Intel® Inspector 2021.4

Release Notes

21 September 2021

Customer Support

For technical support, including answers to questions not addressed in this product, visit the technical support forum, FAQs, and other support information at:

- Intel Inspector Forum
- General oneAPI Support
- Intel Inspector Home Page

Please remember to register your product at https://registrationcenter.intel.com/ by providing your email address. Registration entitles you to free technical support, product updates and upgrades for the duration of the support term. It also helps Intel recognize you as a valued customer in the support forum.

NOTE: If your distributor provides technical support for this product, please contact them for support rather than Intel.

Contents

1 Introduction

Intel® Inspector helps developers identify and resolve memory and threading correctness issues in their C, C++ and Fortran applications on Windows* and Linux*. Additionally, on Windows platforms, the tool allows the analysis of the unmanaged portion of mixed managed and unmanaged programs and identifies threading correctness issues in managed .NET C# applications.

Intel Inspector is a dynamic error checking tool for developing multithreaded applications on Windows or Linux operating systems. Intel Inspector maximizes code quality and reliability by quickly detecting memory, threading, and source code security errors during the development cycle. You can also use the Intel Inspector to visualize and manage Static Analysis results created by Intel® compilers in various suite products. Intel Inspector is an easy, comprehensive solution that delivers rapid results by isolating memory and multithreading errors.

Intel Inspector has a standalone graphical user interface (GUI) as well as a command line interface (CLI). In addition, on Microsoft Windows systems, the Intel Inspector integrates into the Microsoft Visual Studio* 2017 and later versions.

This document provides system requirements, issues and limitations, and legal information.

To learn more about this product, see:

- New features listed in the What's New section below.
- The help documents at:
  - Linux: https://software.intel.com/en-us/inspector-user-guide-linux
  - Windows: https://software.intel.com/en-us/inspector-user-guide-windows

2 What's New

Intel® Inspector 2021.4 Release

- Intel® Inspector has been updated to include more recent versions of 3rd party components, which include functional and security updates. Users should update to the latest version.
- **Improved C++ stack frames visualization**. This is an enhancement of stack frames reporting for C++ methods in CLI and GUI. With this change, C++ members will now be reported using complete signature. (example: "fetch()" => "std::atomic::fetch(int)")
- **Reduced number of false reports on libc and OpenCL libraries.**

Intel® Inspector 2021.3 Release

- Added correctness check for memcpy() function arguments on Windows. Inspector will report diagnostic if it detects overlapping of memcpy() arguments.

Intel® Inspector 2021.2 Release

- Locate and debug threading and memory errors in the GPU target Data Parallel C++ and OpenMP codes. (This feature is in technical preview mode)

Intel® Inspector 2021 Initial Release

- Preview: memory and threading errors analysis for DPC++ and OpenMP offloaded codes, executed on CPU target. To enable, set the following environment variables:

```
SYCL_BE=PI_OPENCL
SYCL_DEVICE_TYPE=CPU
CL_CONFIG_USE_VTUNE=True
CL_CONFIG_USE_VECTORIZER=false
```

3 System Requirements

Supported Architectures and Terminology

Intel® Inspector supports the following architectures:

- **IA-32 Architecture** refers to systems based on 32-bit processors generally compatible with the Intel Pentium® processors (for example, Intel® Pentium® 4 processor or Intel® Xeon® processor), or processors from other manufacturers supporting the same instruction set, running a 32-bit operating system.
- **Intel® 64 Architecture** refers to systems based on IA-32 architecture processors which have 64-bit architectural extensions, for example, Intel® Core™ architecture processors, running a 64-bit operating

system such as Microsoft Windows* 10 x64 or a Linux* "x86_64" variant. If the system is running a 32-bit operating system, then IA-32 architecture applies instead.

Hardware Requirements

- At least 4 GB of RAM
- 350 MB free disk space required for all product features and all architectures.
- A system based on an IA-32 or Intel® 64 architecture processor supporting the Intel® Streaming SIMD Extensions 2 (Intel® SSE2) instructions (Intel® Pentium 4 processor or later, or Intel® Xeon® Scalable Processors)
    - Since the Intel Inspector requires specific knowledge of assembly-level instructions, its analysis may not operate correctly if code contains non-Intel instructions. In this case, run the analysis with a target executable that contains only Intel® instructions. After you finish using the Intel Inspector, you can use the assembler or optimizing compiler options that provide the non-Intel instructions.
    - For the best experience, a multi-core or multi-processor system is recommended

Software Requirements

- Supported operating systems (embedded editions are not supported):
    - Microsoft Windows* 10 operating system
    - Microsoft Windows Server* 2016 and 2019
    - Red Hat* Enterprise Linux* 7 and 8
    - CentOS* versions equivalent to Red Hat Enterprise Linux versions listed above
    - Fedora* 33 and 34 (`pangox-compat` package should be installed)
    - SUSE* Linux* Enterprise Server 12 and 15
    - Debian* 9, 10
    - Ubuntu* 18.04, 20.04 and 21.04
- Supported Microsoft Visual Studio* versions:
    - Microsoft Visual Studio* 2017
    - Microsoft Visual Studio* 2019
- Recommended compilers:
    - Intel® C/C++ Compiler 12.0 or higher
    - GNU* C/C++ Compiler 3.4.6 and higher (Linux)
    - Intel® Fortran Compiler 12.0 or higher (Linux)
    - Intel® Visual Fortran Compiler 12.0 or higher (Windows)
- Application coding requirements
    - Supported programming languages:
        - Fortran
        - C
        - C++
    - Supported threading methodologies:
        - Intel® Threading Building Blocks (Intel® TBB)
        - Microsoft* PPL* (via Intel® TBB)
        - Intel® Cilk™ Plus
        - OpenMP*
        - Win32* Threads (Windows)

        - POSIX* Threads (Linux)
        - Intel® C++ Compiler parallel language extensions
- Managed application coding requirements
    - Supported programming languages:
        - C# (full support for .NET* 2.0-3.5, partial support for .NET 4.0)

          NOTE: Intel Inspector has limited support for .NET 4.0:
          - The Microsoft* Task Parallel Library classes and algorithms in the `System.Threading.Tasks` namespace are not supported.
          - The `Barrier` synchronization class in the `System.Threading` namespace (new in .NET 4.0) is not supported.
- To view PDF documents, use a PDF reader, such as Adobe Reader*.

NOTE: You can use Intel Inspector to analyze applications in debug and release modes. To learn more about options necessary to produce the most accurate, complete results, refer to the following related resources:

- Memory error analysis: https://software.intel.com/en-us/articles/compiler-settings-for-memory-error-analysis-in-intel-inspector-xe/
- Threading error analysis: https://software.intel.com/en-us/articles/compiler-settings-for-threading-error-analysis-in-intel-inspector-xe/

4 Where to Find the Release

If you have a qualifying license for this product (i.e. an unexpired free license or a paid license with active support at the time of the build date for this product release), you can download the installer by logging in to https://registrationcenter.intel.com/ and selecting the appropriate product.

You may need to create an account and/or register your product. For additional information please see:

- https://software.intel.com/content/www/us/en/develop/articles/oneapi-commercial-faq.html

This product can be also downloaded as part of one of the Intel® oneAPI Toolkits with forum support.

5 Installation Notes

For instructions on installing, updating or removing Intel® Inspector, see the appropriate installation guide for your operating system.

- For Linux*
- For Windows*

Default Installation Folders

The default top-level installation folders for this product are:

- Windows: `C:\Program Files (x86)\Intel\oneAPI\Inspector\2021.2.0`

  - If you are installing on a system with a non-English language version of Windows* OS, the name of the `Program Files` folder may be different. On Intel® 64 systems, the directory name is `Program Files (x86)` or the equivalent.
- Linux: `/opt/intel/oneapi/inspector/2021.2.0`

In both cases, there is also a generic soft link to the most recently installed version.

Known Installation and Configuration Issues

All of the following issues are Windows specific, as they deal with Microsoft Visual Studio.

- In very rare situations, the installer may hang during the phase *Configuring integration with Microsoft Visual Studio* 20XX software…* phase. If the installer hangs for more than 20 minutes, locate process `devenv.exe` and shut down the process. Make sure that you do not have any running instances of Microsoft Visual Studio software. Reboot the machine and then install the product again.
- By default, Microsoft Visual Studio sets the **Launch in Browser** option to display documentation for integrated products. To view the Intel Inspector local help documentation: Choose **Help > Intel Inspector 2020 > Intel Inspector 2020 Help** or use context-sensitive help. If you still encounter problems viewing the help, choose **Help > Set Help Preferences > Launch in Help Viewer**.
- If you have problems viewing Microsoft Visual Studio help pages, check the settings for Internet Explorer* in **Tools > Internet Options > Security**. To allow correct display of help, in the **Internet zone** enable **MIME Sniffing** and **Active scripting**. Alternatively, from **Tools > Internet Options > Security** add "about:internet" to the list of trusted sites.

Setting Up the Command Line

Before using Intel Inspector on the command line, you will need to set up the environment:

- Windows: `<installation_dir>\inspxe-vars.bat`
- Linux: `source <installation_dir>/inspxe-vars.sh`

6 Known Issues

This section is quite long. To skip to operating system specific limitations, use these links:

- Windows* Specific Limitations
- Linux* Specific Limitations

Common Limitations

INTEL® INSPECTOR MAY NOT DETECT ALL ERRORS

Intel does not guarantee that the Intel Inspector will detect or report every memory and threading error in an application.

- Not all logic errors are detectable.
- Heuristics used to eliminate false positives may hide real issues.
- Highly correlated events are grouped into a single problem.

INTEL INSPECTOR MAY INCORRECTLY REPORT DATA RACE PROBLEMS ON SIMPLE OPERATIONS

Certain x86 processors guarantee load or store operations will always be carried out atomically if it confirms well defined restrictions (like memory alignment, operand size etc.). Modern compilers could utilize that fact and generate code without explicit locks. That misleads Intel Inspector because it cannot recognize if this operation is intended to be non-atomic or it is so due to mistake. To mitigate that it is recommended to build application without optimization and enable symbolic information for all modules, that will allow Intel Inspector to intercept standard APIs for atomic operations (e.g. `std::atomic`, `std::atomic_flag`) and verify correctness of memory ordering semantic. In case of non-standard API or compiler intrinsic usage, Intel Inspector might produce false reports on such operations. It might be filtered out by enabling “Filter guaranteed atomics” analysis option, but it might hide true errors as well that appear to be correct for analysis run.

INTEL INSPECTOR MAY DISPLAY MODULE NAMES AND RELATIVE VIRTUAL ADDRESSES IF IT CANNOT LOCATE SYMBOLS

If no symbols are found for a module in which a problem is detected, the Intel Inspector displays the call stack and observation source code of the first location where it can find symbols. If it cannot find any location in the call stack with symbols, it displays the module name and relative virtual address (RVA) for the location.

APPLICATION CRASHES MAY CRASH OR HANG INTEL INSPECTOR

Applications that crash when run outside the Intel Inspector may crash or hang the Intel Inspector runtime analysis engine. For example, a corrupt return address on an application call stack crashes the runtime analysis engine. If a crash occurs, problems detected prior to that time can be viewed, but memory leaks will not be reported. Reviewing the problems that were reported may help, as it is likely that one of them caused the crash.

INTEL INSPECTOR CANNOT START AN ANALYSIS FROM THE GUI IF PREVENTED FROM OPENING A SOCKET

Intel Inspector uses a socket to communicate between the graphical user interface and the runtime analysis engine. Preventing an application from opening a socket prevents an analysis of the application from being started by the graphical user interface. The command-line interface can be used to run an analysis in this case and the results can subsequently be viewed using the graphical interface.

INTEL INSPECTOR MAY INCORRECTLY REPORT CALL STACKS FOLLOWING AN INTERRUPTION OF NORMAL CALL FLOW

Intel Inspector may report an incorrect call stack following an interruption of normal call flow, such as when an exception is thrown and caught. While the Intel Inspector recognizes and attempts to correct result data when this situation occurs, it is possible for a threading or memory problem to be reported before the call stack is fully corrected.

USING TOO MANY OPENMP* THREADS MAY RESULT IN “INSUFFICIENT MEMORY” ERRORS

If the Intel Inspector reports insufficient memory errors while analyzing OpenMP* applications, try setting `OMP_NUM_THREADS` to limit the number of OpenMP threads. In most cases, `OMP_NUM_THREADS=2` is sufficient.

LARGE APPLICATIONS MAY RESULT IN “INSUFFICIENT MEMORY” ERRORS

If the Intel Inspector reports insufficient memory errors during analysis, try analyzing your application in sections by running several analyses and excluding a different set of modules for each run. Note that Intel Inspector does not detect or report issues in excluded modules.

INTEL INSPECTOR ANALYZES ONLY THE FIRST INSTANCE OF A CHILD APPLICATION

If a child application is selected for analysis, the Intel Inspector analyzes only the first instance of that application, even if multiple instances of that application occur.

INTEL INSPECTOR FILTERS DO NOT PERSIST ON RELOADING RESULTS

When taking advantage of the Intel Inspector custom filter by source, be aware that this filter does not persist when reloading results.

MEMORY ERROR ANALYSIS: ON-DEMAND LEAK DETECTION DOES NOT REPORT MEMORY LEAKS UNTIL THE END OF THE BLOCK

On-demand leak detection will not report memory leaks until the end of the basic block in which they occur, as there is still an internal reference in the program to that memory. If the baseline for leak detection is not reset, these leaks will be reported at the next request.

MEMORY ERROR ANALYSIS: INTEL INSPECTOR MAY REPORT FALSE POSITIVES ON APPLICATIONS WITH CUSTOM MEMORY ALLOCATORS

Intel Inspector may report false positives when the analyzed application uses custom memory allocators. Use of `_itt_notify` to annotate your source code can reduce these false positives.

MEMORY ERROR ANALYSIS: INTEL INSPECTOR MAY BEHAVE UNPREDICTABLY IF C RUNTIME ALLOCATOR SEMANTICS ARE CHANGED

If the semantics of standard C runtime allocators are changed, the behavior of the Intel Inspector is unknown and could lead to abnormal analysis termination. For instance, if the application is using non-standard versions of these allocators where the memory returned by the allocator is initialized when it would normally be uninitialized.

THREADING ERROR ANALYSIS: INTER-PROCESS DATA RACES AND DEADLOCKS ARE NOT DETECTED.

Intel Inspector does not detect inter-process data races or deadlocks.

THREADING ERROR ANALYSIS: CUSTOM SYNCHRONIZATION PRIMITIVES MAY RESULT IN FALSE POSITIVES

Intel Inspector may report false positives for analyzed applications using customized synchronization primitives. Use of `_itt_notify` to annotate your source code can reduce these false positives.

THREADING ERROR ANALYSIS: MACROS MUST BE SET BEFORE COMPILING INTEL® TBB APPLICATIONS

To enable correct analysis of Intel® TBB applications, set the following required macros before compiling:

- If using Intel® TBB debug libraries:
  `TBB_USE_DEBUG` (which sets `TBB_USE_THREADING_TOOLS`)
- If using Intel® TBB release libraries:
  `TBB_USE_THREADING_TOOLS`

THREADING ERROR ANALYSIS: STATIC LINKING OF CERTAIN FILES MAY RESULT IN FALSE POSITIVES

Intel Inspector may report false positives if you have the Microsoft C runtime statically linked in the Windows application. On Linux, this issue may happen with `libc` or `libpthread` statically linked to the application. To work around this issue, build your application with Microsoft C runtime (on Windows) or `libc` or `libpthread` (on Linux) dynamically linked in.

Windows* Specific Limitations

Intel Inspector extension could be disabled by default in toolbar of Microsoft Visual Studio* 2019

To enable Intel Inspector in Microsoft Visual Studio* 2019 right click on toolbar and enable Inspector extension from drop-down list.

Microsoft Visual Studio* 2019 debugger integration suffers a GUI related bug

If Inspector is run with option “Enable debugger when problem detected” or “Select analysis start location with debugger”, then the detected problems are displayed as separate windows and you cannot switch between these windows manually. The workaround for this issue is to turn off the option in **Tools** -> **Options** -> **General** -> (by unchecking the checkbox) **Optimize Rendering for screens with different pixel densities.**

Antivirus software can prevent Intel Inspector from instrumenting code

Virus checking can sometimes interfere with our ability to instrument code. The following error indicates the virus checker has modified `ntdll.dll`, which will cause our collections to fail. Currently there is no workaround, except turning off the virus checker.

<”Collection failed” or some form of “Internal error”>

<date and timestamp> Collection failed. The data cannot be displayed.

[Instrumentation Engine]: SYSCALL_INSPECTOR: The NTDLL!NtTerminateThread function jumps out of NTDLL, at 0x7ffeb0fe00c4. It may be hooked by a PIN-incompatible software installed on the system Source\pin\base_w\ipc_server_windows.cpp: LEVEL_BASE::StartServer: 2216: assertion failed: res == TRUE

.NET support is deprecated

With the launch of the 2019 products, we will be listing the deprecation of .NET support in Intel Inspector. We currently only fully support .NET 3.5 which is an older technology that is seeing less and less use. We evaluated updating Intel Inspector to support newer versions of .NET, but this is prohibitively expensive and not consistent with our current business goals.

We will continue to support .NET in Intel Inspector until August of 2020, but will not be enhancing it. We plan to remove it from the product in August of 2020. This deprecation only applies to Intel Inspector; Intel® VTune™ Profiler, our performance profiler, will continue to support it.

Intel Inspector supports the .NET 4.0 Runtime with the exception of the new Task Parallel Library classes.

Manual integration into Microsoft Visual Studio* 2017 does not work

The product integrates into Microsoft Visual Studio 2017 only using official Intel Inspector installation. Any other ways like `inspxe-vsreg.exe` tool or manual launching of the .vsix package will not work.

Using Microsoft Visual Studio* Debugger “attach” command on applications under analysis results in debugging the Intel Inspector analysis tool and not the application being analyzed

To debug an application that is running an Intel Inspector analysis you must choose one of the Intel Inspector debug analysis options when starting the application. Using the Visual Studio Debugger "attach" command after starting an application under analysis does not have the same effect and will result in debugging the Intel Inspector analysis tool and not the application. When starting a debug analysis session in the documented manner, the Intel Inspector uses a probe in the Visual Studio process to intercept and modify the Visual Studio

Debugger behavior such that it controls execution and displays information about the target application rather than the analysis tool.

INTEL INSPECTOR HAS LIMITED SUPPORT FOR WINDOWS* FIBERS

Intel Inspector only has limited support for Windows* fibers.

INTEL INSPECTOR DOES NOT SUPPORT BINARIES BUILT WITH /DEBUG:FASTLINK

Intel Inspector doesn't support binaries built with "`/DEBUG:FASTLINK`" option, which is default in Microsoft Visual Studio* 2017.

WINDOWS* 8 STORE APPS AND WINRT APIS ARE NOT SUPPORTED

Intel Inspector does not support development of new Windows* 8 Store Apps and use of WinRT APIs.

MEMORY ERROR ANALYSIS: CERTAIN COMPILATION OPTIONS CAN HIDE ERRORS FROM INTEL INSPECTOR ANALYSIS

If you turn on Analyze stack accesses for a memory analysis, then make sure the application being analyzed is not compiled with the default Microsoft Visual C++* setting `/RTC1` or with `/RTCs`. These options cause a basic runtime check for stacks which uses a fill technique to detect uninitialized accesses that hides errors from the Intel Inspector's memory analysis.

THREADING ERROR ANALYSIS: APPLICATIONS USING MICROSOFT* CONCURRENCY RUNTIME FRAMEWORK MAY RESULT IN FALSE POSITIVES

Intel Inspector may report false positives when analyzing applications that use the Microsoft* Concurrency Runtime framework (Parallel Patterns Library, Asynchronous Agents Library, etc.).

THREADING ERROR ANALYSIS: INTEL INSPECTOR DOES NOT DETECT DEADLOCKS OR DATA RACES IN SYSTEM MODULES

Intel Inspector does not detect data races or deadlocks in system modules (say, modules residing in `Windows\System32` directory). For example, DX9 libraries and memory accessed by DX libraries installed in Windows system directory are not analyzed. To enable detection of the issues, move the modules to a non-system directory (say to the Windows `Program Files` directory).

THREADING ERROR ANALYSIS: INTEL INSPECTOR MAY HANG IF MULTIPLE VERSIONS OF MICROSOFT* .NET ARE USED

Using multiple versions of the Microsoft .NET* framework in the same application may cause the Intel Inspector to hang.

THREADING ERROR ANALYSIS: INTEL INSPECTOR MAY NOT CAPTURE MAIN THREAD CREATION SITE

Intel Inspector does not capture the main thread creation site if…

- the binary is built without debug symbol information.
- the .pdb symbol file is not in the location specified within the .exe or .dll executable file or in the location containing the .exe or .dll executable file.

Linux* Specific Limitations

SECURITY-ENHANCED LINUX* (SELINUX) SETTINGS ARE NOT SUPPORTED

Intel Inspector does not currently support Security-enhanced Linux* settings (SELinux); it supports only Permissive mode.

To work around this limitation, either disable SELinux by

- setting the line "`SELINUX=disabled`" in your `/etc/sysconfig/selinux` file
- adding "`selinux=0`" kernel argument in `lilo.conf` or `grub.conf` files

or make a SELinux mode adjustment by

- setting the line "`SELINUX=permissive`" in your `/etc/sysconfig/selinux` file
- asking your system administrator to make a SELinux mode adjustment

You may need to reboot your system after changing the system parameters. More information about SELinux can be found at https://www.nsa.gov/what-we-do/research/selinux/documentation/ .

INTEL INSPECTOR DEBUGGER MAY NOT LAUNCH FROM CERTAIN TERMINALS

Intel Inspector cannot launch a debugger when the terminal type konsole is used on KDE* versions older than 4.6. Use a terminal such as xterm on older KDE versions when using the analysis with debug feature.

MULTIPLE DEBUGGER SESSIONS MAY OPEN WHEN ANALYZING MULTI-THREADED APPLICATIONS

When running a multi-threaded application, use of **Enable debugger when problem detected** (`-appdebug=on-error`) can result in multiple debugger sessions being opened. Only the first occurrence will be connected and able to debug the application under analysis. The workaround is to use **Select analysis start location with debugger** (`-appdebug=on-start`) so that only one debug session will be started, then turn on analysis by entering the extended debugger command `monitor begin-analysis` and then continue to resume execution until it encounters a problem.

INTEL INSPECTOR ANALYSIS MAY FAIL IF ATTEMPTING TO STORE RESULTS IN A PARALLEL VIRTUAL FILE SYSTEM 2

If you try to store results on a Parallel Virtual File System 2 (pvfs2), Intel Inspector analysis fails with the following error: "*Error: Failed to create a database. Cannot continue*."

To work around this issue, specify a result directory location that is not on a pvfs2 file system.

COMMANDS FOR INTERACTIVE DEBUGGING WITH GDB ON FORTRAN APPLICATIONS MAY BE SLOW (INSP-217)

When using interactive debugging with gdb on Fortran programs on Linux, the 'next' and 'step' commands in the debugger can take a very long time to execute. To work around this, use the Intel® debugger with Fortran programs. The launched debugger can be changed in the GUI from the **File->Options** menu. It can be set for the CLI using the `INSPXE_DEBUGGER` environment variable or the `-debug-using` command line option.

ANALYSIS MAY FAIL IF PTRACE IS DISABLED

Memory and threading analysis require ptrace to be enabled on Linux platforms. Some newer Linux distributions disable it by default. If ptrace is disabled, it results in the message: "*Error: Failed to start the analysis because the scope of ptrace() system call application is limited."*

To work around this issue, enable ptrace by setting `/proc/sys/kernel/yama/ptrace_scope` to `1`.

INTEL INSPECTOR MAY PRODUCE ERRORS IF THE TARGET APPLICATION ATTEMPTS TO SUSPEND ALL THREADS (INSP-471)

The message "*detected an attempt to suspend an internal thread…*" happens when the target application attempts to suspend all threads, causing the Intel Inspector thread suspension as well. To avoid this error:

1. Disable the **Enable collection progress information** checkbox in the Target tab of the Project Properties dialog box.
2. Disable the **Enable interactive memory growth detection** checkbox in the Analysis Type pane.
3. Rerun the analysis.

Also avoid running an interactive debugging session during analysis.

INTEL INSPECTOR MAY BEHAVE UNPREDICTABLY OR CRASH WITH CERTAIN LINUX DESKTOP WINDOW MANAGERS

When using Intel Inspector with some Linux desktop window managers on Linux OS, there are known Linux window manager issues which may cause unexpected behavior (like disappearance of controls or even the whole GUI) or even spontaneous crashes of Intel Inspector. There is no workaround; we recommend switching to a different window manager.

THREADING ERROR ANALYSIS: INTEL INSPECTOR MAY NOT CAPTURE MAIN THREAD CREATION SITE

Intel Inspector does not capture the main thread creation site if the binary is built without debug symbol information.

7 Attributions

The license information for third party software that was used to develop Intel® Inspector can be found in `<install_dir>/licensing/third-party-programs.txt` .

For the avoidance of doubt, the Intel Inspector is solely governed by the terms and conditions of the End User License Agreement for Intel® Software Development Product that accompanies the Intel Inspector.

8 Legal Information

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

The products and services described may contain defects or errors which may cause deviations from published specifications. Current characterized errata are available on request. . No product or component can be absolutely secure.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting www.intel.com/design/literature.html.