HP Fortify Static Code Analyzer Performance Guide

Software Version: 4.21
HP AIX, Solaris, Windows, Mac, Linux operating systems, PN 1-112-2014-10-321-01

HP Fortify Static Code Analyzer Performance Guide

Document Release Date: October 2014
Software Release Date: October 2014

Legal Notices

Warranty

The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

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

Copyright Notice

Trademark Notices

Adobe® is a trademark of Adobe Systems Incorporated.

Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation.

UNIX® is a registered trademark of The Open Group.

Documentation Updates

The title page of this document contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.

To check for recent updates or to verify that you are using the most recent edition of a document, go to: **http://h20230.www2.hp.com/selfsolve/manuals**

This site requires that you register for an HP Passport and sign in. To register for an HP Passport ID, go to: **http://h20229.www2.hp.com/passport-registration.html**

Or click the **New users - please register** link on the HP Passport login page.

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

Support

Visit the HP Software Support Online web site at: **http://www.hp.com/go/hpsoftwaresupport**

This web site provides contact information and details about the products, services, and support that HP Software offers.

HP Software online support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the support web site to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Download software patches
- Manage support contracts
- Look up HP support contacts
- Review information about available services
- Enter into discussions with other software customers
- Research and register for software training

Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract. To register for an HP Passport ID, go to:

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

To find more information about access levels, go to:

http://h20230.www2.hp.com/new_access_levels.jsp

HP Software Solutions Now accesses the HPSW Solution and Integration Portal Web site. This site enables you to explore HP Product Solutions to meet your business needs, includes a full list of Integrations between HP Products, as well as a listing of ITIL Processes. The URL for this Web site is **http://h20230.www2.hp.com/sc/solutions/index.jsp**

About this PDF Version of Online Help

This document is a PDF version of the online help. This PDF file is provided so you can easily print multiple topics from the help information or read the online help in PDF format. Because this content was originally created to be viewed as online help in a web browser, some topics may not be formatted properly. Some interactive topics may not be present in this PDF version. Those topics can be successfully printed from within the online help.

Contents

Abstract

This document describes the issues involved when trying to select hardware to scan certain codebases, provides guidelines for making those decisions and also offers tips for optimizing memory usage and performance.

Question: How much hardware do I need to scan a particular codebase?

Source code varies wildly in the real world which 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 (i.e. JSP, JavaScript, HTML…)
- Number of vulnerabilities
- Type of vulnerabilities (i.e. what analyzer is used)
- Complexity of the codebase

Of the above stated factors, none is harder to measure than code complexity. In fact there is a great deal of research on this subject; some have suggested the problem is unsolvable in the general case. This problem, especially when combined with the other factors demonstrates why this issue is so complex and non deterministic.

Given that code complexity is so difficult to measure, we must rely upon general guidelines, and anecdotal evidence to help shape a set of “best guess” recommendations for hardware requirements. What we provide here are a set of guidelines that we have developed through our findings from scanning real world applications and distilling down to requirements derived from their performance needs. It is important to note there may be cases where scanning your codebase may require more than our guidelines would imply. In an effort to improve our guidelines we welcome your feedback and input on how your project requirements have mapped to our guidelines.

For SCA with HP Fortify versions 3.x:

Size (LOC)	<100k	100k to 500K	500K-1M	1M+
Java	32-bit machine 2GB RAM	32-bit machine 4GB RAM	64-bit machine 8GB RAM	64-bit machine 16GB RAM

For SCA with HP Fortify versions 3.x:, continued

Size (LOC)	<100k	100k to 500K	500K-1M	1M+
.NET	32-bit machine 2GB RAM 3	32-bit machine 2GB RAM	64-bit machine 8GB RAM	64-bit machine 16GB RAM
C/C++	C/C++ 32-bit machine 2GB RAM	64-bit machine 16GB RAM	64-bit machine 16GB RAM	64-bit machine 16GB RAM

Note: JavaScript will slow the analysis time significantly. If the total LOC in an application is composed of more than 20% JavaScript, please use the next highest recommendation.

For SCA with HP Fortify versions 4.x the table below uses recommendations based on the complexity of the application. As of v4.10 SCA is moving away from support for 32-bit systems. As such all recommendations are using 64-bit hardware:

Application Complexity	CPU Cores	RAM	Average scan time	Notes
Simple	2 cores	4GB	0.5 hours	A system that runs on a server or desktop in a stand-alone manner like a batch job or a command line utility.
Medium	4 cores	16GB	4 hours	A standalone system, which works with complex computer models like a tax calculation system or a scheduling system.
Complex	8 cores	64GB	2 days	A three tiered business system with transactional data processing like a financial system or a commercial website.
Very Complex	16 cores	256GB	4 days	A system that serves up content like an application server, database server, or content management system.

Note: At present SCA is unable to make effective use of more than 16 CPU cores. Please see "CPUs, Parallel Processing and Multi-Threading" for details on making use of multiple cores.

Tips on Improving Performance

This section contains different methods of tuning the Fortify tool to maximize its functionality.

While the Fortify system requirements are documented and mandatory, when it comes to complex and large programs Fortify needs more capable hardware. This includes:

1. **Disk I/O**: Fortify is I/O intensive so the faster the hard drive, the more savings on the I/O transaction. We recommend a 7,200 RPM drive, although it would be better if you can get a 10,000 RPM drive (such as the WD Raptor), or an SSD drive.
   Another option is using a RAM disk. In this case, you need to save the project on the RAM disk. In addition, you need to change the property name `com.fortify.sca.ProjectRoot` in `<SCA Install Directory>\Core\config\fortify-sca.properties` to point to a directory located in the RAM disk.

2. **Memory**: While the default setting is 600 MB, larger applications need more RAM. Please refer to the "Memory Tuning" section for more details on how to determine the increase in the amount of memory.

3. **CPU**: we recommend a 2.1 GHz processor, although it would be better if you can get 3.2 GHz or faster.

On the software side, Fortify can take a long time to process complex projects. The time can be spent in different stages: (1) Translation, (2) Scan, or (3) Audit/Upload. In this section, we will offer different tips on how to improve the performance so that you can complete a scan or manage the processing time in different stages when it is taking too long. The table below is a summary of the stages and the corresponding options that would be most effective. The options are further explained in detail.

Stage	Option(s)	Description	Reference in document
Translation	`-export-build-session`	Translating/scanning on different machines	*"Mobile Build Sessions"*
Translation	`com.fortify.sca.IncrementFileMaxSizeMB`	Pare down the incremental session file	*"Incremental Session File" on page 16*
Scan	`-Xmx<size>M` `-Xmx<size>G`	Allocation of more RAM	*"Memory Tuning" on page 11*
Scan	`-Xss<size>M` `-Xss<size>G`	Allocation of stack size	*"Memory Tuning" on page 11*

Stage	Option(s)	Description	Reference in document
Scan	`-j <processes>`	Number of processes when utilizing multi processing	*"CPUs, Parallel Processing and Multi-Threading" on page 14*
Scan	`com.fortify.sca.RmiWorkerMaxHeap`	RAM allocation for individual processes when utilizing multi processing	*"CPUs, Parallel Processing and Multi-Threading" on page 14*
Scan	`com.fortify.sca.ThreadCount`	Hardcoded number of threads	*"CPUs, Parallel Processing and Multi-Threading" on page 14*
Scan	`com.fortify.sca.DisableSwapTaintProfile=True`	SCA will Keep taint information in the memory	*"Keeping Tainted Information in Memory" on page 17*
Scan	`-bin`	Scan the files related to the given binary (for C/C++)	*"Breaking Down Codebases" on page 18*
Scan	`-filter <filename>`	Applying filter by means of filter file	*"Filter Files" on page 22*
Scan	`-quick`	Quick scan	*"Quick Scan" on page 18*
Scan	`-disable-source-bundling`	Do not include source files in FPR file	*"Creating FPRs without Source Code" on page 23*

Mobile Build Sessions

You can translate modules in different machines using the same Build ID. You can then scan everything together using the same Build ID on a different machine with better hardware.

This can be done manually or by using Mobile Build Sessions that allow a project to be translated on one machine and analyzed on another. The advantage of the mobile build session is to perform the translation on the original machine and move the build session to a machine with better specs to perform the scan. This way the developers can run translations on their own machines and only one powerful machine is needed to run large scans.

It's highly recommended to move build sessions around using the export/import function. The `-export-build-session` and `-import-build-session` commands store/load the build session to/from a file specified on the command line.

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

Below is an example of the step by step commands needed for this process. Let's call "Machine T" the computer where we perform the translation and "Machine S" the computer where we apply the scan. Here are the commands that need to be run on each machine:

(i) **"Machine T":** Translation

```
sourceanalyzer -b <Build ID> <Translation Files>
```

(ii) **"Machine T":** We then package and export the mobile build into a file called *build-session.mbs*

```
sourceanalyzer -b <Build ID> -export-build-session build-session.mbs
```

(iii) Transfer `build-session.mbs` from "Machine T" to "Machine S"

(iv) **"Machine S":** Unpack the translation of the build folder into SCA's Project Root directory on the scan machine

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

(v) **"Machine S":** Finally perform a scan using the same Build ID we performed the translation with:

```
sourceanalyzer -b Build ID -scan -f myResults.fpr
```

Note: that there is no support for merging multiple mobile build sessions into a single build session; each exported build session must use a unique Build ID and be imported under that unique Build ID. However, once all of the Build IDs are present in the same sourceanalyzer installation, they can be scanned as part of the same scan using multiple Build IDs with the b option, just as if they were all translated on the same machine as the scan.

For example, assuming all Build IDs were created locally, or imported to the local machine using mobile builds, a command similar to the line below can be used:

```
sourceanalyzer -b BuildID_1 -b BuildID_2 -b BuildID_3 -scan -f myResults.fpr
```

While the resulting FPR (myResults.fpr) will cover the same files as if we translated all of the files into one Build ID from the beginning, it's worth noting that there are rare instances where dataflow between files may be lost if they are not translated together.

Mobile Build Compatibility

SCA's internal version numbering differs to the widely used HP Fortify version numbering. It instead follows the pattern: major.minor+patch.buildnumber. For example HP Fortify v4.10 shipped with SCA v6.10.0120. For a full mapping of SCA version numbers please see the HP Fortify System Requirements.

It's important to note the SCA version numbers on both the Translate and Scan machines when using Mobile Build Sessions. The following table shows the compatibility between the MBS stages:

SCA version creating MBS	SCA version performing Scan
before 6.00 (HP Fortify 4.00)	Has to match exactly including buildnumber
6.00 (HP Fortify 4.00)	6.00.x
6.01 (HP Fortify 4.01)	6.01.x
6.02 (HP Fortify 4.02)	6.02.x
6.1x (HP Fortify 4.1x)	Can be any 6.1x.x
6.2x (HP Fortify 4.2x)	Can be any 6.2x.x

Memory Tuning

By its very nature static analysis can be a very resource intensive process. As discussed in the "Abstract", the amount of physical RAM required for a scan will depend on the complexity of the code itself. As this will be an unknown until the first attempt to scan an application, it's possible that you will encounter OutOfMemory errors during the analysis.

OutOfMemory errors in SCA can be classified as follows:

- Java Heap Exhaustion
- Java Permanent Generation Exhaustion
- Native Heap Exhaustion
- Stack Overflow

Java Heap Exhaustion

Java heap exhaustion is the most common memory problem during SCA scans and is the result of allocating too little heap space to the Java virtual machine used by SCA for the project being scanned.

It can be identified by the following symptoms:

Symptoms

One or more of these messages will appear in the SCA 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

Resolving a Java heap exhaustion problem involves allocating more heap space to SCA's Java virtual machine when starting the scan. By default, SCA runs with a max heap value of 600 MB. To increase this value, use the `-Xmx` command line argument when running the SCA scan. For example, `-Xmx1300M` will make 1300MB available to SCA and `-Xmx1G` will make 1GB available.

Before adjusting this parameter, determine the maximum allowable value for Java heap space. The maximum value depends on two factors:

- Available physical memory
- Virtual address space limitations

Each of these may limit the amount of Java heap that can be allocated to SCA. The lower of the two limiting values should be used as the upper bound for a -Xmx argument.

Physical Memory

It’s recommended that the value of -Xmx should either not exceed 90% of the total physical memory or not exceed the total physical memory minus 1.5GB to allow for the operating system. If the system is to be dedicated to running SCA, you need not change it from here, however if the system resources will be shared with other memory-intensive processes, an allowance should also be subtracted for those other processes.

Note that other processes that are resident but not active while SCA is running can be swapped to disk by the operating system and do not need to be accounted for. Allocating more physical memory to SCA than is available in the environment may cause “thrashing” which will likely slow the scan down along with everything else on the system.

Virtual Address Space

Prior to v4.10, SCA ran as a 32-bit process by default. All 32-bit processes are subject to virtual address space limitations, the specifics of which depend on the underlying system. The size of the Java heap is constrained by the amount of contiguous virtual address space that can be reserved. On modern Linux systems, this limit is usually near 3GB. On Windows systems, address space fragmentation due to the way DLLs are loaded means the limit is typically between 1200MB and 1600MB. This value will change between systems due to different DLLs being loaded into the Java process (virus scanning software is one example).

On 64-bit capable hardware, SCA can be run in 64-bit mode. In 64-bit mode, virtual address space limitations are not usually a factor, and Java heap space is therefore limited only by available physical

memory. To activate 64-bit mode you need to pass the -64 option to SCA at translate- and scan-times. This option is no longer required in v4.10 as SCA runs as a 64-bit process by default.

Java Permanent Generation Exhaustion

Java maintains a separate memory region from the main heap which is called the Permanent Generation. In some rare cases, this memory region may get filled up during a scan, causing an OutOfMemoryError. Permanent generation exhaustion can be identified by the following error message:

Symptoms

This message will typically appear in an SCA log file, but may also appear in the command line output:

```
java.lang.OutOfMemoryError: PermGen space
```

Resolution

Permanent generation exhaustion is resolved by increasing the maximum size of the permanent generation. The permanent generation size can be tuned by passing to -XX:MaxPermSize argument to SCA. For example `-XX:MaxPermSize=128M`.

The default maximum value for the permanent generation if unspecified is 64 MB. Note that the permanent generation is allocated as a separate memory region from the Java heap, so increasing the permanent generation will increase the overall memory requirements for the process. See the discussion of virtual address space and physical memory limitations in the previous section for determining overall limits.

Native Heap Exhaustion

Native heap exhaustion is a very 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 (either virtual address space or physical memory) for its native operations (such as garbage collection) that it eventually encounters a fatal memory allocation failure that immediately terminates the process.

Symptoms

Native heap exhaustion can be identified by abnormal termination of the SCA process with 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 ...
```

Since this is a fatal Java virtual machine error, it will usually be accompanied by an error log created in the working directory, named “hs_err_pidNNN.log”.

Resolution

The resolution to this type of problem is slightly counter-intuitive. Since 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 and Java permanent generation). Reducing either of these values should reduce the crowding problem and allow the scan to be completed successfully.

Stack Overflow

Each thread in a Java application has its own stack. The stack is used to hold return addresses, function/method call arguments, etc. So if a thread tends to process large structures via recursive algorithms, it may need a large stack for all those return addresses and such. With the Sun JVM, you can set that size via -Xss parameter.

Symptoms

This message will typically appear in an SCA log file, but may also appear in the command line output:

```
java.lang.StackOverflowError
```

Resolution

The default stack size is only 1MB. This can be increased by passing the -Xss to your sourceanalyzer command. For example, `-Xss8M` will increase the stack to 8MB and `-Xss16M` to 16MB.

CPUs, Parallel Processing and Multi-Threading

Prior to HP Fortify v4.0 SCA operates as a single threaded process and is confined to one core. However v4.x introduces major changes to the SCA scan phase. Multi-threaded execution is now implemented during: pre-analysis, where we construct the data structures used by the analyzers; post-analysis, where we conduct whole-program analysis and generate the final issues; and FPR generation, where we bundle up the source and write the issues to the FPR file itself. Parallel processing can now be triggered during the main analysis stage, i.e. during the running of the SCA analyzers (dataflow, buffer, control flow etc.) Parallel processing allows you to reduce scan times by harnessing multiple cores, memory, and processing power in your machine.

Modes of execution in v4.x

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

Default Mode

We recommend that scans which take less than 2 hours are run in default mode. This has multi-threading enabled but parallel processing disabled. The memory requirements are as follows:

	Cores	Memory
Minimum	1	4 GB
Recommended	8	32 GB

Parallel Mode

We recommend that scans which take longer than 2 hours are run in parallel mode. This is designed for workstation or server-class hardware and has both multi-threading and parallel processing enabled. The memory requirements are as follows:

	Cores	Memory
Minimum	4	16 GB
Recommended	8	64 GB

Configuring Multi-Threading

By default SCA will use all available threads on the scanning machine. However if you need to lower the number of threads used for any reason, such as a resource constraint, this can be done by setting the `com.fortify.sca.ThreadCount` property in the following properties file: `<SCA Install Directory>\Core\config\fortify-sca.properties`.

Setting this property to 1 will force SCA to run as a single threaded process. While setting it to a higher value than the number of available cores on the machine is not recommended.

Running in Parallel Analysis Mode

Parallelization in SCA is implemented using a master-worker design. The analyzer execution running on the primary JVM is replaced with the master coordinator. This manages the program model and inter-process communication, while also spawning worker JVMs which execute the analysis. Each worker executes as a single-threaded process on a separate core.

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

```
-j <# worker processes>
```

Alternatively you can set this using the `com.fortify.sca.RmiWorkers` property in the SCA properties file: `<SCA Install Directory>\Core\config\fortify-sca.properties`.

The ideal number of worker processes is n 2, where n represents the number of processors in your machine. For example, if your machine has 8 processors, the ideal number of worker processes would be 6. Worker count properties are independent of thread count properties, so feel free to mix and match.

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

The heap size of the master is specified by passing the -Xmx value to SCA in the traditional manner - see the Memory Tuning section of this doc for further details. Our recommendation is to set this to 2 times the worker heap size.

By default the worker heap size will inherit the value of -Xmx from the master. However it's possible to alter the worker heap size by setting the com.fortify.sca.RmiWorkerMaxHeap property in the SCA properties file: `<SCA Install Directory>\Core\config\fortify-sca.properties`. For example com.fortify.sca.RmiWorkerMaxHeap=1G will assign each worker 1GB. Our recommendation is to assign each worker the same amount of heap required to run a successful scan in version 3.x.

Bear in mind you may need to balance the -Xmx and -j options to insure you don't allocate more memory than is physically available. To figure out the maximum number of workers for your installation you can use the following formula:

T = master memory + (physical memory per Java process×number of processes)

Where T does not exceed 90% of the total physical memory or the total physical memory minus 1.5GB.

Strategies for Best Use

It will likely be necessary to experiment with the parallel analysis and multi-threading settings in order to achieve a healthy balance between the available memory and scan speed. Our recommendations for getting the best use from these new features are as follows:

Small to medium projects (< 500K LOC)

- Scan in default mode
- Allow SCA to spin up threads on all cores
- Set SCA Java heap size to all of physical memory minus about 1.5GB for the OS

Medium to large projects (> 500K LOC)

- Scan in parallel mode
- Allow multi-threaded phases to use all cores
- Use 5 to 10 workers for parallel analysis
- Set master heap size as 2x worker heap size
- Use care to ensure that you do not over-commit physical memory

Incremental Session File

The *Session File* contains information about the session currently in use. In most cases a Build ID will be directly associated with a session, and the Session File keeps information about this, such as some translation settings, information about some of the files being translated and mostly metadata that's required for SCA to scan properly.

Occasionally you may experience poor performance due to SCA swapping information on and off the disk with limited space. This can be resolved by increasing the maximum size of the session file. To do this you will need to specify the property `com.fortify.sca.IncrementFileMaxSizeMB` in the property file `<SCA Install Directory>\Core\config\fortify-sca.properties` along with a value. For example:

```
com.fortify.sca.IncrementFileMaxSizeMB=1024
```

The units are in megabytes, so 1024MB is 1GB and should work in general. If the performance improves but the translation still fails, you can increase the number even more. Please note that this mainly affects users with SCA version 2.6.5 or earlier, as the functionality has significantly improved since then.

Keeping Tainted Information in Memory

To save memory, SCA saves the tainted information to disk and swaps the information when needed. You can set a flag to keep tainted information in memory by adding the following setting into `<SCA Install Directory>\Core\config\fortify-sca.properties`:

```
com.fortify.sca.DisableSwapTaintProfiles=true
```

Or by passing the `-Dcom.fortify.sca.DisableSwapTaintProfiles=true` parameter along with the `-scan` option. This will cause SCA to keep all the data in memory. In one case, the scan was taking 33 hours to complete, but by setting this property, the scan time was reduced to 13 hours.

This comes at a cost of higher memory usage. If SCA has enough memory available then this option will improve scan time. Otherwise, it may cause the analysis to run out of memory and produce no results.

Scan Quality vs. Performance

Breaking Down Codebases

It will be 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 such modules separately. The caveat to this is that dataflow may be lost if those few interactions do occur.

For C/C++ you might be able to reduce the scan time by using the -bin option in conjunction with –scan. You need to pass the binary file to it (such as "`-bin <filename>.exe -scan`" or "`-bin <filename>.dll -scan`"), and it will find the related files associated with that binary and scan them. This is particularly useful when you have several binaries in the Make file.

(i) **–bin**: 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.

(ii) **–show–binaries**: displays the independent binaries.

(iii) **–show-build-tree**: displays all files used to create binary and all files used to create those files in a tree layout.

Limiters

The depth of the analysis SCA performs sometimes depends on the available resources. SCA uses a complexity metric to trade off these resources against the number of vulnerabilities that can be found. Sometimes, this means giving up on a particular function when it doesn't look like SCA has enough resources available.

SCA allows the user to control the "cut off" point via some SCA properties called limiters. Different analyzers have different limiters; a predefined set of these limiters can be run using a Quick Scan. A full set of Limiters can be found in *"Appendix B: Performance Tuning Properties" on page 33*.

Quick Scan

Quick Scan Mode provides a way to quickly scan your projects for major defects. When using Quick Scan Mode, you should be aware that although the scan is significantly quicker, it does not provide a robust result set. To turn on quick scan, use `-quick` option with `-scan`.

When Quick Scan Mode is enabled, SCA applies the properties from the `<SCA Install Directory>\Core\config\fortify-sca-quickscan.properties` file, in addition to the standard `<SCA install directory>\Core\config\fortify-sca.properties` file. By default, this scan searches for high-confidence, high-severity issues. You can alter the limiters used by SCA by editing the `fortify-sca-quickscan.properties` file. If this file is empty, then the quick scan will be identical

to the full scan. In general, modifying `fortify-sca.properties` will also affect quick scan behavior. Fortify recommends doing performance tuning on quick scan, and leaving the full scan in the default settings to produce a highly accurate scan.

Quick Scan applies the "*Critical Exposure*" filter set to the scan; this limits the results to only the high-confidence and high-severity issues. For a full list of the default quick scan properties please see *"Appendix B: Performance Tuning Properties" on page 33*.

Using Quick Scan & Full Scan

- **Run periodic full scans:** when choosing to use quick scans, a periodic full scan is important as it may find issues not detected by the Quick Scan. A full scan should be run at least once per software iteration. If possible, a full scan should be run periodically when it will not interrupt workflow, such as over weekends.
- **Compare Quick Scan With a Full Scan:** To evaluate the accuracy impact of Quick Scan, perform a Quick Scan and a full scan on the same code base, then load the Quick Scan and merge it into the full scan. Select "*Group By: New*" in the pane on the left showing Critical, High, Medium and Low issues to produce a list of issues found in the full scan but not found in the Quick Scan.
- **Quick Scans and SSC Server:** To avoid overwriting the results of a full scan, by default SSC Server does not accept FPR's scanned using Quick Scan. It is however possible to get around this. Simply open the Project Version you wish to upload to on the SSC GUI. Navigate to General-> Analysis Processing Rules and uncheck the box labeled *Ignore SCA Scans performed in Quick Scan mode*.

False Positive Reduction

As users of the HP Fortify v3.x releases may have found, there were certain scenarios were an excessive number of false positives were reported. If you're using SCA from one of the v3.x releases we strongly recommend you upgrade to v4.x where this problem has been tackled.

If you continue to see large numbers of false positives in your scan results it's possible to fine tune SCA manually to counter this. Please see the appendices of the SCA User Guide for detailed information on how to go about this.

Limiting Analyzers and Languages

On occasion you may find that a significant amount of the scan time is spent either running one particular analyzer or analyzing a particular language. It's also possible that this particular analyzer or language is not of great interest to your security requirements. In these cases it's possible to limit the specific analyzers which run, and also the specific languages which are translated.

Disabling Analyzers

To disable specific analyzers you will need to pass the analyzers option to SCA at scan time along with a colon-delimited* white list of analyzers you wish to have enabled. The full list of analyzers is:

```
dataflow, semantic, controlflow, configuration, structural, content, buffer
```

As such, if you wished to run a scan using only the Dataflow, Controlflow and Buffer analyzers you would use the following command to scan:

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

It's also possible to configure this by setting `com.fortify.sca.DefaultAnalyzers` in the SCA configuration file `<SCA Install Directory>\Core\config\fortify-sca.properties`. For example the equivalent of the above would be set as:

```
com.fortify.sca.DefaultAnalyzers=dataflow:controlflow:buffer
```

*Please note, Windows requires using semi-colons to delimit the list. All other platforms take colons.

Disabling Languages

The method to disable specific languages is similar to the previous section. However you will need to pass the `-disable-language` option to SCA for the translation as opposed to the scan phase. This is followed by a colon* separated black list of languages you wish to have disabled. The full list of language options is:

```
actionscript, c, cpp, plsql, tsql, any_sql, jsp, csharp, vb, cfml, html, java,
javascript, php, asp, vbscript, vb6, cobol, python, abap, objc, llvm
```

As such, if you wished to run a scan excluding all SQL and HTML files you'd use the following command to perform the translation:

```
sourceanalyzer -b <Build ID> <Translation Files>-disable-language any_sql:html
```

It's also possible to configure this by setting `com.fortify.sca.DISabledLanguages` in the SCA configuration file `<SCA Install Directory>\Core\config\fortify-sca.properties`. For example the equivalent of the above would be set as:

```
com.fortify.sca.DISabledLanguages=any_sql:html
```

```
```

*Please note, Windows requires using semi-colons to delimit the list. All other platforms take colons.

Scanning Complex Functions

While performing a scan using SCA, the data flow analyzer might encounter a function for which it cannot complete the analysis and will report the following message:

```
Function <name> is too complex for <analyzer> analysis and will be skipped
(<identifier>)
```

<name> is the name of the function in the source code.

<analyzer> is the name of the analyzers within SCA (dataflow, control flow, null pointer, or buffer).

<identifier> is the type of complexity, it can be one of the following:

- "l" for too many distinct locations
- "m" for out of memory
- "s" for stack size too small
- "t" for taken too much time

The resolution to this issue is discussed in a knowledge base article, a copy of which can be found in *"Appendix A: Question regarding the SCA message: Function too complex to analyze" on page 30*.

Scan Size vs. Performance

Filters

Filters are usually part of the Project Template and determine how the results from SCA are shown. An example is that you may have a filter to put SQL Injection issues found into a separate folder called “SQL Injections”, or you may have a filter so that issues with a confidence below a certain threshold are hidden from the user. Along with Filters, there are Filter Sets, which allow you to have a selection of Filters showing 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 what’s the most important or relevant information for them.

Each FPR has a Project Template associated with it, and in SSC these are specified on a project version basis. For further information about Project Templates and customizing them, please refer to the Audit Workbench User Guide.

Filter Files

Filter files are flat files that can be specified along with a scan using `-filter <filter file name>`. This is then used as a blacklist of Category IDs, Instance IDs and/or Rule IDs. This means that if you feel a certain category of issues or rules are not relevant for this particular scan, you can stop them from flagging any issues that would go into the resulting FPR. This can be used in order to decrease the size of a results file along with the time it takes to scan a particular codebase.

For example if the scan is on a simple program that just reads a file specified, we may not want to see issues showing path manipulation issues, since these would likely be planned as part of the functionality. So, to do this we would create a new file just containing a single line saying:

```
Path Manipulation
```

Then after saving this as filter.txt, during the scan we would specify:

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

Now, in the newly created myResults.fpr there won’t be any issues shown with the category Path Manipulation.

Scan-Time Filters

An alternative way to filter at scan-time is by using filter sets to narrow down specifically what you want using the filters within a project template. These are called Scan-Time Filters and can dramatically reduce the size of an FPR.

To do this, you create a set of filters as usual in a new filter set. So, if for example you use the OWASP Top 10 2013, but you don't want to see any issues not categorized within this standard, then you could create a filter in Audit Workbench such as:

```
If [OWASP Top 10 2013] does not contain A Then hide issue
```

What this does is look through the issues and if it doesn't map to an OWASP Top 10 2013 category with 'A' in the name, then it hides it. And due to the fact that all OWASP Top 10 2013 categories start with 'A' (A1, A2, …, A10), it means that any without the letter 'A' must not be in the OWASP Top 10 2013. So this will hide them, but they will still be in the FPR.

If we set this within a new filter set called "*OWASP_Filter_Set*", and then export the Project Template to a file *ProjectTemplate.xml* we can specify this at scan-time as such:

```
sourceanalyzer -b <Build ID> -scan -f myScanTimeFilterResults.fpr -project-
template ProjectTemplate.xml
 -Dcom.fortify.sca.FilterSet=OWASP_Filter_set
```

This uses the Project Template file *ProjectTemplate.xml* to determine how the results should be represented, then `-Dcom.fortify.sca.FilterSet` is a property that tells SCA to use this filter set, so any filters that hide issues from a user's view normally are instead used to remove them so that they are not written to the FPR in the first place. Therefore this can be easily used in order to reduce the number of issues shown, making a scan very targeted and thus reducing the size of the resulting FPR file. However, it should be noted that although this can reduce the size of the FPR, it will not usually reduce the time it takes to scan, as scan-time filters are looked at after the issues have been calculated and to decide whether to write them to the FPR or not. Whereas the filters used from a filter file are used to determine the types of rules that should be loaded.

Creating FPRs without Source Code

Writing the source code into the FPR can be I/O intensive and can be part of the reason for large FPR files if the codebase is large. By not saving the source code information into the FPR, the scan can be faster, especially when the files are of large size. The saved time is not significant for smaller files or smaller codebases.

To disable source code being bundled into the FPR file:

Within the `<SCA Install Directory>\Core\config\fortify-sca.properties` file, set `com.fortify.sca.FPRDisableSourceBundling=true`. Alternatively this can be specified at scan-time with the option -disable-source-bundling.

In addition to this you can also disable code snippets from being written to the FPR by setting the property `com.fortify.sca.FVDLDisableSnippets=true`, which may be specified at scan time with the option `–fvdl-no-snippets`. This can also save the I/O time, depending on the size of the program and is useful if you have security requirements that state that no source should be included within these files.

The latter can also be specified at scan-time, an example of using both is:

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

Please note that references to the code are still in use within the FPR, so if testing this on a single machine, the source code location may have to be changed so that it isn't automatically picked up by Audit Workbench when trying to view the results. This doesn't place the source code back into the FPR however, and is just functionality within Audit Workbench to clearly see the full source for auditing purposes.

Opening Large FPRs

There are a few ways to open large FPRs. The first way to do this is by making the results file smaller in the first place so that it's not necessary to change other settings.

The quickest way to do this without affecting results would be to disable the source from the being presented within the FPR as shown in the section *"Creating FPRs without Source Code" on the previous page*.

Alternatively there are a few more switches and properties that can be used to fine tune what is left out of the FPR. These properties can be set in the SCA properties file: `<SCA Install Directory>\Core\config\fortify-sca.properties`, or during the scan with `-D<Property name>=true`, alternatively most of them also have associated switches shown in the table below:

Set in <SCA Install Directory>\Core\config\fortify-sca.properties	
`com.fortify.sca.FPRDisableMetatable=true`	
Command line option: `-disable-metatable`	This will disable the *metatable* within the FPR. This is used for mapping information to be able to find for example where a function is declared. This is used heavily by the Functions view within Audit Workbench.
`com.fortify.sca.FVDLDisableDescriptions=true`	
Command line option: `-fvdl-no-description`	This will disable the *Descriptions* that show for issues. If not using custom descriptions will likely be the same as the description shown in the HP Fortify Taxonomy page called *vulncat* (http://www.hpenterprisesecurity.com/vulncat/en/vulncat/index.html)
`com.fortify.sca.FVDLDisableEngineData=true`	
Command line option: `-fvdl-no-enginedata`	This will disable the *Analysis Information* within the FPR. This can be useful if your FPR contains such a large number of warnings that AWB is struggling to open the file. The caveat of this option is that you will need to be merge the FPR with the current Project File locally prior to uploading to SSC. As the FPR does not contain the SCA version SSC is unable to merge it on the server side.

Set in <SCA Install Directory>\Core\config\fortify-sca.properties	
com.fortify.sca.FVDLDisableProgramData	
Command line option: -fvdl-no-progdata	This will the disable the *ProgramData* section from within the FPR. This removes the Taint Sources information from the Functions tab within AWB. This property will likely only have a minimal effect on the overall size of the FPR file.

Audit Workbench (AWB)

Along with those, there are some specific properties that can be set in the general Fortify properties file: `<SCA Install Directory>\Core\config\fortify.properties`. These are shown in the following table:

Set in <SCA Install Directory>\Core\config\fortify.properties
com.fortify.DisableProgramInfo=true
This disables use of the code navigation features within AWB
com.fortify.model.IssueCutOffStartIndex=<number> (inclusive) com.fortify.model.IssueCutOffEndIndex=<number> (exclusive)

The *IssueCutOffStartIndex* property is inclusive and *IssueCutOffEndIndex* is exclusive so that you can specify a subset of issues you wish to see. E.g. To see the first 100 issues, you can specify:
`com.fortify.model.IssueCutOffStartIndex=0`
`com.fortify.model.IssueCutOffEndIndex=101`
However because the IssueCutOffStartIndex is 0 by default, this can be left out.

com.fortify.model.IssueCutOffByCategoryStartIndex=<number> (inclusive) com.fortify.model.IssueCutOffByCategoryEndIndex=<number> (exclusive)

These are similar to the above properties except these are specified for every category. E.g. If you wanted to see the first 5 issues for every category you would specify:
`com.fortify.model.IssueCutOffByCategoryEndIndex=6`
`com.fortify.RestrictIssueLoading=true`
This restricts the data that is held in memory, but may cause poor performance.

com.fortify.model.MinimalLoad=true
This restricts a lot of data from being loaded in the FPR so that only the bare minimum information is loaded. This will also restrict usage of the functions view and may prevent the source being loaded from within the FPR.
com.fortify.model.MaxEngineErrorCount=<number>

Set in <SCA Install Directory>\Core\config\fortify.properties
Available from v4.20. Limits the number of errors loaded with the FPR. For projects with a large number of scan warnings this can significantly reduce both load time in AWB and the amount of memory required to open the FPR.

Monitoring Long Running Scans

When running SCA large and complex scans can often take a significant amount of time to complete. During this time it's not always clear what is happening and if SCA is doing anything. In situations such as this, while we recommend you provide your debug logs to the HP Fortify Tech Support team, there are a couple of ways to see in real time what SCA is doing and how it is performing.

SCAState

The SCAState tool can be found in the `<SCA Install Directory>\bin` directory. This provides a simple means of checking what SCA is currently working on. It also provides a set of timers and counters showing where SCA has spent its time so far. You can run SCAState using the following:

```
SCAState [options] <SCA process ID>
```

For further details on the options available with SCAState please see the *SCA Utilities Guide*.

JMX

It's also possible to use a variety of tools to monitor SCA via JMX. These tools have a variety of advantages and disadvantages, however the GUI based tools specifically can offer a means to track SCA's performance over time as opposed to the snapshot given by SCA State.

Please bear in mind these are 3rd party tools and are not provided or supported by HP or HP Fortify.

HPROF & HAT

The most straight forward tools to use are the Heap/CPU Profiling Tool (HPROF) and the Heap Analysis Tool (HAT). These two tools go hand-in-hand.

HPROF is a simple command line tool for heap and CPU profiling. It is a JVM native agent library which is dynamically loaded through a command line option, at JVM startup, and becomes part of the JVM process.

By supplying HPROF options at startup, it's possible to request various types of heap and/or CPU profiling features from HPROF. The data generated can be in textual or binary format, and can be used to track down and isolate performance problems involving memory usage and inefficient code.

HPROF can be called when the scan is initiated with:

```
sourceanalyzer -b <Build ID> -scan -f
```

```
myResults.fpr -Xrunhprof:cpu=samples,interval=1,depth=10,format=b,
file=java.hprof.bin,heap=dump
```

For all the possible command line options please see the full Oracle documentation available at:

http://docs.oracle.com/javase/7/docs/technotes/samples/hprof.html

The binary file format output by HPROF can be used with tools such as HAT to browse the allocated objects in the heap. HAT parses a Java heap dump file and launches a webserver. By default this webserver listens on port 7000. To call HAT you would use a command such as the following:

```
jhat -J-Xmx4G ./java.hprof.bin
```

For further details on HAT please see the full Oracle documentation available at:

http://docs.oracle.com/javase/7/docs/technotes/tools/share/jhat.html

Using the `--openfile option`, the binary output from HPROF can also be viewed with *JConsole* or *Java VisualVM* which are discussed later in this guide.

HPROF does have a number of disadvantages however. The greatest of these being that it is not real time. This means the statistics are only reported once the process is done. It's also not interactive. As such it's necessary to use a text editor or HAT to view the results. Finally, it's not possible to use it in a remote fashion, it can only run on the box running the application.

JConsole

JConsole is an interactive, real-time monitoring tool which complies with the JMX specification. It uses the extensive instrumentation of the JVM to provide information about the performance and resource consumption of applications running on the Java platform.

In order to use JConsole it's necessary to first set some additional JVM parameters. This is done by setting 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"
```

More on these parameters can be found in the full Oracle documentation available at:

http://docs.oracle.com/javase/7/docs/technotes/guides/management/jconsole.html

Once the JMX parameters have been set, begin an SCA scan as usual. Once the scan is running you can launch JConsole to monitor SCA locally or remotely with:

```
jconsole <HostName>:9090
```

```
```

The major disadvantage of JConsole is that you cannot save the output.

Java VisualVM

Java VisualVM offers the same capabilities as JConsole. However it also offers some more advanced functionality. It provides more detailed information on the JVM and also allows you to save the monitor information to a so called “application snapshot file”. These files can be stored and opened afterwards with Java VisualVM.

As with JConsole, before you can use Java VisualVM, you will need to set the same JVM parameters as detailed in the previous section on "JConsole".

Once the parameters have been set, launch the scan as usual. You can then launch Java VisualVM to monitor the scan either locally or remotely with:

```
jvisualvm <HostName>:9090
```

For further details on Java VisualVM please see the full Oracle documentation available at:

http://docs.oracle.com/javase/7/docs/technotes/guides/visualvm/index.html

Appendix A: Question regarding the SCA message: Function too complex to analyze

This knowledge base article can also be found via following link:

https://na4.salesforce.com/articles/Question/Question-regarding-the-SCA-message-function-too-complex-to-analyze

Question

While performing a scan using SCA, the data flow analyzer might encounter a function for which it cannot complete the analysis and will report the following message:

```
Function <name> is too complex for <analyzer> analysis and will be skipped
(<identifier>)
```

<name> is the name of the function in the source code.

<analyzer> is the name of the analyzers within SCA (dataflow, control flow, null pointer, or buffer).

<identifier> is the type of complexity, it can be one of the following:

- "l" for too many distinct locations
- "m" for out of memory
- "s" for stack size too small
- "t" for taken too much time

Answer

The depth of analysis SCA performs sometimes depends on the available resources. SCA uses a complexity metric to tradeoff these resources against the number of vulnerabilities that can be found. Sometimes, this means giving up on a particular function when it doesn't look like SCA has enough resources available. This is normally when you will see the "Function too complex" messages appear.

When this message appears, it doesn't necessarily mean the function in the program has been completely ignored. For example, the dataflow analyzer will typically visit a function many times before analysis is complete, and may not run into this complexity limit in the early visits (since its model of other functions is less developed). In this case, anything learned from the early visits will be reflected in the results.

That said, we do allow the user to control the "give up" point via some SCA properties called limiters. Different analyzers have different limiters, see details below.

Dataflow Analyzers

There are currently 3 types of complexity for dataflow analyzers:

- l: too many distinct locations
- m: out of memory
- s: stack size too small

For 'm' and 's', you can increase the memory allocation or stack size for SCA by using -Xmx or -Xss respectively. By default, SCA uses 600M for -Xmx and 1M for -Xss.

'l' is a little more complicated. There are three limiters involved:

```
com.fortify.sca.limiters.MaxTaintDefForVar, default 1000
com.fortify.sca.limiters.MaxTaintDefForVarAbort, default 4000
com.fortify.sca.limiters.MaxFieldDepth, default 4
```

The **MaxTaintDefForVar** limiter is a dimensionless value expressing the complexity of a function, while **MaxTaintDefForVarAbort** is the upper bound for it. The **MaxFieldDepth** limiter is used to measure the precision when dataflow analyzer analyzes any give object. SCA would always try 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 will analyze that function with a lower level of precision (by reducing the **MaxFieldDepth** limiter). Reducing the precision reduces the complexity of the analysis. When the precision cannot be reduced any further, SCA will then proceed with analysis at the lowest precision level until either it finishes or the complexity exceeds the **MaxTaintDefForVarAbort** limiter. In other words, SCA will try harder at the lowest precision level than at higher precision levels, in order to get at least some results from the function. If SCA reaches the **MaxTaintDefForVarAbort** limiter, though, it will give up on the function entirely, thus the "Function too complex" warning.

Control Flow and Null Pointer Analyzers

There are currently 2 types of complexity for both control flow and null pointer analyzers:

- m: out of memory
- t: taken too much time

Due to the way dataflow analyzer handles function complexity, it will not take indefinite amount of time. Control flow and null pointer analyzers, however, can take a very long time when analyzing really complex functions. Therefore, SCA needs to have a way to abort the analysis when this happens, then you will see the "Function too complex" message with 't'.

To change these times, you can use the following parameters:

```
com.fortify.sca.CtrlflowMaxFunctionTime, default 10 minutes (600000
```

```
milliseconds)
com.fortify.sca.NullPtrMaxFunctionTime, default 5 minutes (300000 milliseconds)
```

For 'm', the solution is again to increase the memory allocation for SCA.

Note: Increasing these limiters or time settings will make analysis of complex functions take longer. It is hard to characterize the exact performance implications of a particular value for the limiters/time, since it depends on the specific function in question. If you never want see the "Function too complex" warning, you can set the limiter/time to an infeasible high value in fortify-sca.properties or on the command line, but this is likely to cause unacceptable performance degradation.

Appendix B: Performance Tuning Properties

Set -quick to use fortify-sca-quickscan.properties	
`com.fortify.sca.FilterSet`	
Default value is not set Quick Scan value: `Critical Exposure`	When set to targeted, this property runs rules only for the targeted filter set. Running only a subset of the defined rules allows the SCA scan to complete more quickly. This causes SCA to run only those rules that can cause issues identified in the named filter set, as defined by the default project template for your application.
`com.fortify.sca.FPRDisableSrcHtml`	
Default value: **False** Quick Scan value: **True**	When set to true, this property prevents the generation of marked -up source files. If you plan to upload FPRs that are generated as a result of a quick scan, you must set this property to false.
`com.fortify.sca.limiters.ConstraintPredicateSize`	
Default value: **50000** Quick Scan value: **10000**	Skips calculations defined as very complex in the buffer analyzer to improve scanning time
`com.fortify.sca.limiters.BufferConfidenceInconclusiveOnTimeout`	
Default value: **true** Quick Scan value: **false**	Skips calculations defined as very complex in the buffer analyzer to improve scanning time.
`com.fortify.sca.limiters.MaxChainDepth`	
Default value: **5** Quick Scan value: **4**	Controls the max call depth through which the dataflow analyzer tracks tainted data. Increasing this value increases the coverage of dataflow analysis, and results in longer analysis times. **Note:** In this case, call depth refers to the max call depth on a dataflow path between a taint source and sink, rather than call depth from the program entry point, such as `main()`.
`com.fortify.sca.limiters.MaxTaintDefForVar`	
Default value: **1000** Quick Scan value: **500**	This property sets the complexity limit for data flow precision backoff. Dataflow incrementally decreases precision of analysis for functions that exceed this complexity metric for a given precision level.

Set -quick to use fortify-sca-quickscan.properties	
`com.fortify.sca.limiters.MaxTaintDefForVarAbort`	
Default value: **4000** Quick Scan value: **1000**	This property sets a hard limit for function complexity. If complexity of a function exceeds this limit at the lowest precision level, the analyzer will not analyze that function.
`com.fortify.sca.DisableGlobals`	
Default value: **False** Quick Scan value: **False**	This property prevents the tracking of tainted data through global variables to allow faster scanning.
`com.fortify.sca.CtrlflowSkipJSPs`	
Default value: **False** Quick Scan value: **False**	This property skips control flow analysis of JSPs in your project.
`com.fortify.sca.NullPtrMaxFunctionTime`	
Default value: **300000** Quick Scan value: **30000**	This property sets a time limit, in milliseconds, for Null Pointer analysis for a single function. The default is five minutes. Setting it to a shorter limit decreases overall scanning time.
`com.fortify.sca.CtrlflowMaxFunctionTime`	
Default value: **600000** Quick Scan value: **30000**	This property sets a time limit, in milliseconds, for control flow analysis for a single function. The default is 10 minutes.
`com.fortify.sca.TrackPaths`	
Default value is not set Quick Scan value: **NoJSP**	This property disables path tracking for control flow analysis. Path tracking provides more detailed reporting for issues, but requires more scanning time. You can disable this for JSP only by setting it to `NoJSP`, or for all functions by setting it to None.

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 system, click the link above and an email window opens with the following information in the subject line:

Feedback on HP Fortify Static Code Analyzer Performance Guide (Fortify Static Code Analyzer Performance Guide 4.21)

Just add your feedback to the email and click send.

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

We appreciate your feedback!