INTEL® THREAT DETECTION TECHNOLOGY (TDT) RUNTIME THREAT DETECTION WITH HW TELEMETRY DEVELOPER GUIDE

Legal

Intel provides these materials as-is, with no express or implied warranties. All products, dates, and figures specified are preliminary, based on current expectations, and are subject to change without notice. Intel, processors, chipsets, and desktop boards may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request. Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No product or component can be absolutely secure. Check with your system manufacturer or retailer or learn more at http://intel.com. Some results have been estimated or simulated using internal Intel analysis or architecture simulation or modeling, and provided to you for informational purposes. Any differences in your system hardware, software or configuration may affect your actual performance.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained at: http://www.intel.com/design/literature.htm

All products, platforms, dates, and figures specified are preliminary based on current expectations, and are subject to change without notice. All dates specified are target dates, are provided for planning purposes only and are subject to change.

This document contains information on products in the design phase of development. Do not finalize a design with this information. Revised information will be published when the product is available. Customers, licensees and other third parties are not authorized by Intel to use code names in advertising, promotion or marketing of any product or services and any such use of Intel's internal code names is at the sole risk of the user. Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

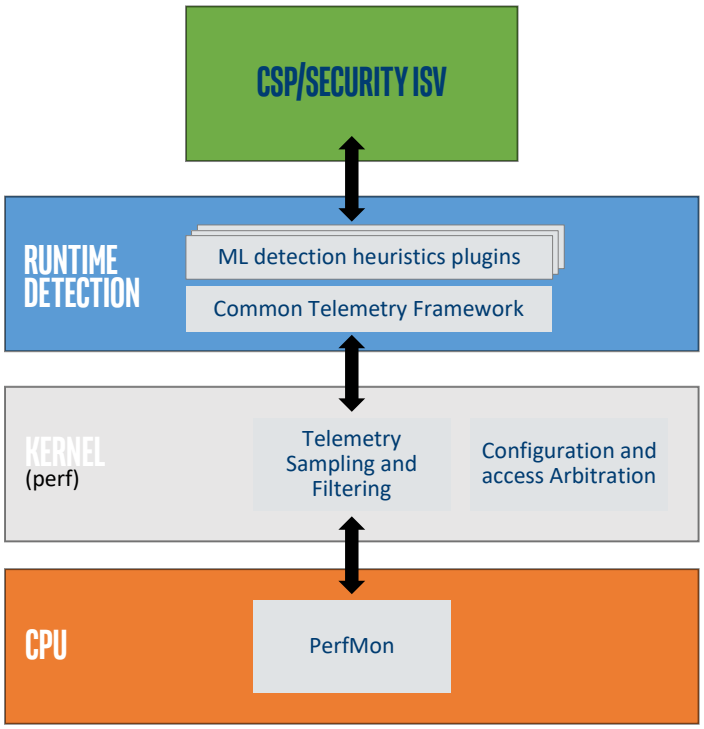
*Other names and brands may be claimed as the property of others.

Table of Contents

1 Overview

Intel® Threat Detection Technology (TDT) comprises of a set of libraries and a public API that uses low level CPU telemetry data to perform early detection of malware threats. It is intended for Independent Software Vendors (ISVs) and Cloud Service Providers (CSPs) to integrate with their security solutions and add signature-less hardware-based runtime threat detection capability. Through its solution stack Intel TDT enables deployment of advanced machine learning based classification and modeling algorithms to learn system behaviors and profile threats with CPU telemetry as feature vectors. Upon detection of malicious activity, notifications are generated for further investigation and remediation by the integrating security application (Figure 1). TDT's APIs, telemetry framework, and Machine Learning (ML) pipeline also provide security developers the opportunity to build their own detection heuristics to address threats of interest to them.

Figure 1 - TDT Solution Stack



TDT's ML based heuristics primarily use the following telemetry sources:

- Performance Monitoring Unit (PMU, aka PerfMon) counters – a set of programmable and fixed event counters maintained by the CPU PMU. Specific counter sets are identified for each threat class, filtered to reduce noise and pre-processed for feature extraction intended for ML.
- PMU metadata – derived from the telemetry sampling context in the PMU driver. Includes RIP, RSP, PID, TID and other data representative of process execution context.
- Last Branch Records (LBR) – A facility in Intel CPUs to record the most recent control flow history of executing processes.

2 Intel® Security Libraries (SecL) – DC integration

Intel® SecL-TDT is a new feature set added to Intel® SecL-DC security solutions for adding runtime threat detection and anomaly detection support with hardware telemetry and machine learning in the data center and cloud. Please visit https://01.org/intel-secl for more information on Intel® SecL-DC.

ISecL-TDT provides two components:

- **Intel® SecL Threat Detection Agent (TDA)** – a user level application running as a daemon to be deployed on each managed server node. This agent integrates the TDT libraries, starts threat detection and pushes TDT detection event reports (notifications) to the TDS.
- **Intel® SecL Threat Detection Service (TDS)** - a service or daemon that can be deployed in baremetal, VM or container that receives the threat event report from TDA and provides REST APIs for threat event query and other APIs for management.

Intel® SecL-TDT includes the `telemetry_recorder` (see Table *1*) as an example heuristic plugin.

3 Source Organization

Table *1* below describes the TDT solution stack as integrated with Intel® SecL-DC.

Table 1 - TDT Stack Component Organization

TDT Component Category	Components	Purpose

Telemetry infrastructure and ML pipeline framework	TDT Library and Go interface	Implements the TDL APIs ISecL is integrating with.
	Data pipeline and threading framework	Internal framework used by TDT to pass around telemetry data between the different plugin modules. Includes thread management framework to manage data collection and processing parallelization.
	Plugin integration interface	Common plugin framework used by all TDT plugins, e.g. heuristics. Plugins can be data collectors (e.g. collect telemetry), data processors (e.g. heuristics) or both.
Plugin Libraries	PMU publisher	Collector plugin to read PMU telemetry configuration required by a heuristic, apply to driver and read the data.
	Notification publisher	Receives detection, error and other alerts from the heuristic plugin and issues the notification via a callback interface.
	Process monitor	Special plugin to monitor running processes in the system, primarily used for whitelisting.
	Telemetry recorder	Example plugin to just collect raw PMU telemetry data and dump to a csv file. Can be used for debugging or just exercising the base telemetry framework. See section 4.1 for more info.
Configuration	Heuristic profile	Text file containing structured configuration required to execute and runtime configure the heuristic – PMU event config, detection thresholds, whitelisting, time series analysis parameters, etc. Some of these can be updated with the SetConfiguration() API.

The components listed above are organized in the directory structure listed in Table *2* below.

Table 2 - TDT Source Organization Structure

|---Application	root folder for the TDT user space libraries and tools.
| \---Plugins	root folder for TDT exploit detection heuristics and telemetry collection plugins.
| \---common	common source for telemetry framework, plugin interface and data pipeline.
| \---csv_telemetry_recorder	Telemetry Recorder (see Table *1*)
| \---library_reporter	Notification publisher (see Table *1*)
| \---pmu_publisher	PMU publisher (see Table *1*)
| \---process_monitor	Process Monitoring and whitelisting (see Table *1*)
| \---API	Internal API interface used by the TDT telemetry framework, ML and data pipeline.
| \---library	TDT public API library, includes C, C++, Go interface definitions.

3.1 TDT API

Public API intended for applications integrating TDT libraries.

C – `Application\library\tdt_agent.h`

C++ – `Application\library\tdt_agent.hpp`

Go - `Application\library\golang\src\tdt_lib\tdt_agent.go`

Please refer to code documentation in the headers for invocation details.

3.2 TDT Plugin API

Internal interface that TDT plugins implement to plug into the core framework. This is defined in C++.

C++ - `Application\API\plugin_api.h`

4 Example Plugins

4.1 Telemetry Recorder

TDT provides an example plugin to showcase how to configure the PMU for data sampling and exercise the TDT data pipeline to receive the telemetry data in batches. The Telemetry Recorder plugin enables collection of up to 4 PMU events at a time and then records the collected record batches in a csv file in the local directory.

The plugin implements the TDT Plugin API interface mentioned in section 3.2. Security vendors wishing to innovate with the TDT framework can use this plugin as reference and build their own heuristics based on it.

When installed the telemetry recorder will be discoverable via the TDT *discover()* API as '`telemetry_collect`'.

4.2 Plugin Configuration

Each TDT plugin needs a configuration file that specifies the hardware events to be used by the associated heuristic. This configuration is loaded by the plugin and applied to the PMU publisher plugin for programming the hardware (e.g. CPU PMU) and collect the data. Plugin config files have a `.profile` suffix.

The telemetry collector plugin's configuration is located in Application\library\configuration\linux\telemetry_collect_tdtlib.profile

PMU Events/Counter Lists

A full list of PMU events available for programming via the TDT telemetry framework can be found at https://download.01.org/perfmon/index/. Set the CPU family selector box to 'Skylake Client Platform' to see the events available on Intel 6th (Skylake) and 7th gen (Kabylake) CPUs.

Note for trying out new PMU configuration and interrupting thresholds
It is not feasible for the built-in perf module to validate all possible PMU event combination and interrupting thresholds. Incorrect configuration may cause machine-check errors which would lead to OS crashes. Setting very low interrupting thresholds on fast moving counters (e.g. ref cycles, cache misses, etc.) can also cause 'interrupt storms' which the driver may not be able to keep up with. It is thus advisable to review experimental PMU configuration thoroughly prior to applying.

4.3 Threat Detection with Intel® SecL-TDT stack

Intel demonstrated runtime threat detection utilizing the TDT framework at the RSA Conference 2019 with AI-ML based heuristics that use a set of PMU events, recorded through the TDT PMU publisher (see Table *1*), to profile the execution of specific exploits. We are encouraging the developer community to take advantage of the TDT framework to develop their own threat detection heuristics.

5 Hardware Requirements

TDT threat detection heuristics are dependent on CPU telemetry generated by Intel CPUs and are thus applicable only on Intel platforms.

Individual heuristics may have their own minimum CPU SKU requirement depending on the kind of metrics being used by the heuristic.

TDT is only supported on Intel 6th generation (SkyLake) and later Core and Xeon CPU families at this time.

A minimum of 8GBs of RAM is recommended.

6 Build and Runtime Requirements

- Linux Kernel 3.1 or up.
- CMake 3.11.2 or up. (https://cmake.org/)
- GCC
    - 7.X or better on non-RHEL distributions
    - 4.8.5 or better with devtoolset-7 on RHEL distributions
- Boost 1.68 or up (https://www.boost.org/)
- Intel Thread Building Blocks (TBB: https://software.intel.com/en-us/intel-tbb)
- Perf (included with kernel). This is required for execution only, no build dependencies.

Please setup internet settings to let CMake download the proper versions of Boost and TBB during the build process.

6.1 Root/Admin privilege

The TDT PMU publisher plugin interfaces with the perf kernel module that comes built-in with the Linux kernel. This requires the plugin to execute with root/admin privileges and have permissions to invoke the `perf_event_open` syscall.

Security applications integrating TDT may execute directly with root/admin privileges. Alternatively, administrators can use kernel namespaces and control groups to allow `perf_event` subsystem access for the app from the non-root user or isolation context (e.g. containers) it is executing in.

7 Build Instructions

TDT is built using CMake.

- Open a terminal and set proxy settings allow download of Boost and TBB automatically (This step is only required in case of using proxy to access the Internet)

```
$ export HTTP_PROXY=http://<proxy>:<port>
$ export HTTPS_PROXY=https://<proxy>:<port>
```

- Navigate to the root of the TDT source directory (see Table *2*).

- Create the build directory and configure and build the TDT libraries and release package

```
$ ./compile.sh
```

- The build binaries and configuration files will be copied to the `release_package` folder in the source root folder.

8 Support

8.1 Mailing List

TDT Libraries and Telemetry Framework – tdt.ae@intel.com

TDT-Intel® SecL integration - https://01.org/intel-secl

9 Bugs

Bugs and security vulnerabilities can be reported on the mailing list https://01.org/intel-secl or using our bug tracking system, at https://github.com/intel/isecl/issues.

10 Documentation

Product documentation and tutorials can be found here: https://01.org/intel-secl