WHITE PAPER

Intel® Software Guard Extensions (Intel® SGX)

Performance Considerations for Intel® Software Guard Extensions (Intel® SGX) Applications

Scope

This paper covers four areas with respect to Intel® Software Guard Extensions (Intel® SGX) enabled applications where, depending on how an application is designed/behaves, noticeable performance impacts can be experienced. The article explains the reasons behind these potential impacts and makes recommendations to minimize them. The article assumes a basic knowledge of Intel SGX. General information on Intel SGX is provided on the Intel SGX portal at: https://software.intel.com/en-us/sgx.

Introduction

Intel SGX is designed to protect select code and data from disclosure or modification through the use of memory enclaves, supported by Intel® architecture instructions and additional hardware assistance in the processor. At application runtime, the Intel SGX instructions build an enclave with a restricted entry and exit location. Enclave code and data inside the CPU perimeter runs in the clear and enclave data written to disk is encrypted and checked for integrity.

This combination of a new security model, with new processor instructions and hardware assistance, adds some overhead when an application creates and uses enclaves. This is the classic security versus performance tradeoff. There are some situations, however, where, depending on the application design and behavior, code running in or related to enclaves can result in some *additional* overhead.

Fortunately, the reasons for this additional overhead are understood and judicious design decisions can be made to reduce/eliminate them, while still enjoying the security benefits of Intel SGX. The following subsections describe four areas in Intel SGX application behavior where an additional overhead can be experienced, describes the reasons for this additional overhead, and provides recommendations to minimize it. These four areas are:

- Enclave creation
- Transitioning to/from enclaves
- Excessive cache misses
- Excessive enclave paging

Enclave creation

Enclave creation is the first area to consider. As the following discussion makes clear, enclave size greatly affects the time it takes to create an enclave because enclave measurement occurs to ensure the code loaded into the enclave is trusted.

During enclave creation, a series of EADD and EXTEND instructions are run to load and measure enclave pages.

- Each EADD instruction records EPCM information in the cryptographic log stored in the SECS and copies 4 Kbytes of data from unprotected memory outside the EPC to the allocated EPC page.
- Each EEXTEND instruction measures a 256 byte region (generating a cryptographic hash), which means it takes 16 invocations of EEXTEND to measure the 4KB page created by EADD. Each of the 16 invocations of EEXTEND adds to the cryptographic log information and to the measurement of the section.

Since cryptographic processing takes processor cycles, the time to create an enclave scales directly with the size of the enclave, because each additional 4KB page that an enclave uses results in cryptographic processing that occurs for the EADD instruction *and* cryptographic processing for the 16 EEXTEND instructions to measure that 4KB page. Creation of the enclave is visible to the application.

If you are concerned that the enclave creation time for your application is impacting its overall performance, consider the following approaches to reduce this impact:

- Reduce the size of your enclave. Scrutinize each code and data element currently in your enclave and move every element that does not reasonably need to be there to the untrusted part of your application. (You can use Intel® VTune™ XE Amplifier to help experiment with how long it takes to create various size enclaves.)
- Enclave Dynamic Memory Management (EDMM) available with SGX2 allows enclaves to expand after creation. When your OS supports EDMM, your application can create a smaller size enclave at first, then expand it later when additional enclave space is needed. This shifts some of the time to copy pages and measure regions from enclave creation time to a later point in time.
- Look for ways to hide the enclave loading time by having the application perform processing that occupies the user's attention.
- Avoid excessive enclave destruction/reloads to minimize repeating load overhead.

Note: This discussion includes a subset of enclave creation actions. For details, see *Constructing an Enclave* in *Intel® 64 and IA-32 Architectures Software Developer's Manual: Volume 3D.*

Transitions to/from enclaves

Transitioning control to/from enclaves is the second area to consider. The following discussion explains how the amount of data marshalled back and forth between the untrusted part of an application and the enclave greatly affects transition time to/from enclaves. Also discussed is balancing the time an application spends executing in an enclave versus how often that application enters/exits the enclave.

Transitions *to* an enclave and *from* an enclave resemble a context switch in many ways. When an EENTER instruction is executed to transition control into an enclave, register state and other information regarding the untrusted state is saved; then inside the enclave thread state and other information regarding the trusted state is loaded so execution can begin in the enclave. Much of this is performed by SDK generated code. A reverse process occurs on the transition from an enclave, initiated by an EEXIT instruction. The trusted thread state information is saved; then untrusted register state and other information is loaded. Security checks are also performed during all transitions. Again, much of this is performed by SDK generated code. These actions constitute a *fixed* element of the overhead associated with transitioning to/from enclaves.

However, there is also a *variable* element of the overhead associated with transitions at runtime. Parameters are marshalled from the untrusted part of the application to the trusted part, and return values are un-marshalled. In the trusted part, parameters from the untrusted part are un-marshalled and return values for the untrusted part are marshalled. Since parameters can vary greatly in size, if an application is passing large parameters between the two parts of an application, a noticeable overhead can be experienced.

If the overhead associated with transitions to/from enclaves for your application is impacting its performance, consider the following approaches to reduce this impact:

- Reduce the total size of parameters being passed. Examine each parameter currently being passed between the untrusted part of the application and the enclave and remove non-critical parameters. If possible, reduce the

remaining parameters to their smallest reasonable size. You can use Intel® VTune™ XE Amplifier to compare transition time with frequency of transitions to help achieve a healthy performance balance.

- If the code in your enclave needs to operate on large data structures, you may wish to pass a pointer to the data structure into the enclave instead of the actual data. This is *allowed* using the `user_check` parameter with pointers in the Enclave Definition Language (EDL) file. However, there is a security risk because the Edger8r tool does not verify the pointer before passing it to the enclave when `user_check` is used. *You as the developer must ensure that you are not exposing secrets in untrusted memory. You must also implement your own pointer verification if you choose this method.* For details on the EDL `user_check` parameter, see this discussion in the *Intel Software Guard Extensions SDK Developer Reference*: https://software.intel.com/en-us/node/708978. For details on the functions used to determine if a pointer and its associated data is inside or outside the enclave, see this discussion in the *Intel Software Guard Extensions SDK Developer Reference:* https://software.intel.com/en-us/node/709040.
- If transition time is a concern, you may want to investigate other approaches, such as implementing an *Exit-less Service*. With exit-less services, parameters are held in a buffer in the untrusted part of the application only; the trusted part polls the buffer looking for notification to perform its work on the data. So no transition occurs. This trades transition overhead for dedicating an enclave thread for additional polling, which may be appropriate in some cases. The following link allows you to download a paper describing exit-less based services: https://sites.google.com/site/silbersteinmark/Home/cr-eurosys17sgx.pdf?attredirects=1.

Notes:

- When an interrupts occurs while executing in an enclave, an Asynchronous Exit (AEX) occurs, which results in a transition from the enclave. An AEX has more overhead associated with it than a standard (non-Intel SGX) interrupt context switch. Since interrupts are under the control of the OS, there is nothing you can do in your application to control this. But it's good to understand that interrupts may effect perceived application performance.
- This discussion describes a subset actions that occur during enclave transitions. For details, see *Enclave Entry and Exiting* in *Intel® 64 and IA-32 Architectures Software Developer's Manual: Volume 3D.*
- For more information on data marshalling, see the discussion of Proxy Functions in Part 7 of the Intel SGX Tutorial Series.

Excessive cache misses

The third area to consider is the effect of cache misses, which in some situations can result in an overhead increase for Intel SGX applications. The general actions (and overhead) associated with a cache write to system memory or a cache line fill in a non-SGX enabled system are very well known. However, Intel SGX adds another dimension to cache misses, as all memory transactions that are outside the portion of the processor cache used for an enclave are protected. This protection adds some overhead when fetching cache lines from the memory. This overhead will be model-specific to the implementation of Intel SGX. (A technical whitepaper describing the initial implementation can be found at: http://eprint.iacr.org/2016/204.pdf.)

Intel SGX architecture adds two potential overhead elements to each cache miss beyond typical cache overhead:

- The time to perform integrity check/anti-replay check for each cache line not currently in the processor cache, and to update the data structure in system memory (if needed). This overhead depends on the memory access pattern.
- The time to encrypt/decrypt the actual data being moved between the cache and system memory.

This additional overhead could become significantly greater as the frequency of cache misses increases. (Note that memory access inside the enclave already in the cache are not impacted. And accessing memory outside the enclave from inside the enclave has little impact.)

If your application is experiencing overhead associated due to a high number of cache misses, consider taking the following steps:

- Reduce the size of your enclave's data. Inspect your data to ensure that only critical elements are in the enclave. Less data means less encryption/decryption and less data structure checking by the Intel SGX memory control/protection mechanism. You can use Intel® VTune™ XE Amplifier to inspect caching behavior in your application to help make tuning decisions.
- Consider the recommendations of the following sources to help create a more "cache friendly" application:

  https://software.intel.com/en-us/articles/resolve-cache-misses-on-64-bit-intel-architecture.

  https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf

Excessive writing of pages

The fourth area to consider is how extensive page writes in Intel SGX applications can increase overhead, and how to minimize that effect. Typically OS's support paging, which includes some overhead; however, the overhead associated with paging in Intel SGX is greater, as described below:

Intel SGX uses secure storage, called the Enclave Page Cache (EPC), to stored enclave contents. Enclave pages are 4KB in size. When enclaves are larger than the total memory available to the EPC, enclave paging can be used by privileged SW to evict some pages to swap in other pages. The CPU performs the following steps using the EWB instruction when the OS swaps out an enclave page:

- Read the SGX page to be swapped out (evicted)
- Encrypt the contents of that page
- Write the encrypted page to unprotected system memory

Since this process has an inherent overhead associated with it, the more pages that are swapped out, the more often the overhead is incurred.

To prevent your application from experiencing the additional overhead associated with excessive page writes, do what you can to ensure enclave size is less than the EPC. By including only secrets and the code to operate on them in your enclaves, you can help minimize the chances of incurring paging overhead. You can use Intel® VTune™ XE Amplifier to inspect paging behavior in your application to help make tuning decisions.

Notes:

- While developers have control over use of finite resources in their application design, users/usages also have significant control over how many applications are running. If users/usages end up causing a lot of enclaves to be run, enclave performance can be impacted, despite the efforts you put into performance optimization.
- The following technical forum discussion provides additional details related to paging of enclave code/data: https://software.intel.com/en-us/forums/intel-software-guard-extensions-intel-sgx/topic/722444

Additional Performance Notes

If your application is multi-threading, look into optimizing data synchronization, locking, the threading model, and the memory allocation algorithm selected to improve performance.

- The Intel SGX SDK (for Microsoft Windows and Linux) has some synchronization and locking primitives that are already optimized.

- For heavily threaded applications, it may be better to select one memory allocation algorithm over another. The Intel SGX SDK for Linux V1.9 (and later) supports TCMalloc memory allocation, which can result in a significant performance increase for many heavily-threaded applications over the dlmalloc alternative.

Summary

Intel SGX provides the tools to help protect code and data from malware in a wide variety of applications. The flexibility in application design can, in some cases, result in a noticeable overhead being experienced at application runtime that goes beyond the overhead incurred in the classic security versus performance tradeoff.

Developers that understand the potential overhead in the areas described in this article and apply the recommendations made here, can go a long way to creating applications that do not experience these addressable performance issues.

References

References listed under each subtopic.

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. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, 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.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL® AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL® OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL® PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

No computer system can provide absolute security under all conditions. Built-in security features available on select Intel® processors may require additional software, hardware, services and/or an Internet connection. Results may vary depending upon configuration. Consult your system manufacturer for more details.

Intel®, the Intel® Logo, Intel® Inside, Intel® Core™, Intel® Atom™, and Intel® Xeon® are trademarks of Intel Corporation in the U.S. and/or other countries. Other names and brands may be claimed as the property of others.

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

Copyright © 2018 Intel® Corporation