during both the translation and analysis phases:

```
com.fortify.sca.cobol.legacy.enabled=true
```

Local traditional SAST uses COBOL translation to support modern COBOL dialects. Alternatively, we have a local traditional SAST legacy COBOL translation (see Using the Legacy COBOL Translation) that can be used across platforms but is compatible with fewer dialects.

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

32.1. Preparing COBOL source and copybook files for translation

Before you can analyze a COBOL program, copy the following program components to the system where you run Fortify 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 the SQL INCLUDE files that the COBOL source code references (a SQL INCLUDE file is technically a copybook file)

  > **Important**
  > For local analysis, the copybook files must have the extension `.CPY` or `.cpy`.
  > When using AI-powered SAST, copybook files additionally allow extensions `.copy`, `.cobcopy`, `.copybook`.

  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`.

When using local analysis, 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.

32.2. COBOL command-line syntax

AI-powered SAST

Use the following basic command-line syntax to analyze COBOL code using AI-powered SAST:

```
sourceanalyzer –b <build_id> <path>
```

For more information about configuring scans for AI-powered SAST, see Analyzing using AI-powered SAST.

Local translation (deprecated)

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
```

For more information, see Specifying Files and Directories

32.2.1. 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
```

32.2.2. 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).

32.2.3. 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 Fortify SAST looks for copybook files. **Note** This option does not accept wildcards. **Equivalent property name:** `com.fortify.sca.CobolCopyDirs`

32.3. Using Legacy COBOL translation (deprecated)

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

- You run Fortify 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).

32.3.1. 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 Fortify 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 Fortify 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 • Fortify SAST reports numerous parsing errors in the COBOL translation **Equivalent Property Name:** `com.fortify.sca.CobolFixedFormat`

33. Analyzing SQL

On Windows (and Linux for .NET projects only), Fortify 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 Fortify 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
```

33.1. 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
```

33.2. 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.

34. Analyzing Infrastructure as Code (IaC)

Fortify 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`. Fortify SAST supports other extensions only if they are not commonly used by other languages or file types (such as `.java` or `.html`).

By default, Fortify 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"
```

> **Important**
> Bicep requires an Internet connection on the scan machine to download dependencies to get optimal results. Using a scan machine without Internet connection may results in suboptimal results.

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"
```

> **Important**
> Terraform requires an to internet connection on the scan machine to download dependencies to get optimal results. Using a scan machine without internet connection may results in suboptimal results.

Translating JSON

Translating YAML

35. Analyzing JSON

By default, Fortify 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"
```

36. Analyzing YAML

By default, Fortify 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"
```

37. Analyzing Dockerfiles

By default, Fortify 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.

Fortify SAST accepts the following escape characters in Dockerfiles: backslash (`\`) and backquote (`` ` ``). If the escape character is not set in the Dockerfile, then Fortify 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, Fortify 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
```

38. Analyzing Bash code

This section describes how to analyze Bash projects. For projects combined with other files, see the relevant section for those languages.

This section contains the following topics:

- Bash analysis prerequisites
- Bash translation syntax

38.1. Bash analysis prerequisites

Currently, Bash code is only compatible with AI-powered SAST. For more information about configuring scans for AI-powered SAST, see Analyzing using AI-powered SAST.

If AI-powered SAST is not configured, Bash code will only be scanned via Regular Expression Analysis and users may see suboptimal results.

38.2. Bash translation syntax

To include Bash code for analysis using AI-powered SAST, include all source files that you want to analyze.

Use the following basic command-line syntax to analyze Bash code:

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

See Specifying files and directories for more information.

Example

```
sourceanalyzer -b <build_id> "**/*.sh"
```

Important

Supported file extensions for the Bash source files are: `.sh`, `.bash`.

39. Analyzing PowerShell code

This section describes how to analyze PowerShell projects. For projects combined with other files, see the relevant section for those languages.

This section contains the following topics:

- PowerShell analysis prerequisites
- PowerShell translation syntax

39.1. PowerShell analysis prerequisites

Currently, PowerShell code is only compatible with AI-powered SAST. For more information about configuring scans for AI-powered SAST, see Analyzing using AI-powered SAST.

If AI-powered SAST is not configured, PowerShell code will only be scanned via Regular Expression Analysis and users may see suboptimal results.

39.2. PowerShell translation syntax

To include PowerShell code for analysis using AI-powered SAST, include all source files that you want to analyze.

Use the following basic command-line syntax to analyze PowerShell code:

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

See Specifying files and directories for more information.

Example

```
sourceanalyzer -b <build_id> "**/*.ps1"
```

Important

Supported file extensions for the PowerShell source files are: `.ps1`, `.psm1`, `.psd1`, `.psd1xml`.

40. Analyzing R code

This section describes how to analyze R projects. For projects combined with other files, see the relevant section for those languages.

This section contains the following topics:

- R analysis prerequisites
- R translation syntax

40.1. R analysis prerequisites

Currently, R code is only compatible with AI-powered SAST. For more information about configuring scans for AI-powered SAST, see Analyzing using AI-powered SAST.

If AI-powered SAST is not configured, R code will only be scanned via Regular Expression Analysis and users may see suboptimal results.

40.2. R translation syntax

To include R code for analysis using AI-powered SAST, include all source files that you want to analyze.

Use the following basic command-line syntax to analyze R code:

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

See Specifying files and directories for more information.

Example

```
sourceanalyzer -b <build_id> "**/*.r"
```

Important

Supported file extensions for the R source files are: `.r`, `.R`.

41. 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

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

Managing compiler versions

Fortify SAST downloads compilers that are referenced in the code with the pragma statement from the Solidity compiler repository. By default, Fortify 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.

42. Analyzing other languages and configurations

- Analyzing Flex and ActionScript
- Analyzing ColdFusion code
- Analyzing ASP/VBScript virtual roots
- Classic ASP command-line example
- VBScript command-line example

42.1. 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.

42.1.1. 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.

Fortify 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

42.1.2. 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. Fortify 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"
```

42.1.3. 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 Fortify 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 Fortify SAST can complete the analysis.

42.2. 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.

42.2.1. 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**
>
> Fortify SAST calculates the relative path to each CFML source file with the `-source-base-dir` directory as the starting point. Fortify SAST uses these relative paths when it generates instance IDs. If you move the entire application source tree to a different directory, the Fortify SAST-generated instance IDs remain the same if you specify an appropriate parameter for the `-source-base-dir` option.

42.2.2. 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

42.3. Analyzing ASP/VBScript virtual roots

Fortify SAST allows you to handle ASP virtual roots. For web servers that use virtual directories as aliases that map to physical directories, Fortify 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 Fortify SAST, you must set the `com.fortify.sca.ASPVirtualRoots.name_of_virtual_directory` property in your Fortify 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 Fortify 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 Fortify SAST encounters the `#include` directive:

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

Fortify SAST determines if the project contains a physical directory named `Include`. If there is no such physical directory, Fortify SAST looks through its runtime properties and finds the `-Dcom.fortify.sca.ASPVirtualRoots.Include=`

`"C:\WebServer\CustomerOne\inc"` setting. Fortify 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 Fortify SAST command line as follows:
>
> `-Dcom.fortify.sca.ASPVirtualRoots=C:\WebServer\CustomerTwo\Stuff;`
>
> `C:\WebServer\CustomerOne\inc`

This prompts Fortify 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 Fortify 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.

42.4. Classic ASP command-line example

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

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

42.5. VBScript command-line example

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

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

43. 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 Fortify 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.

44. Scanning for Secrets

Fortify SAST scans are made up a series of analyzers, one of which is able to find information generally across any file type. This enables Fortify 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.

44.1. Regular Expression Analysis

Regular expression (regex) analysis provides the ability to use 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 Fortify 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
```

45. Scanning for PQC issues

Fortify SAST supports scanning for PQC (Post Quantum Cryptography) issues, particularly useful in identifying places where cryptographic algorithms are being used that are not resilient to PQC attacks.

Since PQC concerns are not open to every industry, these are disabled by default. To enable them you can specify the property:

```
com.fortify.sca.rules.enablePQCRules=true
```

in either `fortify-sca.properties` or on the command-line at scan time with `-Dcom.fortify.sca.rules.enablePQCRules=true`.

Refer OpenText SAST security content release information about which libraries and languages are supported based on the rulepacks you are using.

46. Optimizing results

This section provides guidelines and tips to optimize results when analyzing different codebases with Fortify SAST.

46.1. 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, Fortify 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. For more information, see Translation and Analysis Phase Properties.

46.2. 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** Fortify 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** Fortify 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** Fortify SAST applies rule ID filters in the initialization phase before any analysis has taken place.	`823FE039-A7FE-4AAD-B976-9EC53FFE4A59`
Priority1	The priority values in ascending order are `low`, `medium`, `high`, and `critical`. See Ranking Issue Severity for further details.	`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™ Fortify SAST Custom Rules Guide*.	`(SYSTEMINFO || EXCEPTIONINFO)`
		`(WEB || (DATABASE && PRIVATE))`
		`(NETWORK && !XSS)`

Impact1	See Ranking Issue Severity for further details.	`impact < 0.5`

Likelihood1	See Ranking Issue Severity for further details.	`likelihood <= 1.5`
Confidence1	See Ranking Issue Severity for further details.	`confidence < 1.8`
Probability1	See Ranking Issue Severity for further details.	`probability <= 1.2`
Accuracy1	See Ranking Issue Severity for further details.	`accuracy <= 1.0`
min_virtual_call_confidence1	An aggregate confidence value calculated by the engine based all virtual calls within a given issue's trace. Used within `confidence` scoring but can be used for more fine-grained filtering. This is a value between 0.4 and 1.0.	`min_virtual_call_confidence <= 0.75`
Analyzer	The valid values are `buffer`, `content`, `configuration`, `controlflow`, `dataflow`, `scripted`, `semantic`, and `structural`.	
File type	The valid file type values are `ABAP`, `ACTIONSCRIPT`, `ADA`, `APEX`, `ASPNET`, `ASP`, `ASPX`, `BYTECODE`, `C`, `CFML`, `CLOUDFORMATION`, `COBOL`, `CPP`, `CSHARP`, `DART`, `DELPHI`, `DOCKERFILE`, `ELIXIR`, `ERLANG`, `GO`, `GROOVY`, `HCL`, `HOCON`, `HTML`, `INI`, `JAVA`, `JAVA_PROPERTIES`, `JAVASCRIPT`, `JINJA`, `JSON`, `JSP`, `JSPX`, `JUPYTER`, `KOTLIN`, `LUA`, `MSIL`, `MXML`, `OBJC`, `OBJCPP`, `PERL`, `PHP`, `PLSQL`, `POWERSHELL`, `PYTHON`, `R`, `RUBY`, `RUST`, `SCALA`, `SOLIDITY`, `SWIFT`, `SWC`, `SWF`, `SQL`, `TERRAFORM`, `TLD`, `TSQL`, `TYPESCRIPT`, `VB`, `VB6`, `VBSCRIPT`, `VISUAL_FORCE`, `VUE`, `XML`, and `YAML`	

1For the priority and metadata filters, use less than (`<`) or less than or equal to (`<=`).

Composite Filters

When you specify a filter on different lines, Fortify 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.

Filter File Example

46.2.1. 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
```

46.3. 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, Fortify 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 Fortify 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 Fortify 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. Fortify 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 Fortify SAST loads.

46.4. 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 Fortify 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, Fortify SAST logs the issues that are removed, including their location and category.

> **Note**
> This functionality is available and enabled bv 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 automatically protects against the attack")
```

46.5. 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.

46.5.1. 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`

46.5.2. 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`.

46.5.3. 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() {
 ...
 }
}
```

47. Optimizing performance

This section provides guidelines and tips to optimize memory usage and performance when analyzing different types of codebases with Fortify SAST.

- Antivirus software
- Hardware considerations
- Tuning options
- Quick scan
- Configuring scan speed with speed dial
- Breaking down codebases

- Limiting analyzers and languages
- Optimizing FPR files
- Monitoring long running scans

47.1. Antivirus software

The use of antivirus software can negatively impact Fortify SAST performance. If you notice long scan times, OpenText recommends that you temporarily exclude the internal Fortify 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, Fortify 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 Fortify SAST you are using.

47.2. 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, Fortify SAST requires more capable hardware. This includes:

- **Disk I/O**—Fortify 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.

47.3. Tuning options

Fortify SAST can take a long time to process complex projects. The time is spent in different phases:

- Translation
- Analysis

Fortify SAST can produce large analysis result files (FPRs), which can take a long time to audit and upload to Fortify Software Security Center. 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

47.4. Quick scan

Quick scan mode provides a way to quickly scan your projects for critical- and high-priority issues. Fortify 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 Fortify SAST analysis sometimes depends on the available resources. Fortify 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 Fortify SAST has enough resources available.

Fortify SAST enables the user to control the "cutoff" point by using Fortify 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, Fortify 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 Fortify 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 Fortify 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 Fortify Software Security Center**—To avoid overwriting the results of a full scan, by default Fortify Software Security Center ignores uploaded FPR files scanned in quick scan mode. However, you can configure a Fortify Software Security Center application version so that FPR files scanned in quick scan are processed. For more information, see analysis results processing rules in the *OpenText™ Applicaton Security User Guide*.

47.5. 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, Fortify Software Security Center blocks uploaded analysis results that were created with a precision level less than four. However, you can configure your Fortify Software Security Center 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
```

47.6. 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`). Fortify 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 Fortify 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, Fortify SAST displays the tree for each binary.

47.7. 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 Fortify SAST translates, however, this may lead to suboptimal results.

- Disabling analyzers
- Disabling languages

47.7.1. Disabling analyzers

To disable specific analyzers, include the `-analyzers` option to Fortify 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 Fortify 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
```

47.7.2. 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 Fortify 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.

47.8. 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.

- Using filter files
- Using filter sets
- Excluding source code from the FPR
- Reducing the FPR file size
- Opening large FPR files

47.8.1. 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 Fortify 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.

47.8.2. Using filter sets

Filters in an issue template determine how the results from Fortify 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 Fortify 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. Fortify 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 Fortify SAST loads.

47.8.3. 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 Fortify 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
```

47.8.4. 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 Fortify Software Security Center. Fortify Software Security Center cannot merge it on the server because the FPR does not contain the Fortify 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.

47.8.5. 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 Fortify 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.

47.9. Monitoring long running scans

When you run Fortify 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 Fortify SAST is doing and how it is performing in real-time.

- Using the SCAState tool
- Using JMX tools

47.9.1. 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 Fortify SAST spends its time during the analysis phase. For more information about how to use SCAState, see the *Checking the Fortify SAST Scan Status*.

47.9.2. Using JMX tools

You can use tools to monitor Fortify SAST with JMX technology. These tools can provide a way to track Fortify 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.

- Using JConsole
- Using Java VisualVM

47.9.2.1. 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 Fortify SAST locally or remotely with the following command:

```
jconsole <host_name>:9090
```

47.9.2.2. 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
```

48. Using mobile build sessions

With an Fortify SAST mobile build session (MBS), you can translate a project on one machine and scan it on another. A common use case for this is to improve scan time, such that the translation can be performed on a build computer with fewer resources, and then a better equipped computer can be utilized for the scan.

> **Note**
> This is also necessary for several project types when utilizing Fortify ScanCentral SAST or OpenText Core Application Security (Fortify on Demand)

Using an MBS allows you to translate on one machine and do one of the following:

- Use Fortify ScanCentral SAST client to move the MBS to sensors for analysis (see Fortify ScanCentral SAST)
- Move the build session (MBS file) to another computer that has an Fortify SAST installation, import the MBS (see Importing a mobile build session), and then run the analysis.
- Provide the MBS file to OpenText Core Application Security (Fortify on Demand) for analysis

You must have the same version of Fortify Software Security Content (Rulepacks) installed on both the system where you perform the translation and the system where you perform the analysis.

> **Note**
> The scan machines may have dependencies for certain types of projects and may not be platform independent. For more information on dependencies that may be necessary on scan machines, see Software Requirements.
> Bicep, Solidity and Terraform (HCL) projects additionally require an Internet connection for optimal results.

- Mobile build session version compatibility
- Creating a mobile build session
- Importing a mobile build session

48.1. Mobile build session version compatibility

The Fortify SAST version on the translate machine must be compatible with the Fortify SAST version on the analysis machine. The version number format is *<major>.<minor>.<patch>.<build_number>* (for example, 26.3.0140). The *<major>* and *<minor>* portions of the Fortify SAST version numbers on both the translation and the analysis machines must match. For example, 26.3.0 and 26.3.x are compatible. To determine the Fortify SAST version number, type `sourceanalyzer -v` on the command line.

You can obtain the build ID and the Fortify SAST version from an MBS file with the following command:

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

48.2. 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 Fortify SAST mobile build session.

48.3. 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 Fortify SAST project root directory.

To import the mobile build session, type the following command:

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

After you import your Fortify 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 Fortify SAST installation, you can scan multiple build IDs in one scan with the `-b` option (see Analysis Phase).

> **Note**
> The scan machines may have dependencies for certain types of projects. See Software Requirements for dependencies that may be necessary on scan machines.
> Bicep and Terraform (HCL) projects additionally require an internet connection for optimal results.

49. Troubleshooting

- Exit codes
- Memory tuning
- Scanning complex functions
- Issue non-determinism
- Locating the log files
- Configuring log files
- Reporting issues and requesting enhancements

49.1. Exit codes

The following table describes the possible Fortify 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 Fortify 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, Fortify 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.

49.2. Memory tuning

The amount of physical RAM required for a scan depends on the complexity of the code. By default, Fortify 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.

- Java heap exhaustion
- Native heap exhaustion
- Stack overflow

49.2.1. Java heap exhaustion

Java heap exhaustion is the most common memory problem that might occur during Fortify SAST scans. It is caused by allocating too little heap space to the Java virtual machine that Fortify 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 Fortify 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 Fortify SAST Java virtual machine when you start the scan. To increase the heap size, use the `-Xmx` command-line option when you run the Fortify 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 Fortify 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 Fortify SAST is running) that the operating system might swap to disk. Allocating more physical memory to Fortify SAST than is available in the environment might cause "thrashing," which typically slows down the scan along with everything else on the system.

49.2.2. 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 Fortify 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.

49.2.3. 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 Fortify 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.

49.3. 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 Fortify SAST performs sometimes depends on the available resources. Fortify 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 Fortify 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 Fortify 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 Fortify SAST properties called limiters. Different analyzers have different limiters.

The following sections provide a discussion of a resolution for this issue.

- Dataflow Analyzer limiters
- Control Flow and Null Pointer analyzer limiters

49.3.1. 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 Fortify SAST.

To resolve the complexity identifier of `l`, you can adjust the following limiters in the Fortify 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. Fortify 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, Fortify SAST then proceeds with analysis at the lowest precision until either it finishes, or the complexity exceeds the `MaxTaintDefForVarAbort` limiter. In other words, Fortify SAST

tries harder at the lowest precision to get at least some results from the function. If Fortify 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`.

49.3.2. 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, Fortify 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 Fortify 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 Fortify 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.

49.4. 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
   ```

49.5. 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, Fortify 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 Fortify SAST that you are using.

The following table describes the Fortify 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 Fortify 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 Fortify SAST Support log file to Customer Support.

49.6. Configuring log files

You can configure the information that Fortify 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

49.7. 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: Fortify SAST
- Version number of Fortify SAST and any independent Fortify 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.

50. Command-line reference

This section describes general Fortify 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.

- Specifying files and directories
- Directives
- Translation options
- Analysis options
- Output options
- Other options

50.1. Specifying files and directories

File specifiers are expressions that allow you to pass a long list of files or a directory to Fortify SAST using wildcard characters. Fortify 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).

When you use wildcard patterns, files and directories whose names start with a period are included by default, even on platforms that treat these entries as hidden. To avoid including these files and directories, add explicit exclusions or set

`com.fortify.LegacyHiddenGlobSyntax=true` .

```
<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`

50.2. 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 Fortify 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.
`-regex-scan-minified`	Directs the regex analyzer to scan minified JavaScript and TypeScript files during regex analysis. By default, minified JavaScript and TypeScript files are not scanned by the regex analyzer.

50.2.1. LIM license directives

Fortify 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, Fortify 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 Fortify SAST uses the LIM for licensing. The proxy information is optional. Fortify SAST stores the pool password and the proxy credentials provided with this directive in the `fortify-sca.properties` file as encrypted data. If your license pool credentials change after you have installed Fortify 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 for exclusive use on this system for the specified duration (in minutes). This enables you to run Fortify 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.

50.3. 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. Fortify 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. Fortify 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 Fortify SAST first reads the source code file. Fortify 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, Fortify SAST uses `file.encoding` from the `java.io.InputStreamReader` constructor with no encoding parameter. In a few cases (for example with the ActionScript parser), Fortify SAST defaults to `UTF-8` encoding. **Equivalent property name:** `com.fortify.sca.InputFileEncoding`
`-nc`	When specified before a compiler command line, Fortify 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`, `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. Fortify 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`

50.4. 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 Fortify 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. Fortify 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 Fortify Software Security Center, 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. Fortify 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. Fortify 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, Fortify SAST includes all the files in the directory with the `.bin` and `.xml` extensions. **Equivalent property name:** `com.fortify.sca.RulesFile`

`-regex-scan-minified`	Directs the regex analyzer to scan minified JavaScript and TypeScript files during regex analysis. By default, minified JavaScript and TypeScript files are not scanned by the regex analyzer. **Equivalent property name:** `com.fortify.sca.regex.ScanMinifiedJS`

50.5. 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, Fortify 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 Fortify 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`

Output option	Description
`-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 Fortify Software Security Content information, command-line options, system properties, warnings, errors, and other information about the execution of Fortify 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, Fortify 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. Fortify 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. Fortify 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. Fortify SAST does not use this version for code analysis. **Equivalent property name:** `com.fortify.sca.BuildVersion`

Output option	Description
`-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 Fortify Software 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 Fortify Software Security Content information, command-line options, system properties, warnings, errors, and other information about the Fortify 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`

50.6. 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 Fortify 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 Fortify SAST Support log. **Equivalent property name:** `com.fortify.sca.DebugTrackMem`
`-verbose`	Sends verbose status messages to the console and to the Fortify SAST Support log file. **Equivalent property name:** `com.fortify.sca.Verbose`
`-logfile <file>`	Specifies the log file that Fortify SAST creates. For default log file locations, see Locating the log files. **Equivalent property name:** `com.fortify.sca.LogFile`

Other option	Description
`-clobber-log`	Directs Fortify SAST to overwrite the log file for each run of sourceanalyzer. Without this option, Fortify 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 Fortify SAST version and versions of various independent modules included with Fortify SAST (all other functionality is contained in Fortify 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 Fortify 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.

51. Configuration options

The Fortify SAST installer places a set of properties files on your system. Properties files contain configurable settings for Fortify SAST runtime analysis, output, and performance.

- Properties files
- fortify-sca.properties
- fortify-sca-quickscan.properties
- fortify-rules.properties

51.1. 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 Fortify SAST properties files. Property files for the Fortify SAST applications and tools are described in the *OpenText™ Application Security Tools Guide*.

Properties file name	Description	More information
`fortify-sca.properties`	Defines the Fortify SAST configuration properties.	fortify-sca.properties
`fortify-sca-quickscan.properties`	Defines the configuration properties applicable for an Fortify SAST quick scan.	fortify-sca-quickscan.properties
`fortify-rules.properties`	Defines the configuration properties that determine rule behavior.	fortify-rules.properties

51.1.1. 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
```

51.1.2. Overriding settings

Fortify 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 Fortify 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	Fortify 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.
	Fortify 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	Fortify SAST configuration file	Properties specified in the Fortify 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.

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

51.2. 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.

- Translation and analysis phase properties
- Regex analysis properties
- LIM license properties
- Rule properties
- Java and Kotlin properties
- Visual Studio and MSBuild project properties
- JavaScript and TypeScript properties

- Python properties
- Go properties
- Ruby properties
- COBOL properties
- PHP properties
- ABAP properties
- Flex and ActionScript properties
- ColdFusion (CFML) properties
- SQL properties
- Output properties
- Mobile build session (MBS) properties
- Proxy properties
- Logging properties
- Debug properties

51.2.1. 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.BuildI D	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 file provided with @<filename> (see Other Options). You can use this property, for example, to specify Unicode file paths in the options file. Valid encoding names are from the java.nio.charset.Charset This property is only valid in the fortify-sca.properties file and does not work in the fortify-sca-quickscan.properites file or with the -D option. **Value type:** String **Default:** JVM system default encoding **Example:** com.fortify.sca.CmdlineOptionsFileEncoding=UTF-8
com.fortify.sca.DISabledLanguages	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 . **Value type:** String **Default:** (none) **Command-line option:** -disable-language
com.fortify.sca.EnabledLanguages	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 . **Value type:** String **Default:** All languages in the specified source are translated unless explicitly excluded with the com.fortify.sca.DISabledLanguages property. **Command-line option:** -enable-language

`com.fortify.sca.DisableCompilerName`	If set to true, Fortify SAST includes build script files that have the same name as a build tool (such as gradlew) during translation as source files. **Value type:** Boolean **Default:** `false` **Command-line option:** `-disable-compiler-resolution`
`com.fortify.sca.ProjectRoot`	Specifies the directory to store intermediate files generated in the translation and analysis phases. Fortify SAST makes extensive use of intermediate files located in this project root directory. In some cases, you achieve better performance for analysis by making sure this directory is on local storage rather than on a network drive. **Value type:** String (path) **Default (Windows):** `${win32.LocalAppdata}/Fortify` `${win32.LocalAppdata}` is a variable that points to the Windows Local Application Data shell folder. **Default (non-Windows):** `$home/.fortify` **Command-line option:** `-project-root` **Example:** `com.fortify.sca.ProjectRoot=C:\Users\<username>\AppData\Local\`
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`
This is a partial list. For the complete list, see the properties file.
	Specifies how to translate specific file name extensions of languages that do not require build integration. The valid extension types are `ABAP`, `ACTIONSCRIPT`, `ADA`, `AI_RULES_FILES`, `APEX`, `APEX_OBJECT`, `APEX_TRIGGER`, `ARCHIVE`, `ASPNET`, `ASP`, `ASPX`, `BASH`, `BICEP`, `BITCODE`, `BSP`, `BYTECODE`, `CFML`, `CLOUDFORMATION`, `COBOL`, `CSHARP`, `DART`, `DELPHI`, `DOCKERFILE`, `ELIXIR`, `ERLANG`, `FLIGHT`, `FORTRAN`, `GENERIC`, `GO`, `GROOVY`, `HCL`, `HOCON`, `HTML`, `INI`, `JAVA`, `JAVA_PROPERTIES`, `JAVASCRIPT`, `JINJA`, `JSON`, `JSP`, `JSPX`, `JUPYTER`, `KOTLIN`, `LUA`, `MARKDOWN`, `MSIL`, `MXML`, `OBJECT`, `PERL`, `PHP`, `PLSQL`, `POWERSHELL`, `PYTHON`, `R`, `RUBY`, `RUBY_ERB`, `RUST`, `SCALA`, `SOLIDITY`, `SWIFT`, `SWC`, `SWF`, `TLD`, `SQL`, `TSQL`, `TYPESCRIPT`, `VB`, `VB6`, `VBSCRIPT`, `VISUAL_FORCE`, `VUE`, and `XML`, and `YAML`
	Value type: String (valid language type)
	Default: See the `fortify-sca.properties` file for the complete list.
	Examples:
	`com.fortify.sca.fileextensions.java=JAVA`
	`com.fortify.sca.fileextensions.cs=CSHARP`
	`com.fortify.sca.fileextensions.js=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_script>` to programmatically supply a language type. Provide a script that accepts one command-line parameter of a file name that matches the specified extension. The script must write the valid Fortify SAST file type (see previous list) to stdout and exit with a return value of zero. If the script returns a non-zero return code or the script does not exist, the file is not translated and Fortify SAST writes a warning to the log file.
	Example:
	`com.fortify.sca.fileextensions.jsp=oracle:<path_to_script>`

`com.fortify.sca.compilers.javac=com.fortify.sca.util.compilers.JavacCompiler`
`com.fortify.sca.compilers.c++=com.fortify.sca.util.compilers.GppCompiler`
`com.fortify.sca.compilers.make=com.fortify.sca.util.compilers.TouchlessCompiler`
`com.fortify.sca.compilers.mvn=com.fortify.sca.util.compilers.MavenAdapter`
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-sca.properties` file for the complete list.
	Example:
	To tell Fortify SAST that "my-gcc" is a gcc compiler:
	`com.fortify.sca.compilers.my-gcc=com.fortify.sca.util.compilers.GccCompiler`
	• **Notes:**
	• Compiler names can begin or end with an asterisk (*), which matches zero or more characters.
	• Execution of clang/clang++ is not supported with the gcc/g++ command names. You can specify the following:
	`com.fortify.sca.compilers.g++= com.fortify.sca.util.compilers.GppCompiler`

`com.fortify.sca.UseAntListener`	If set to true, Fortify SAST includes `com.fortify.dev.ant.SCAListener` in the compiler options. **Value type:** Boolean **Default:** `false`
`com.fortify.sca.exclude`	Specifies one or more files to exclude from translation. Separate multiple files with semicolons (Windows) or colons (non-Windows). See Specifying Files and Directories for more information on how to use file specifiers. **Value type:** String **Default:** Not enabled **Command-line option:** `-exclude` **Example:** `com.fortify.sca.exclude=file1.x;file2.x`

`com.fortify.sca.InputFileEncoding`	Specifies the source file encoding type. Fortify SAST allows you to scan a project that contains differently encoded source files. To work with a multi-encoded project, you must specify the `-encoding` option in the translation phase, when Fortify SAST first reads the source code file. Fortify SAST remembers this encoding in the build session and propagates it into the FVDL file. Typically, if you do not specify the encoding type, Fortify SAST uses `file.encoding` from the `java.io.InputStreamReader` constructor with no encoding parameter. In a few cases (for example with the ActionScript parser), Fortify 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.exe` system utility. Specify the paths in Windows syntax, not Cygwin syntax, even when you run Fortify SAST from within Cygwin. Escape backslashes with an additional backslash. **Value type:** String (path) **Default:** `reg` **Example:** `com.fortify.sca.RegExecutable=C:\\Windows\\System32\\reg.exe`
`com.fortify.sca.xcode.TranslateAfterError`	Specifies whether the xcodebuild touchless adapter continues translation if the xcodebuild subprocess exited with a non-zero exit code. If set to false, translation stops after encountering a non-zero xcodebuild exit code and the Fortify SAST touchless build halts with the same exit code. If set to true, the Fortify SAST touchless build executes translation of the build file identified prior to the xcodebuild exit, and Fortify SAST exits with an exit code of zero (unless some other error also occurs). Regardless of this setting, if xcodebuild exits with a non-zero code, then the xcodebuild exit code, stdout, and stderr are written to the log file. **Value type:** Boolean **Default:** `false`
Scan	

`com.fortify.sca.AddImpliedMethods`	If set to true, Fortify SAST generates implied methods when it encounters implementation by inheritance. **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 timeouts. **Value type:** Boolean **Default:** `true`
`com.fortify.sca.BinaryName`	Specifies a subset of source files to scan. Only the source files that were linked in the named binary at build time are included in the scan. **Value type:** String (path) **Default:** (none) **Command-line option:** `-bin` or `-binary-name`
`com.fortify.sca.DefaultAnalyzers`	Specifies a comma- or colon-separated list of the types of analysis to perform. The valid values for this property are `buffer`, `content`, `configuration`, `controlflow`, `dataflow`, `nullptr`, `semantic`, and `structural`. **Value type:** String **Default:** This property is commented out and all analysis types are used in scans. **Command-line option:** `-analyzers`
`com.fortify.sca.DisableFunctionPointers`	If set to true, disables function pointers during the scan. **Value type:** Boolean **Default:** `false`
`com.fortify.sca.EnableAnalyzer`	Specifies a comma- or colon-separated list of analyzers to use for a scan in addition to the default analyzers. The valid values for this property are `buffer`, `content`, `configuration`, `controlflow`, `dataflow`, `nullptr`, `semantic`, and `structural`. **Value type:** String **Default:** (none)

`com.fortify.sca.EnableSubtraceFiltering`	If set to true, filters out partial duplicates where issues are a subtrace of a given issue.
	For example, if the engine finds 2 similar issues with the traces:
	`A -> B -> C -> D`
	`B -> C -> D`
	The second issue is removed as a subtrace duplicate of the first, leaving only the longer issue, as it is the overall more accurate one.
	Value type: Boolean
	Default: true

`com.fortify.sca.ExitCodeLevel`	Extends the default exit code options. See Exit Codes for a description of the exit codes and the valid values for this property.
`com.fortify.sca.FilterFile`	Specifies the path to a filter file for the scan. See About Filter Files for more information. **Value type:** String (path) **Default:** (none) **Command-line option:** `-filter`
`com.fortify.sca.FilteredInstanceIDs`	Specifies a comma-separated list of IIDs to be filtered out using a filter file. **Value type:** String **Default:** (none) **Example:** `com.fortify.sca.FilteredInstanceIDs=CA4E1623A2424919B98EC19FCA279FFA,4418B3DC072647158B3758E6183C14CD`
`com.fortify.sca.FilteredRuleLanguages`	Specifies a comma- or colon-separated list of languages for which to remove rules. The valid language values are `abap`, `actionscript`, `apex`, `cfml`, `cobol`, `configuration`, `cpp`, `dart`, `dotnet`, `golang`, `objc`, `php`, `python`, `ruby`, `swift`, and `vb`. **Value type:** String **Default:** (none) **Example:** `com.fortify.sca.FileredRuleLanguages=apex:php`
`com.fortify.sca.MaxPassthroughChainDepth`	Specifies the length of a taint path between input and output parameters in a function call. **Value type:** Integer **Default:** `4`

`com.fortify.sca.MultithreadedAnalysis`	Specifies whether Fortify SAST runs in parallel analysis mode. **Value type:** Boolean **Default:** `true`
`com.fortify.sca.Phase0HigherOrder.Languages`	Specifies a comma-separated list of languages for which to run higher-order analysis. Higher-order analysis improves the ability to track dataflow through higher-order code, which is commonly used in modern dynamic languages. Valid values are `python`, `swift`, `ruby`, `javascript`, and `typescript`. **Value type:** String **Default:** `python,ruby,swift,javascript,typescript`
`com.fortify.sca.Phase0HigherOrder.Timeout.Hard`	Specifies the total time (in seconds) for higher-order analysis. When the analyzer reaches the hard timeout limit, it exits immediately. OpenText recommends this timeout limit in case some issue causes the analysis to run too long. OpenText recommends that you set the hard timeout to about 50% longer than the soft timeout, so that either the fixpoint pass limiter or the soft timeout occurs first. **Value type:** Number **Default:** `2700`
`com.fortify.sca.PrecisionLevel`	Specifies the scan precision. Scans with a lower precision level are performed faster. The valid values are `1`, `2`, `3`, and `4`. **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. This only affects scans on the local machine. If you upload the FPR to Fortify Software Security Center, it uses the issue template assigned to the application version. **Value type:** String **Default:** (none) **Command-line option:** `-project-template` **Example:** `com.fortify.sca.ProjectTemplate=test_issuetemplate.xml`

`com.fortify.sca.QuickScanMode`	If set to true, Fortify SAST performs a quick scan. Fortify SAST uses the settings from `fortify-sca-quickscan.properties`, instead of the `fortify-sca.properties` configuration file. **Value type:** Boolean **Default:** (not enabled) **Command-line option:** `-quick`
`com.fortify.sca.ScanPolicy`	Specifies the scan policy for prioritizing reported vulnerabilities (see Applying a Scan Policy to the Analysis). The valid 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 mode. Add this property only if you need to reduce the number of threads used because of a resource constraint. If you experience an increase in scan time or problems with your scan, a reduction in the number of threads used might solve the problem. **Value type:** Integer **Default:** (number of available processor cores)
`com.fortify.sca.TypeInferenceFunctionTimeout`	The amount of time (in seconds) that type inference can spend to analyze a single function. Unlimited if set to zero or is not specified. **Value type:** Long **Default:** `60`
`com.fortify.sca.TypeInferenceLanguages`	Comma- or colon-separated list of languages that use type inference. This setting improves the precision of the analysis 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 type inference can spend in phase 0 (the interprocedural analysis). Unlimited if set to zero or is not specified. **Value type:** Long **Default:** `300`

`com.fortify.sca.UniversalBlacklist`	Specifies a colon-separated list of functions to hide from all analyzers. **Value type:** String **Default:** `.*yyparse.*`
`com.fortify.LegacyHiddenGlobSyntax`	Specifies whether wildcards in the file specifiers, such as `*` and `**`, include hidden files and directories on non-Windows platforms. In earlier legacy product versions, Linux files and directories whose names started with a period (`.`) were not matched unless you explicitly specified them on the command line. Use this property to re-enable that legacy behavior. By default, the current behavior includes all files and directories except those excluded because they are known to cause issues or add unnecessary data. Value type: Boolean Default: `false`

51.2.2. 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`
`com.fortify.sca.regex.ScanMinifiedJS`	Scans minified JavaScript and TypeScript files during regex analysis. By default, Fortify SAST excludes these files from regex analysis. **Value type:** Boolean **Default:** `False`

Regular Expression Analysis

51.2.3. 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

Property name	Description
`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, Fortify SAST waits for a LIM license to become available before starting a translation or scan. If this property is set to false, Fortify SAST aborts if it cannot obtain a LIM license. **Value type:** Boolean **Default:** `true`

LIM License Directives

51.2.4. 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. Fortify 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. Fortify 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

Property name	Description
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

51.2.5. 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 Fortify 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`

Property name	Description
`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, Fortify SAST only translates Java source files that are referenced by the target file list. Otherwise, Fortify 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`

Property name	Description
`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

51.2.6. 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

51.2.7. 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, Fortify SAST generates JavaScript code to model the DOM tree that an HTML file generated during the translation phase and identifies DOM-related issues (such as cross-site scripting issues). Enable this property if the code you are translating includes HTML files that have embedded or referenced JavaScript code. 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 additional coma-separated HTML tags names for Fortify 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 Fortify SAST can download referenced JavaScript files for the scan. Delimit the URLs with vertical bars. **Value type:** String **Default:** (none) **Example:** `com.fortify.sca.JavaScript.src.domain.whitelist=http://www.xyz.com|http://www.123.org`
`com.fortify.sca.DisableJavascriptExtraction`	If set to true, JavaScript code embedded in JSP, JSPX, PHP, and HTML files is not extracted and not scanned. **Value type:** Boolean **Default:** `false`
`com.fortify.sca.EnableTranslationMinifiedJS`	If set to true, enables translation for minified JavaScript files. **Value type:** Boolean **Default:** `false`

Property name	Description
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 that are not translated. You can use regular expressions in the file names. Note that the regular expression `'(-\d\.\d\.\d)?'` is automatically inserted before `.min.js` or `.js` for each file name included in the `com.fortify.sca.skip.libraries.jQuery` property value. **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, jquery-ui.js,jquery-ui.min.js, jquery.mobile.js,jquery.mobile.min.js, jquery.color.js,jquery.color.min.js, jquery.color.svg-names.js, jquery.color.svg-names.min.js, jquery.color.plus-names.js, jquery.color.plus-names.min.js, jquery.tools.min.js` • javascript: `bootstrap.js,bootstrap.min.js,typescript.js,typescriptServices.js` • typescript: `typescript.d.ts,typescriptServices.d.ts`
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 imported node_modules are included in the translation. This property is only applied if `com.fortify.sca.exclude.node.modules` is set to false. **Value type:** Boolean **Default:** `true`

Translating JavaScript and TypeScript Code

51.2.8. 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. Fortify 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. Fortify 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 Fortify SAST does not automatically discover Django templates. **Value type:** Boolean **Default:** (none) **Command-line option:** `-django-disable-autodiscover`

Property name	Description
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 Fortify SAST does not automatically discover Django or Jinja2 templates. **Value type:** Boolean **Default:** (none) **Command-line option:** -disable-template-autodiscover

Translating Python Code

51.2.9. 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)

Translating Go Code

51.2.10. 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

51.2.11. 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.cobol.legacy.enabled`	If set to true, local COBOL translation is enabled and AI COBOL is disabled. If set to false, local COBOL translation is disabled and AI COBOL is enabled. If not set, AI COBOL is enabled if `com.fortify.sca.ai.provider` or `com.fortify.sca.ai.db.url` is set. Otherwise, local COBOL translation is enabled. **Default:** null
`com.fortify.sca.CobolCopyDirs`	Specifies one or more semicolon- or colon-separated directories where Fortify 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 Fortify 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`

Property name	Description
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

51.2.12. 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

51.2.13. 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, Fortify SAST adds ABAP statements to debug messages. **Value type:** Boolean **Default:** (none)
com.fortify.sca.AbapIncludes	When Fortify SAST encounters an ABAP 'INCLUDE' directive, it looks in the named directory. **Value type:** String (path) **Default:** (none)

51.2.14. 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

51.2.15. 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, Fortify 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

51.2.16. 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

51.2.17. 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, Fortify 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, Fortify 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. Fortify 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. Fortify SAST does not use this name but includes it in the results. **Value type:** String **Default:** (none) **Command-line option:** `-build-project`

Property name	Description
`com.fortify.sca.BuildVersion`	Specifies a version number for the scanned project. Fortify 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 Fortify 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 Fortify Software 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`

Property name	Description
com.fortify.sca.FVDLDisableProgramData	If set to true, excludes the ProgramData section from the analysis results file (FVDL). **Value type:** Boolean **Default:** false **Command-line option:** -fvdl-no-progdata
com.fortify.sca.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

51.2.18. 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, Fortify SAST does not copy source files into the build session directory. **Value type:** Boolean **Default:** true
com.fortify.sca.ExtractMobileInfo	If set to true, Fortify SAST extracts the build ID and the Fortify SAST version number from the mobile build session. **Note** Fortify SAST does not extract the mobile build with this property. **Value type:** Boolean **Default:** false

Mobile Build Sessions

51.2.19. 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)

51.2.20. 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.ProjectRoot}/log/sca_FortifySupport.log` **Command-line option:** `-logfile`
`com.fortify.sca.LogLevel`	Specifies the minimum log level for both log files. The valid values are `DEBUG`, `INFO`, `WARN`, `ERROR`, and `FATAL`. For more information, see Accessing Log Files and Configuring Log Files. **Value type:** String **Default:** `INFO`
`com.fortify.sca.ClobberLogFile`	If set to true, Fortify 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, Fortify SAST writes performance-related data to the Fortify SAST Support log file after the scan is complete. This value is automatically set to true when in debug mode. **Value type:** Boolean **Default:** `false`

Configuring Log Files

51.2.21. 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 Fortify 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 Fortify 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 Fortify 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, Fortify SAST monitors its memory use and warns when JVM garbage collection becomes excessive. **Value type:** Boolean **Default:** `true`

51.3. fortify-sca-quickscan.properties

Fortify 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.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 Fortify 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 Fortify Software Security Center, 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`

Property name	Description
`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`

Property name	Description
`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`

51.4. 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 Fortify 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)

52. Command-line tools

Fortify SAST command-line tools enable you to manage Fortify Software 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 Fortify SAST.

Note

By default, log files for Fortify 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 Fortify Software 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 Fortify SAST, specify a locale, and specify a proxy server for security content updates and for Fortify Software Security Center.	Running the post-install tool
SCAState	Provides state analysis information on the JVM during the analysis phase	Checking the scan status with SCAState

- About updating OpenText Application Security Content
- Checking the scan status with SCAState

52.1. 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 Fortify 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 Fortify SAST installation. If your installation is current, a message is displayed to that effect.

- Updating OpenText Application Security Content
- fortifyupdate command-line options

52.1.1. 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 Fortify 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 Fortify Software Security Center server using credentials:

```
fortifyupdate -url <ssc_url> -sscUser <username> -sscPassword <password>
```

52.1.2. 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 Fortify Software 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.

fortifyupdate option	Description
`-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 Fortify Software Security Center server by providing a Fortify Software Security Center URL.
Specify one of the following types of credentials if you update security content from Fortify Software Security Center with the `-url` option:	
`-sscUsername` `-sscPassword`	Specifies a Fortify Software Security Center user account by user name and password.
`-sscAuthToken`	Specifies a Fortify Software Security Center authentication token of type UnifiedLoginToken, CIToken, or ToolsConnectToken.

52.2. 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>]
```

SCAState command-line options

52.2.1. 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 Fortify 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 Fortify SAST process ID. Use this option if there are multiple Fortify 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.

SCAState option	Description
`-properties`	Displays configuration settings (this does not include sensitive information such as passwords).
`-scaversion`	Displays the Fortify 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 Fortify 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

Fortify 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.

53. Installation and upgrade notes

Complete instructions for installing Fortify SAST are provided in the *OpenText SAST User Guide*.

- The default scan policy has been updated to improve filtering of low accuracy buffer overflow issues. The scan policies are now commented to provide further details about what is filtered and when that change was introduced.
- Starting in version 26.3, Fortify SAST uses a new Kotlin translator.
  If you face any issues with the new translator, you can enable the legacy Kotlin translator with the `-use-k1` command-line option and report the issues to Customer Support.

© Copyright 2026 Open Text

For more info, visit https://docs.microfocus.com