EnclaveSim:A Micro-architectural Simulator with Enclave Support

Yashika Verma
Dept. of CSE
Indian Institute of Technology Kanpur
yashikav@cse.iitk.ac.in

Dixit Kumar
Dept. of CSE
Indian Institute of Technology Kanpur
malviyadx@gmail.com

Biswabandan Panda
Dept. of CSE
Indian Institute of Technology Bombay
biswa@cse.iitb.ac.in

***Abstract*—Intel SGX preserves the confidentiality and integrity aspects of data and code through enclaves (that reside in the trusted part of the memory) and protects it from different layers of the malicious system software, including the OS. Micro-architecture research in the presence of SGX is an interesting theme to explore as SGX does not mitigate timing side-channel attacks at various levels of a memory hierarchy and causes significant performance slowdown. The research community extensively uses existing benchmark suites like SPEC CPU 2017 for evaluating new proposals on the various aspects of micro-architecture research. As there is no benchmark suite available for micro-architecture research with SGX, state-of-the-art micro-architecture research in the presence of SGX assumes an entire SPEC benchmark is running inside an enclave. In reality, Intel SGX assumes that a major portion of the application's code and data do not require security, and only a tiny fraction of it needs security via an enclave. To the best of our knowledge, there are no open-source micro-architectural simulators that can simulate Intel SGX fairly, for micro-architecture research. In this regard, we propose EnclaveSim, a detailed yet flexible, trace-based micro-architectural simulator that simulates trusted code execution through enclaves.**

I. Introduction

Trusted Execution Environment (TEE) provides confidentiality and integrity through secure code execution via enclaves. Intel SGX [1] [2] [3] is one of the popular TEEs. Although Intel SGX ensures secure code execution, it is still vulnerable to timing attacks at the various micro-architecture units such as caches. In addition to this, Intel SGX provides confidentiality and integrity at the cost of system performance, which motivates the computer-architecture community to improve the micro-architecture performance in the presence of an enclave for performance and thwarting timing channels. State-of-the-art research in terms of evaluating different micro-architecture ideas with the enclave [4], mostly use existing benchmark suites like SPEC CPU benchmarks. These works evaluate the performance of an enclave by running the entire benchmark inside an enclave. However, Intel SGX primarily assumes that an application will need enclave support only for a small portion of the code and the corresponding data, and not for the entire application; one of the main reasons because of which an enclave page cache is assigned uses few hundreds of MBs only.

The problem: As there is no specific benchmark suite available for micro-architecture research in the presence of an enclave, recent works assume the entire benchmark will run inside an enclave. Also, there is no micro-architectural simulator that can simulate the micro-architecture faithfully in the presence of an enclave.

Our approach: To tackle the existing problems, we propose a highly modular open-source simulator, EnclaveSim. EnclaveSim simulates an application code in an enclave. We build EnclaveSim on top of a trace-driven micro-architectural simulator, ChampSim [5]. EnclaveSim uses an application trace generated by the Intel Pin tool [6] and isolates the trace into the enclave and non-enclave accesses. It simulates the enclave accesses with enclave support while non-enclave accesses do not demand security, without enclave support. We provide special knobs to traces, that help in enforcing isolation of enclave and non-enclave portion of the trace. For micro-architecture research that involves the memory system in the presence of an enclave, we provide a way using which we can distinguish whether a memory request is for the enclave portion of the application or not. Through this feature, the micro-architecture community can concentrate on improving the memory system only for a tiny part of the application that is running within an enclave. In a nutshell, we make the following contributions: (i) we propose the first open-source simulator for enclave execution. (Section III), (ii) we show the effectiveness of EnclaveSim by simulating different parameters of interest. We also discuss the flexibility aspect of EnclaveSim and the various rich features that we provide. (Section IV). Enclavesim is available at: https://github.com/YashikaVerma156/EnclaveSim.git

II. Background

A. Intel SGX

Intel SGX is an x86 architecture extension specific to Intel processors that allows an application to protect its code/ data from various system software layers: host OS, hypervisors, and other privileged software. SGX enables part of an application or complete application to run inside a secure, isolated container known as the *enclave*. SGX provides confidentiality and integrity guarantees of data residing within an enclave using the Memory Encryption Engine (MEE) [7].

At a 10K feet view, SGX reserves up to 256 MB of DRAM space as Processor Reserved Memory (PRM). PRM holds Enclave Page Cache (EPC) that stores sensitive code/data and metadata associated with the enclaves. The processor restricts any non-enclave memory access in this PRM region, including

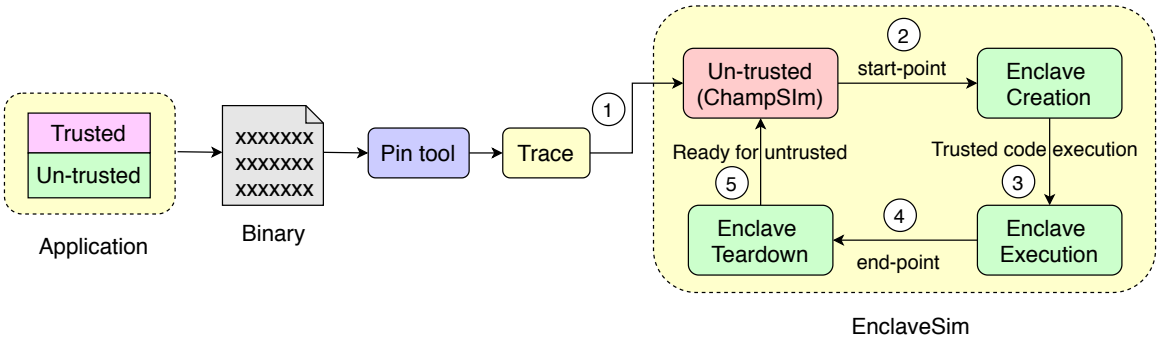

Fig. 1: **High Level Design of EnclaveSim.**

system software accesses. Current Intel processors support PRM of size 64, 128, and 256 MB [8], which is configurable at boot time. SGX also provides a set of instructions to support enclave memory management and secure code execution. Some of the ring-0 level instructions such as `ECREATE`, `EADD`, and `EINIT` manage EPC and ring-3 level instructions such as `EENTER`, `ERESUME`, and `EEXIT` allow user applications to run their code inside an enclave.

1) Life Cycle of an Enclave: The life cycle of an enclave consists of three major phases: creation, execution, and teardown. The OS uses `ECREATE` instruction to initiate the enclave creation process; it allocates a region of virtual memory within an application for trusted code and data. An instruction such as `EADD` loads initial code and data that is supposed to run within an enclave into the EPC region. While loading an enclave, the OS uses `EEXTEND` to compute the cryptographic hash of the content. After loading, `EINIT` establishes the enclave identity and sets up the enclave for execution. The application then uses `EENTER` to switch the processor into enclave mode and starts secure code execution. After executing the trusted code, the application uses `EEXIT` to clear the enclave mode and returns the execution control to the caller untrusted code.

When an enclave finishes its job, the OS uses `EREMOVE` instruction to deallocate corresponding EPC pages and destroys the enclave. **ChampSim:** ChampSim [5] is a trace-based micro-architectural simulator that simulates a detailed memory hierarchy with an out-of-order processor. Recent micro-architecture championships at ISCA 2015, ISCA 2017, ISCA 2019, and ISCA 2020 use ChampSim as the infrastructure to evaluate data prefetchers, instruction prefetchers, and cache replacement policies.

III. High-level design

EnclaveSim provides two flavors of enclave simulation: (i) Programmer-defined, where the programmer, while writing the code, specifies which part of the application code should run within an enclave. Then it generates the application's trace using the Intel Pin tool, which marked trusted code/data differently in the trace. We term this trace as enclave aware trace. EnclaveSim uses this trace to decide which portion of the application should run inside an enclave or not. (ii) User-defined, where the user generates trace of an existing binary which is not enclave aware. EnclaveSim provides different knobs to simulates specific portions of these traces inside the enclave.

EnclaveSim extends the functionality of ChampSim to support enclave simulation. EnclaveSim provides two additional features on top of ChampSim: (i) It mimics the life cycle of enclaves, and performs enclave memory management (ii) It provides a unique enclave-id field in data packets to distinguish between trusted and untrusted requests throughout the memory hierarchy.

A. Design overview

EnclaveSim takes an application trace generated by the Intel Pin tool and simulates multiple enclaves per application. We propose EnclaveSim in two flavors: (i) **Programmer-defined flavor**, where the programmer marks a certain portion of the code and data to be executed inside an enclave. For example, if a programmer wants to run a specific function of SPEC CPU 2017 benchmark `mcf` inside an enclave, then the programmer has to specify the region of trusted code by wrapping it inside two function calls, `enable_trusted_code_execution()` and `disable_trusted_code_execution()`. Our proposed tracer translates these function calls to start-point and end-point in the trace file, which indicates enclave entry and exit. Thus, programmer-defined flavor demands code re-writing of existing benchmarks. (ii) **User-defined flavor**, where the programmer has privilege to use the existing benchmarks. For this flavor, we provide special run-time knobs in EnclaveSim that specify *start-point* and *end-point* of an enclave. Here, *start-point* defines the entry point, from where an enclave starts its execution, and *end-point* defines the exit point, at which the enclave finishes its execution, in terms of instruction numbers.

Figure 1 shows a high level design of EnclaveSim. EnclaveSim extends ChampSim with three additional modules: enclave creation, execution, and teardown. EnclaveSim starts execution of an application with the untrusted (non-isolation) part of the code (①). When execution of an application reaches the start-point (②), EnclaveSim sets up the enclave using the enclave creation module and enables the processor to execute trusted code (③). Once the enclave is set up, enclave execution module performs various functions such as enclave memory management, initializing an enclave-id and communicating the same throughout the memory systems, and imitating the MEE cryptographic operations. During an enclave execution, when the enclave reaches its end-point, EnclaveSim assumes the enclave has finished its execution and starts the

enclave teardown phase. (④) Enclave teardown module releases the memory resources occupied by the enclave, switches the processor into the non-enclave mode, and resumes the untrusted code execution. (⑤).

B. Implementation Details

Enclave Creation: Enclave creation module uses `check_enclave_init()` function to keep track of an application's execution. When execution reaches the start-point for an enclave, this module switches the processor's mode to an enclave mode using `enable_enclave_mode()`. The processor starts enclave code execution only after the enclave mode is enabled.

Enclave Execution: Enclave execution module is the heart of EnclaveSim. The function `va_to_pa_enclave()` manages EPC and performs enclave page allocation/eviction. It takes a virtual address (`va`) as an argument and allocates a physical page in EPC or NON-EPC region, depending on whether the processor is in enclave mode or not (see Figure 2). When an enclave page-fault occurs, `va_to_pa_enclave()` selects the least recently used page from EPC, moves the content of the selected page to a NON-EPC region, and allocates the selected page to the `'va'` translation request. In EnclaveSim, on an SGX page swap, we flush the TLB entries used by the enclave and add a latency of an average 40K processor cycles [9] [10] for each enclave page-swap. This overhead includes system calls, context switches, and page swaps.

Enclave execution module uses the function `assign_enclave_id()` to assign an enclave-id (`eid`) to all the memory requests generated by the application in its enclave mode; `'eid'` helps in identifying a particular enclave when multiple applications are running their trusted code inside the enclaves. On every off-chip EPC access, i.e., on LLC miss and LLC write-back, MEE encrypts/decrypts the data blocks and checks EPC data integrity. EnclaveSim adds a combined latency of around 500 cycles for encryption/decryption and integrity verification. The integrity verification latency varies according to the height of the integrity tree [9]. The function `check_enclave_endpoint()` keeps track of execution of an enclave, once the execution reaches the end-point, it calls enclave teardown module.

Enclave Teardown: Enclave teardown module has three major functions: `invalidate_page()`, `deallocate_page()`, and `disable_enclave_mode()` (see Figure 3). The function `invalidate_page()` flushes cache lines, and TLB entries of the corresponding enclave page (`epage`) while `deallocate_page()` removes the `epage` from the EPC region. It further swaps-out the `epage` to NON-EPC region, in case the `epage` is dirty. EnclaveSim adds a latency of 12K cycles for every dirty page swap-out. At the end, Enclave teardown module uses `disable_enclave_mode()` to reset the processor enclave mode and resume the untrusted code execution.

IV. Evaluation and Validation

We evaluate EnclaveSim on both single-core and multi-core configurations with programmer-defined and user-defined enclaves. Due to space constraints, we are showing results

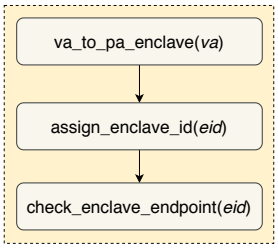

Fig. 2: **Enclave Execution**

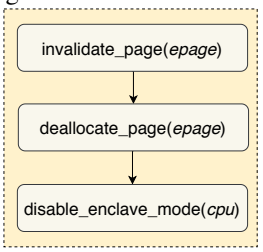

Fig. 3: **Enclave Teardown**

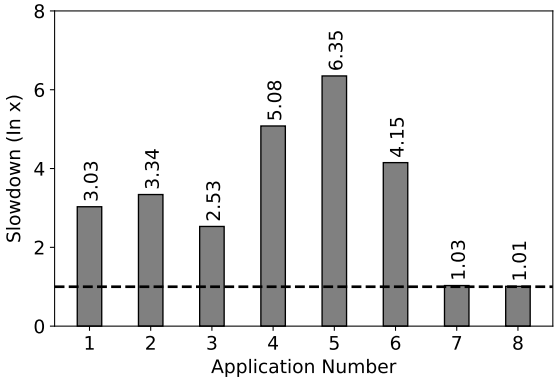

Fig. 4: **Slowdown with enclave execution for single-core.**

for user-defined enclaves only, with a multi-core environment. Table I shows various parameters used in the experiments. We define two variants of applications: enclave application (EA) and non-enclave application (NEA). An EA has both trusted and untrusted code, while an NEA consists only of the untrusted code. We associate different parameters with EA and NEA. An EA has three parameters: application's name, number of enclaves, and a <start-point, end-point> pair for each enclave. An NEA only has one parameter: the application's name. We use a subset of SPEC CPU 2017 benchmarks for creating mixes. The subset selection is made based on application cache sensitivity, which is measured in terms of Misses Per Kilo Instructions (MPKI) at LLC (see Table II). EA [A, 1, {10, 35}] specifies an enclave application, in which *'A'* represents benchmark name (refer Table II), *'1'* denotes number of enclaves and the pair *{10, 35}* denotes start-point and end-point of the enclave in terms of millions of instructions.

Experiments: Figure 4 shows the performance overhead due to enclave execution. A high MPKI application having 20% portion running within an enclave (like App-1, 2, and 3) causes a slowdown of 2 to 3X. However, when we increase the trusted code portion to 40% for the same high MPKI applications (App-4, 5, and 6), we observe even a steeper

TABLE I: **Parameters of simulated system.**

Processor	8-core, 4GHz, out-of-order
L1-I, L1-D, L2, Last-level Cache (LLC)	32KB, 48KB, 512KB, 2MB/core, 64B line size
DRAM Controller	DDR3 3200 MHz (40-40-40)
EPC size	192 MB
Enclave LLC-miss/major-fault/minor-fault/teardown latency	500/40K/28K/12K cycles

TABLE II: **Selected SPEC CPU2017 benchmarks.**

Naming	Benchmarks	LLC MPKI
A, B, and C	623.xalancbmk, 605.mcf, and 602.gcc	High
D and E	641.leela and 648.exchange2	Low

TABLE III: **Applications for single core simulation.**

App. No.	Enclave applications
1	EA [A, 1, {20, 10}]
2	EA [B, 1, {20, 10}]
3	EA [C, 1, {20, 10}]
4	EA [A, 2, {5, 10}, {20, 10}]
5	EA [B, 2, {5, 10}, {20, 10}]
6	EA [C, 2, {5, 10}, {20, 10}]
7	EA [D, 2, {5, 10}, {20, 10}]
8	EA [E, 2, {5, 10}, {20, 10}]

slowdown of 4 to 6X. In contrast, for low MPKI applications (like App-7, 8), we observe negligible slowdown even with 40% of trusted code. Figure 5 showcases the contribution of individual enclave event overhead in the overall slowdown. Enclave-minor-fault and enclave-LLC-miss are the major contributors of performance slowdown. We validate the correctness of EnclaveSim by correlating enclave event overheads with the overall performance slowdown. Table IV shows different kinds of 8-core mixes that we simulate on EnclaveSim. We create the mixes from a pool of SPEC'17 benchmarks and simulate 50 million instructions with a 10 million warm-up. We create mixes based on two features: EPC sensitivity and LLC sensitivity. We define the three categories as shown in Table IV. Mixes with category ET-LT cause a slowdown in range of 4-28X, while EPC-Fitting mixes EF-LT and EF-LF cause a slowdown up to 1.43X. These slowdown numbers vary according to the number of EAs, duration of enclave execution, and LLC sensitivity of the applications.

Extensibility: EnclaveSim is easy to extend and can be integrated with many state-of-the-art enclave based proposals. EnclaveSim leverages capabilities of the underlying simulator ChampSim, which allows EnclaveSim to study important micro-architectural features like prefetchers, cache replacement policy, TLB management, page-table walkers, DRAM controllers for enclaves.

Security evaluation: EnclaveSim can be used for micro-architecture defenses. For example, page-based and cache based timing channels are some of the side-channels that are prevalent in Intel SGX too. Defences like MI6 [4] can be easily implemented. We provide the implementation of MI6 as a case study with our Github link, so that the architecture community can use it easily. Apart from defenses, even the attacks that exploit micro-architecture units can be reproduced.

TABLE IV: **List of mixes used in 8-core simulations. EA: Enclave application, NEA: Non-enclave application.**

Category	Mix no.	Mix
3*ET-LT	1	8 EA [A, 2, {10, 35}, {40, 45}] + 0 NEA
	2	8 EA [B, 2, {10, 35}, {40, 45}] + 0 NEA
	3	8 EA [C, 2, {10, 35}, {40, 45}] + 0 NEA
3*EF-LT	4	4 EA [A, 1, {25, 30}] + 4 NEA[A]
	5	4 EA [B, 1, {25, 30}] + 4 NEA[B]
	6	4 EA [C, 1, {25, 30}] + 4 NEA[C]
2*EF-LF	7	8 EA [D, 1, {10, 35}] + 0 NEA
	8	8 EA [E, 1, {10, 35}] + 0 NEA

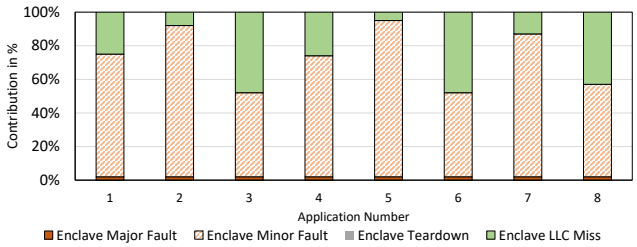

Fig. 5: **Major contributors of performance slowdown.**

V. Conclusion

We proposed EnclaveSim for detailed micro-architecture research involving memory systems in the presence of an enclave. EnclaveSim comes with two flavors: programmer-defined and user-defined. EnclaveSim will help the micro-architecture community for security and performance trade-off research.

VI. Acknowledgement

This work is supported by the SRC grant SRC-2853.001.

References

[1] Victor Costan et al., Intel SGX explained. Cryptology ePrint Archive, Report 2016/086, February 2017. https://eprint.iacr.org/2016/086.pdf.
[2] *Intel® Software Guard Extensions Programming Reference*, Intel Corp., October 2014. Ref. #329298-002US
[3] Matthew Hoekstra et al., Using Innovative Instructions to Create Trustworthy Software Solutions. *In 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, 2013
[4] T. Bourgeat et al., Mi6: Secure enclaves in a speculative out-of-order processor. *In MICRO*, pages 42-56, 2019.
[5] ChampSim, *https://github.com/ChampSim/ChampSim*
[6] Chi-Keung Luk et al., Pin: building customized program analysis tools with dynamic instrumentation. *In PLDI* pages 190–200, 2005.
[7] Shay Gueron, A memory encryption engine suitable for general purpose processors. Cryptology ePrint Archive, Report 2016/204, 2016. https://eprint.iacr.org/2016/204.
[8] *10th Generation Intel® Core™ Processor Families, Datasheet, Volume 1 of 2.* Intel Corp., April 2020. Document Number: 341077-003.
[9] Meysam Taassori et al., VAULT: Reducing paging overheads in SGX with efficient integrity verification structures. *In 23rd ASPLOS,* pages 665–678, 2018.
[10] Meni Orenbach et al., Eleos: Exitless OS services for SGX enclaves. *In EuroSys,* pages 238–253, 2017.