Micro Focus Fortify Static Code Analyzer

Software Version: 18.10

Performance Guide

Document Release Date: May 2018
Software Release Date: May 2018

Contents

Preface 5

- Contacting Micro Focus Fortify Customer Support 5
- For More Information 5
- About the Documentation Set 5

Change Log 6

Chapter 1: Introduction 7

- Hardware Recommendations 7
- Sample Scans 8
- Related Documents 8
  - All Products 9
  - Micro Focus Fortify Software Security Center 9
  - Micro Focus Fortify Static Code Analyzer 10

Chapter 2: Performance Improvement Tips 11

- Hardware Considerations 11
- Tuning Options 11
- Mobile Build Sessions 12
- Memory Tuning 13
  - Java Heap Exhaustion 14
  - Native Heap Exhaustion 14
  - Stack Overflow 15

Chapter 3: Scan Quality and Performance 16

- Breaking Down Codebases 16
- Quick Scan 16
  - Limiters 17
  - Using Quick Scan and Full Scan 17
- Limiting Analyzers and Languages 17
  - Disabling Analyzers 18
  - Disabling Languages 18
- Scanning Complex Functions 19

Chapter 4: Optimizing FPR Files 20

- Filters 20

Filter Files 20
Scan-Time Filters 21
Excluding Source Code from the FPR 21
Reducing the FPR File Size 22
Opening Large FPR Files 23

Chapter 5: Monitoring Long Running Scans 25
Using the SCAState Utility 25
Using JMX Tools 25
Using JConsole 25
Using Java VisualVM 26

Chapter 6: Troubleshooting 27
Function too Complex to Analyze Message 27
Dataflow Analyzer Limiters 27
Control Flow and Null Pointer Analyzer Limiters 28

Send Documentation Feedback 30

Preface

Contacting Micro Focus Fortify Customer Support

If you have questions or comments about using this product, contact Micro Focus Fortify Customer Support using one of the following options.

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

https://softwaresupport.softwaregrp.com

To Call Support

1.844.260.7219

For More Information

For more information about Fortify software products:
https://software.microfocus.com/solutions/application-security

About the Documentation Set

The Fortify Software documentation set contains installation, user, and deployment guides for all Fortify Software products and components. In addition, you will find technical notes and release notes that describe new features, known issues, and last-minute updates. You can access the latest versions of these documents from the following Micro Focus Product Documentation website:

https://www.microfocus.com/support-and-services/documentation

Change Log

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

Software Release / Document Version	**Changes**
18.10	Updated: • "Sample Scans" on page 8 - Table updated to show data for the current release
17.20	Updated: • "Sample Scans" on page 8 - Table updated to show data for the current release • "Opening Large FPR Files" on page 23 - Added a description of the property used to increase memory for external utilities Removed: • "Parallel Processing" - In this release, Parallel Analysis Mode is enabled by default.
17.10	Updated: • "Sample Scans" on page 8 - Table updated to show data for the current release • "Tuning Options" on page 11 - Added the new parallel processing option • Replaced "CPUs, Parallel Processing, and Multithreading" with "Parallel Processing" to describe the new parallel processing implementation

Chapter 1: Introduction

This document provides guidelines and tips to optimize memory usage and performance when scanning different types of codebases with Fortify Static Code Analyzer.

This section contains the following topics:

Hardware Recommendations 7

Sample Scans 8

Related Documents 8

Hardware Recommendations

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 such as:

- Code type
- Size of the codebase
- Ancillary languages used (such as JSP, JavaScript, HTML)
- Number of vulnerabilities
- Type of vulnerabilities (analyzer used)
- Complexity of the codebase

The information in this guide is based on general guidelines to provide a set of “best guess” recommendations for hardware requirements. Fortify developed these guidelines from results gathered from real-world application scans. It is important to note that there might be cases where your codebase scan requires more than these guidelines imply. To improve these guidelines, Fortify welcomes your feedback on how your project requirements map to our guidelines.

The following table provides recommendations based on the complexity of the application.

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 utility.
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.

Application Complexity	CPU Cores	RAM (GB)	Average Scan Time	Description
Very Complex	32	256	7+ days	A system that delivers content such as an application server, database server, or content management system.

Note: JavaScript scans increase the analysis time significantly. If the total lines of code in an application consists of more than 20% JavaScript, use the next highest recommendation.

Sample Scans

These sample scans were performed using Fortify Static Code Analyzer version 18.10 on a dedicated Linux virtual machine with four CPUs and 32 GB of RAM. The following table shows the scan times you can expect for several common open-source projects.

Project Name / Language	Scan Time (Min:Sec)	Total Issues (2018R1)	LOC
Apache-HTTPd (C/C++)	07:35	1,941	32,562
Azureus 2 (Java)	18:08	7,710	95,546
WebGoat 7.0 (Java)	01:32	413	3,869
WordPress (Java)	02:41	777	10,437
CakePHP (PHP)	03:25	2,422	54,548
phpBB 3 (PHP)	03:03	1,276	39,749
SmartStoreNET (.NET)	47:06	4,616	208,349

Related Documents

This topic describes documents that provide information about Micro Focus Fortify software products.

Note: You can find the Micro Focus Fortify Product Documentation at https://www.microfocus.com/support-and-services/documentation.

All Products

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

Document / File Name	Description
About Micro Focus Fortify Product Software Documentation About_Fortify_Doc_<*version*>.pdf	This paper provides information about how to access Micro Focus Fortify product documentation. **Note:** This document is included only with the product download.
Micro Focus Fortify Software System Requirements Fortify_Sys_Reqs_<*version*>.pdf Fortify_Sys_Reqs_Help_<*version*>	This document provides the details about the environments and products supported for this version of Fortify Software.
Micro Focus Fortify Software Release Notes FortifySW_RN_<*version*>.txt	This document provides an overview of the changes made to Fortify Software for this release and important information not included elsewhere in the product documentation.
What's New in Micro Focus Fortify Software <version> Fortify_Whats_New_<*version*>.pdf Fortify_Whats_New_Help_<*version*>	This document describes the new features in Fortify Software products.
Micro Focus Fortify Open Source and Third-Party License Agreements Fortify_OpenSrc_<*version*>.pdf	This document provides open source and third-party software license agreements for software components used in Fortify Software.

Micro Focus Fortify Software Security Center

The following documents provide information about Fortify Software Security Center. Unless otherwise noted, these documents are available on the Micro Focus Product Documentation website.

Document / File Name	Description
Micro Focus Fortify Software Security Center User Guide SSC_Guide_<version>.pdf SSC_Help_<version>	This document provides Fortify Software Security Center users with detailed information about how to deploy and use Software Security Center. It provides all of the information you need to acquire, install, configure, and use Software Security Center.

Document / File Name	Description
	It is intended for use by system and instance administrators, database administrators (DBAs), enterprise security leads, development team managers, and developers. Software Security Center provides security team leads with a high-level overview of the history and current status of a project.

Micro Focus Fortify Static Code Analyzer

The following documents provide information about Fortify Static Code Analyzer. Unless otherwise noted, these documents are available on the Micro Focus Product Documentation website.

Document / File Name	Description
Micro Focus Fortify Static Code Analyzer Installation Guide SCA_Install_<*version*>.pdf SCA_Install_Help_<*version*>	This document contains installation instructions for Fortify Static Code Analyzer and Applications.
Micro Focus Fortify Static Code Analyzer User Guide SCA_Guide_<*version*>.pdf SCA_Help_<*version*>	This document describes how to use Fortify Static Code Analyzer to scan code on many of the major programming platforms. It is intended for people responsible for security audits and secure coding.
Micro Focus Fortify Static Code Analyzer Performance Guide SCA_Perf_Guide_<*version*>.pdf SCA_Perf_Help_<*version*>	This document provides guidelines for selecting hardware to scan different types of codebases and offers tips for optimizing memory usage and performance.
Micro Focus Fortify Static Code Analyzer Custom Rules Guide SCA_Cust_Rules_Guide_<*version*>.zip SCA_Cust_Rules_Help_<*version*>	This document provides the information that you need to create custom rules for Fortify Static Code Analyzer. This guide includes examples that apply rule-writing concepts to real-world security issues. **Note:** This document is included only with the product download.

Chapter 2: Performance Improvement Tips

This section contains different methods of tuning the Fortify Static Code Analyzer to maximize its performance.

This section contains the following topics:

Hardware Considerations 11
Tuning Options 11
Mobile Build Sessions 12
Memory Tuning 13

Hardware Considerations

The system requirements are documented in the *Micro Focus Fortify Software System Requirements* document. However, for large and complex applications, Fortify Static Code Analyzer requires more capable hardware. This includes:

- **Disk I/O**—Fortify Static Code Analyzer is I/O intensive so the faster the hard drive, the more savings on the I/O transaction. Fortify 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" on page 13 for more information about how to determine the amount of memory required for optimal performance.
- **CPU**—Fortify recommends a 2.1 GHz processor or faster.

Tuning Options

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

- Translation
- Analysis

The Fortify Static Code Analyzer analysis results can be large and therefore cause a long time to audit and upload to Micro Focus Fortify Software Security Center. This is referred to as the following phase:

- Audit/upload

The following table provides 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" below
Analysis	`-Xmx<size>M|G`	Set maximum heap size	"Memory Tuning" on the next page
Analysis	`-Xss<size>M|G`	Set stack size for each thread	"Memory Tuning" on the next page
Analysis	`-bin`	Scan the files related to a binary	"Breaking Down Codebases" on page 16
Analysis	`-quick`	Run a quick scan	"Quick Scan" on page 16
Analysis Audit/upload	`-filter <file>`	Apply a filter using a filter file	"Filters" on page 20
Analysis Audit/upload	`-disable-source-bundling`	Exclude source files from the FPR file	"Excluding Source Code from the FPR" on page 21

Mobile Build Sessions

With a Fortify Static Code Analyzer mobile build session (MBS), you can translate a project on one machine and analyze it on a different machine with better hardware. With an MBS, you can perform the translation on the original computer and then move the build session to a better equipped computer to perform the scan. The developers can run translations on their own computers and use only one powerful computer to run large scans.

Fortify strongly recommends that you move MBS files with the Fortify Static Code Analyzer export and import options.

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

Below is an example of the steps required to use an MBS. This example runs the translation on **Machine T** and the scan on **Machine S**.

1. **Machine T**: Translate the source files.

   ```
   sourceanalyzer -b <build_id> <src_files>
   ```

2. **Machine T**: Package and save an MBS to a file called `build-session.mbs`.

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

3. Transfer `build-session.mbs` from **Machine T** to **Machine S**.
4. **Machine S**: Import the MBS into the Fortify Static Code Analyzer project root directory on the scan machine.

   ```
   sourceanalyzer -import-build-session build-session.mbs
   ```

5. **Machine S**: Perform a scan with the same build ID that was used in the translation.

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

You cannot merge multiple mobile build sessions into a single build session. Each exported build session must use a unique build ID and you must import it under that unique build ID. However, after all of the build IDs are on the same Fortify Static Code Analyzer installation, you can scan multiple build IDs in one scan with the `-b` option.

For example, assuming all the build IDs were imported to the local machine using mobile build sessions, you can use the following command to scan:

```
sourceanalyzer -b <build_id_1> -b <build_id_2>  -b <build_id_3> -scan -f
myResults.fpr
```

Note: The resulting FPR (`myResults.fpr`) covers the same files as if they were all translated into one build ID. However, there are rare instances where dataflow between the files is lost if they are not translated together.

Memory Tuning

As discussed in "Hardware Recommendations" on page 7, the amount of physical RAM required for a scan depends on the complexity of the code itself. By default Fortify Static Code Analyzer automatically allocates the memory it uses based on the physical memory available on the system. This is generally sufficient.

As described in the *Micro Focus Fortify Static Code Analyzer User Guide*, you can adjust the Java heap size with the `-Xmx` command-line 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. Heap sizes smaller than 32 GB are optimized by the JVM. If your scan requires more than 32 GB, then you probably need much more than 48 GB such as 64 GB or higher. As a guideline, assuming no other memory intensive processes are running, do not allocate more than 2/3 of the available memory.

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

Java heap exhaustion is the most common memory problem during Fortify Static Code Analyzer scans and is the result of allocating too little heap space to the Java virtual machine that Fortify Static Code Analyzer uses for scanning the project. Identify Java heap exhaustion from the following symptom.

Symptom

One or more of these messages appears in the Fortify Static Code Analyzer 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 Static Code Analyzer Java virtual machine when you start the scan. To increase the heap size, use the `-Xmx` command-line option when you run the Fortify Static Code Analyzer 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.

Fortify recommends that the value of `-Xmx` not exceed either 90% of the total physical memory or the total physical memory minus 1.5 GB to allow for the operating system. If the system is dedicated to running Fortify Static Code Analyzer, 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 Static Code Analyzer is running) that the operating system might swap to disk. Allocating more physical memory to Fortify Static Code Analyzer than is available in the environment might cause “thrashing,” which typically slows down the scan along with everything else on the system.

Native Heap Exhaustion

Native heap exhaustion is a rare scenario in which the Java virtual machine is able to 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 Static Code Analyzer 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.

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 Static Code Analyzer 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.

Chapter 3: Scan Quality and Performance

This section contains the following topics:

Breaking Down Codebases 16
Quick Scan 16
Limiting Analyzers and Languages 17
Scanning Complex Functions 19

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 very little interactions, you can translate and scan the modules separately. The caveat to this is that you might lose dataflow 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`), and Fortify Static Code Analyzer 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 useful Fortify Static Code Analyzer command-line options for breaking 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 used 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 of 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 Static Code Analyzer displays the tree for each binary.

Quick Scan

Quick Scan mode provides a way to quickly scan your projects for major defects. By default, Quick Scan searches for high ‑confidence, high ‑severity issues. Although the Quick Scan is significantly faster, it does not provide a robust result set.

Limiters

The depth of the Fortify Static Code Analyzer analysis sometimes depends on the available resources. Fortify Static Code Analyzer 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 Static Code Analyzer has enough resources available.

Fortify Static Code Analyzer enables the user to control the "cutoff" point by using Fortify Static Code Analyzer limiter properties. The different analyzers have different limiters. You can run a predefined set of these limiters using a Quick Scan. See the quick scan properties in the *Micro Focus Fortify Static Code Analyzer User Guide* for descriptions of the limiters.

To enable Quick Scan, use `-quick` option with `-scan` option. With Quick Scan enabled, Fortify Static Code Analyzer applies the properties from the `<sca_install_dir>/Core/config/fortify-sca-quickscan.properties` file, in addition to the standard `<sca_install_dir>/Core/config/fortify-sca.properties` file. You can adjust the limiters that Fortify Static Code Analyzer uses by editing the `fortify-sca-quickscan.properties` file. If you modify `fortify-sca.properties`, it also affects quick scan behavior. Fortify 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.

Using Quick Scan and Full Scan

- **Run periodic full scans**—A periodic full scan is important as it might find issues that Quick Scan 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 code base, then load the Quick Scan results in Audit Workbench and merge it into the full scan. Group the issues by **New Issue** to produce a list of issues found in the full scan but not found in the Quick Scan.
- **Quick Scans and Micro Focus Fortify Software Security Center Server**—To avoid overwriting the results of a full scan, by default Fortify Software Security Center does not accept FPR files scanned using Quick Scan. However, you can configure Fortify Software Security Center so that FPR files scanned with Quick Scan are not blocked for an application version. For more information, see the *Micro Focus Fortify Software Security Center User Guide*.

Limiting Analyzers and Languages

Occasionally, you might find that a significant amount of the scan time is spent either running one particular analyzer or analyzing a particular language. It is also possible that this particular analyzer or language is not of great interest to your security requirements. You can limit the specific analyzers that run and the specific languages that are translated.

Disabling Analyzers

To disable specific analyzers, include the `-analyzers` option to Fortify Static Code Analyzer at scan time with a colon- or comma-separated list of analyzers you want to enable. The full list of analyzers is: `buffer`, `content`, `configuration`, `controlflow`, `dataflow`, `findbugs`, `nullptr`, `semantic`, and `structural`.

For example, to run a scan using only the Dataflow, Control Flow, and Buffer analyzers, use the following scan command:

```
sourceanalyzer -b <build_id> -analyzers dataflow:controlflow:buffer -scan
-f myResults.fpr
```

You can also do the same thing by setting `com.fortify.sca.DefaultAnalyzers` in the Fortify Static Code Analyzer property file *`<sca_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
```

Disabling Languages

To disable specific languages, include the `-disable-language` option in the translation phase. This is followed by a colon-separated list of languages that you want to disable. The full list of valid language parameters is:

```
abap, actionscript, apex, cfml, cpp, cobol, configuration, dotnet, java,
javascript, jsp, objc, php, plsql, python, ruby, scala, sql, swift, tsql,
vb
```

For example, to perform a translation that excludes SQL and PHP files, use the following command:

```
sourceanalyzer -b <build_id> <src_files> -disable-language sql:php
```

You can also disable languages by setting the `com.fortify.sca.DISabledLanguages` property in the Fortify Static Code Analyzer properties file *`<sca_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=sql:php
```

Scanning Complex Functions

During a Fortify Static Code Analyzer 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>)
```

To see a discussion of a resolution for this issue, see "Function too Complex to Analyze Message" on page 27.

Chapter 4: Optimizing FPR Files

This chapter describes how to handle performance issues related to the audit results (FPR) file. This includes reducing the scan time, reducing FPR file size, and tips for opening large FPR files.

This section contains the following topics:

Filters 20
Excluding Source Code from the FPR 21
Reducing the FPR File Size 22
Opening Large FPR Files 23

Filters

Filters are usually part of the issue template and determine how the results from Fortify Static Code Analyzer are shown. For example, you can have a filter to put SQL Injection issues found into a separate folder called “SQL Injections”, or you might have a filter so that issues with a confidence below a certain threshold are hidden. Along with filters, filter sets enable you to have a selection of filters used at any one time. This enables you to more easily customize your view and allows you to define a different view for developers, auditors, and managers so that they can more easily see the most important or relevant information for them.

Each FPR has an issue template associated with it, and in Micro Focus Fortify Software Security Center these are specified on an application version basis. For further information about issue templates and customizing them, see the *Micro Focus Fortify Audit Workbench User Guide*.

Filter Files

Filter files are flat files that you can specify along with a scan using `-filter` option. Use a filter file to blacklist specified categories, instance IDs, and rule IDs. If you determine that a certain category of issues or rules are not relevant for a particular scan, you can stop Fortify Static Code Analyzer from flagging these types of issues and adding them to FPR. You can use a filter file to reduce both the scan time and the size of the results file.

For example, if you are scanning a simple program that just reads a specified file, you might not want to see path manipulation issues, since these are likely planned as part of the functionality. To do filter out path manipulation issues, a file that contains a single line:

```
Path Manipulation
```

Save this file as `filter.txt`. Use the `-filter` option for the scan as shown in the following example:

```
sourceanalyzer -b <build_id> -scan -f myResults.fpr -filter filter.txt
```

The `myResults.fpr` does not include any issues with the category Path Manipulation.

For more information about filter files, see the *Micro Focus Fortify Static Code Analyzer User Guide*.

Scan-Time Filters

An alternate way to filter at scan-time is to use filter sets to reduce the number of issues based on conditions you specify with filters in an issue template. A scan-time filter can dramatically reduce the size of an FPR.

To do this, use Audit Workbench to create a filter and a filter set and then run the Fortify Static Code Analyzer scan with the filter set. For more detailed instructions about how to create filters and filter sets in Audit Workbench, see *Micro Focus Fortify Audit Workbench User Guide*. The following example describes the basic steps for how to create and use a scan-time filter:

1. In this example, suppose you use the OWASP Top 10 2017 and you only want to see issues categorized within this standard. Create a filter in Audit Workbench such as:

   ```
   If [OWASP Top 10 2017] 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 2017 category with 'A' in the name, then it hides it. Because all OWASP Top 10 2017 categories start with 'A' (A1, A2, ..., A10), then any category without the letter 'A' is not in the OWASP Top 10 2017. The filter hides the issues from view in Audit Workbench, but they are still in the FPR.
2. In Audit Workbench, create a new filter set called `OWASP_Filter_Set` that contains the previous filter, and then export the issue template to a file called `IssueTemplate.xml`.
3. You can then specify this filter at scan-time with the following command:

   ```
   sourceanalyzer -b <build_id> -scan -f myScanTimeFilterResults.fpr
   -project-template IssueTemplate.xml -Dcom.fortify.sca.FilterSet=OWASP_
   Filter_set
   ```

In the previous example, the inclusion of the `-Dcom.fortify.sca.FilterSet` property tells Fortify Static Code Analyzer to use the `OWASP_Filter_Set` filter set from the issue template `IssueTemplate.xml`. Any filters that hide issues from view are removed and are not written to the FPR. Therefore, you can reduce the visible number of issues, make the scan very targeted, and reduce the size of the resulting FPR file.

Note: Although scan-time filters can reduce the size of the FPR, they do not usually reduce the scan time. Fortify Static Code Analyzer examines scan-time filters after it calculates the issues to determine whether or not to write them to the FPR file. The filters in a filter file determine the rule types that Fortify Static Code Analyzer loads.

Excluding Source Code from the FPR

You can reduce the scan time and the size of the FPR file by excluding the source code information from the FPR, especially with large source files or codebases. You generally do not get a scan time reduction for small source files.

There are two ways to prevent Fortify Static Code Analyzer from including source code in the FPR. You can set the property in the `<sca_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`	This disables source code inclusion in the FPR.
`com.fortify.sca.FVDLDisableSnippets=true` Command-line Option: `–fvdl-no-snippets`	This excludes code snippets from the FPR.

The following command-line example uses both options:

```
sourceanalyzer -b <build_id> -disable-source-bundling
-fvdl-no-snippets -scan -f mySourcelessResults.fpr
```

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" on the previous page.

There are a few other options and properties that you can use to select what is excluded from the FPR. You can set these properties in the Fortify Static Code Analyzer properties file: `<sca_install_dir>/Core/config/fortify-sca.properties` or specify them during the scan phase with `-D<property_name>=true`. Most of these options have an equivalent command-line option.

Property Name	Description
`com.fortify.sca.FPRDisableMetatable=true` Command-line Option: `-disable-metatable`	This disables the metatable inside the FPR. Audit Workbench uses the metatable to map information in Functions view.
`com.fortify.sca.FVDLDisableDescriptions=true` Command-line Option: `-fvdl-no-description`	This excludes descriptions from the FPR. If you do not use custom descriptions, descriptions listed in the Fortify Taxonomy (https://vulncat.fortify.com) are used.

Property Name	Description

`com.fortify.sca.FVDLDisableEngineData=true`
Command-line Option:
`-fvdl-no-enginedata`
	This excludes engine data from the FPR. This is useful if your FPR contains a large number of warnings when you open the file in Audit Workbench. The caveat of this option is that you need to merge the FPR with the current audit project locally before you upload it to Micro Focus Fortify Software Security Center. Because the FPR does not contain the Fortify Static Code Analyzer version, Fortify Software Security Center is unable to merge it on the server.

`com.fortify.sca.FVDLDisableProgramData=true`
Command-line Option:
`-fvdl-no-progdata`
	This excludes the program data section from the FPR. This removes the Taint Sources information from the Functions view in Audit Workbench. This property typically only has a minimal effect on the overall size of the FPR file.

Opening Large FPR Files

To reduce the time required to open a large FPR file, there are some properties that you can set in the `<sca_install_dir>/Core/config/fortify.properties` configuration file. For more information about these properties, see the *Micro Focus Fortify Static Code Analyzer Tools Properties Reference Guide*. The following table describes these properties.

Property Name	Description
`com.fortify.model.DisableProgramInfo=true`	This disables use of the code navigation features in Audit Workbench.

`com.fortify.model.IssueCutOffStartIndex=<num>` (inclusive)
`com.fortify.model.IssueCutOffEndIndex=<num>` (exclusive)
	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=0`
	`com.fortify.model.IssueCutOffEndIndex=101`
	Because the `IssueCutOffStartIndex` is `0` by default, you do not need to specify this property.

Property Name	Description
`com.fortify.model.IssueCutOffByCategoryStartIndex=`*`<num>`* (inclusive) `com.fortify.model.IssueCutOffByCategoryEndIndex=`*`<num>`* (exclusive)	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`	This minimizes the data loaded in the FPR. This also restricts usage of the Functions view and might prevent Audit Workbench from loading the source from the FPR.
`com.fortify.model.MaxEngineErrorCount=`*`<num>`*	This property specifies the number of Fortify Static Code Analyzer reported warnings that are loaded with the FPR. For projects with a large number of scan warnings, this can reduce both load time in Audit Workbench and the amount of memory required to open the FPR.
`com.fortify.model.ExecMemorySetting`	Specifies the JVM heap memory size for Audit Workbench to launch external utilities such as iidmigrator and fortifyupdate.

Chapter 5: Monitoring Long Running Scans

When you run Fortify Static Code Analyzer, large and complex scans can often take a long time to complete. During the scan it is not always clear what is happening. While Fortify recommends that you provide your debug logs to the Micro Focus Fortify Customer Support team, there are a couple of ways to see what Fortify Static Code Analyzer is doing and how it is performing in real-time.

This section contains the following topics:

Using the SCAState Utility 25
Using JMX Tools 25

Using the SCAState Utility

The SCAState command-line utility enables you to see up-to-date state analysis information during the analysis phase. The SCAState utility is located in the `<sca_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 Static Code Analyzer spends its time during the scan. For more information about how to use the SCAState utility, see the *Micro Focus Fortify Static Code Analyzer User Guide*.

Using JMX Tools

You can use tools to monitor Fortify Static Code Analyzer with JMX technology. These tools can provide a way to track Fortify Static Code Analyzer performance over time.

Note: These are third-party tools and Micro Focus does not provide or support them.

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"
```

For more information about these parameters, see the full Oracle documentation available at: http://docs.oracle.com/javase/8/docs/technotes/guides/management/jconsole.html.

After the JMX parameters are set, start a Fortify Static Code Analyzer scan. During the scan, start JConsole to monitor Fortify Static Code Analyzer locally or remotely with the following command:

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

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 as described in "Using JConsole" on the previous page.

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
```

For more information about Java VisualVM, see the full Oracle documentation available at: http://docs.oracle.com/javase/8/docs/technotes/guides/visualvm.

Chapter 6: Troubleshooting

Function too Complex to Analyze Message

During a Fortify Static Code Analyzer scan, an 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

The depth of analysis Fortify Static Code Analyzer performs sometimes depends on the available resources. Fortify Static Code Analyzer uses a complexity metric to tradeoff these resources against the number of vulnerabilities that it can find. Sometimes, this means giving up on a particular function when Fortify Static Code Analyzer 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 the function in the program was completely ignored. For example, the Dataflow Analyzer typically visits a function many times before completing the analysis, and might not run into this complexity limit in the previous visits. In this case, the results include anything learned from the previous visits.

You can control the "give up" point using Fortify Static Code Analyzer properties called limiters. Different analyzers have different limiters.

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

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 Static Code Analyzer.

To resolve the complexity identifier of `l`, you can adjust the following limiters in the Fortify Static Code Analyzer property file `<sca_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 Static Code Analyzer always tries to analyze objects at the highest precision possible.

If a given function exceeds the `MaxTaintDefForVar` limit at a given level of precision, the Dataflow Analyzer analyzes that function with a lower level of 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 Static Code Analyzer then proceeds with analysis at the lowest precision level until either it finishes or the complexity exceeds the `MaxTaintDefForVarAbort` limiter. In other words, Fortify Static Code Analyzer tries harder at the lowest precision level than at higher precision levels, to get at least some results from the function. If Fortify Static Code Analyzer reaches the `MaxTaintDefForVarAbort` limiter, it gives up on the function entirely and you get the "Function too complex" warning.

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 a very long time when analyzing very complex functions. Therefore, Fortify Static Code Analyzer 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 analyzing functions, you can adjust the following property values in the Fortify Static Code Analyzer property file `<sca_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 Static Code Analyzer.

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 see the "Function too complex" warning, you can set the limiters/time to an extremely high value, however it can cause unacceptable scan time.

Send Documentation Feedback

If you have comments about this document, you can contact the documentation team by email. If an email client is configured on this computer, click the link above and an email window opens with the following information in the subject line:

Feedback on Performance Guide (Fortify Static Code Analyzer 18.10)

Just add your feedback to the email and click send.

If no email client is available, copy the information above to a new message in a web mail client, and send your feedback to FortifyDocTeam@microfocus.com.

We appreciate your feedback!