creative commons
COMMONS DEED

저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:

저작자표시. 귀하는 원저작자를 표시하여야 합니다.

비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.

변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 이용허락규약(Legal Code)을 이해하기 쉽게 요약한 것입니다.

Disclaimer

dCollection

공학석사 학위논문

Intel TSX를 활용한 ORAM 가속 기술 탐구

2021년 2월

서울대학교 대학원

전기·정보공학부

장 지 원

Intel TSX를 활용한 ORAM 가속 기술 탐구

지도교수 백 윤 흥

이 논문을 공학석사 학위논문으로 제출함
2021년 2월

서울대학교 대학원
전기·정보공학부
장 지 원

장지원의 석사 학위논문을 인준함
2021년 2월

위 원 장 문 수 묵 (인)

부위원장 백 윤 흥 (인)

위 원 이 병 영 (인)

국문초록

원격계산이 일상이 된 클라우드 컴퓨팅 및 빅데이터 시대가 오면서 개인정보보호가 더욱더 중요해지고 있지만, 클라우드 내에서 사용자들의 데이터들이 복호화된 상태로 처리되어지기 때문에 여러 보안문제가 발생하기 쉽다. 이에 따라, Intel SGX는 블랙박스와 같은 하드웨어 기반의 신뢰실행환경인 Enclave를 통해 프로그램의 기밀성 및 무결성을 보장하여 클라우드 상의 사용자들의 민감한 데이터들을 안전하게 보호한다. 그러나, 최근 Intel SGX가 타이밍 및 접근패턴에 기반한 여러 부채널 공격에 취약하다는 연구결과들이 발표되었다. 이러한 Intel SGX의 취약점을 보완하면서 안전하게 프로그램을 실행하기 위해 제안된 OBFUSCURO[1] 연구는 Oblivious RAM(ORAM)을 함께 활용하여 모든 종류의 부채널 공격을 차단한다. 하지만, ORAM 접근에 따른 성능 오버헤드가 굉장히 크다는 단점이 존재한다.

본 논문에서는 Intel SGX와 ORAM을 활용한 OBFUSCURO[1] 연구에 Intel TSX를 적용시켜 높은 보안성을 유지하면서 ORAM 접근 횟수를 줄여 성능을 개선시키는 방법을 제안한다.

주요어 : 블랙박스 실행, Intel SGX, Intel TSX, ORAM
학 번 : 2019-29139

목 차

표 목 차

그 림 목 차

제 1 장 서론

4차 산업혁명 시대가 온 현대 사회에서 빅데이터, IoT, 인공지능 서비스들이 클라우드 컴퓨팅을 적극적으로 활용하고 있다.[2] 하지만, 클라우드 내에 수집된 다양한 사용자들의 민감한 데이터들이 암호화되지 않은 채 처리되어지고 있어 연산 도중 데이터가 유추될 수 있는 부채널 공격(Side-channel Attack)을 받을 수 있고, 클라우드 관리자들 중 악의적인 의도로 접근하여 데이터를 변조하거나 탈취하는 내부자 위협(Insider Thread)이 있을 수 있다. 이러한 상황이 증가함에 따라 미국과 유럽에서는 개인정보보호관련 법을 제정하는 등 개인정보보호에 대한 국제적인 관심은 나날이 높아지고 있다.

따라서, 클라우드와 같은 원격계산환경에서 민감한 데이터들이 안전하게 처리되어지기 위해 Intel에서는 SGX(Software Guard Extensions) 기술을 개발하였다. SGX는 신뢰실행환경(Trusted Execution Environment, TEE)인 Encalve를 제공한다.[3] 신뢰실행환경은 일반적으로 컴퓨터 시스템의 CPU, 메모리, 저장공간 및 I/O 등 풍부한 자원을 가지는 일반실행환경(Rich Execution Environment, REE)와는 대비되는 개념으로, REE와는 독립된 영역을 갖게 된다. 때문에, Intel SGX는 Enclave라는 독립된 신뢰실행환경 안에서 REE의 모든 권한을 가지는 관리자로부터 기밀성과 무결성을 보장하며 프로그램을 실행하게 되는데, 이를 블랙박스 실행이라고 한다.

그러나, 최근 Intel SGX가 메모리 접근 패턴에 기반한 여러 부채널 공격에 취약하다는 연구결과들이 발표되었다.[4] Intel SGX에 가장 치명적인 부채널 공격에는 페이지 폴트에 기반한 controlled-channel 공격과 캐시에 기반한 캐시 부채널 공격이 존재한다. 이는 Intel SGX의 Enclave 안에서 프로그램이 수행될 때, 프로세서 코어, 캐시, DRAM, MMU와 같은 기존 하드웨어 자원들을 사용하기 때문에 발생하게 되는 취약점인데,

높은 권한을 가지고 있는 OS나 하이퍼바이저를 장악한 공격자가 Enclave 내부에서 수행되고 있는 프로그램이 페이지 폴트 혹은 캐시 미스로 인해 접근하려 하는 메모리 위치를 유추해내는 것이다. 이와 같은 Intel SGX의 취약점을 극복하기 위해 여러 연구들이 진행되었는데, 대표적으로 Oblivious RAM(ORAM)을 활용한 안전한 코드 수행 및 데이터 접근으로 모든 메모리 기반 부채널 공격을 방어한 OBFUSCURO[1] 연구가 있다.

OBFUSCURO[1]는 오픈소스 컴파일러인 LLVM을 통해 기존 SGX 프로그램의 코드를 캐시 라인 크기인 64바이트의 기본 블록으로 나누고, 기본 블록 마다 한 번의 코드와 데이터 접근이 이루어지도록 하였다. 이때 OBFUSCURO[1]의 런타임 라이브러리 함수를 통해 코드용 ORAM과 데이터용 ORAM으로부터 필요한 코드와 데이터를 미리 할당한 메모리 영역인 코드용 Scratchpad와 데이터용 Scratchpad에 64바이트 크기로 적재하여 SGX 프로그램을 수행하게 된다. ORAM으로부터 캐시 라인 하나의 크기만큼만 Scratchpad라는 고정된 주소로만 가져오기 때문에 캐시 부채널 공격으로부터 안전하며, 64바이트로 나누어진 기본 블록 안에서 데이터 접근이 없는 경우 더미 데이터 접근을 하도록 하여 SGX 프로그램의 기본 블록들이 모두 동일한 수행시간을 갖도록 하였다. 또한, SGX 프로그램 이후 더미 코드 블록이 수행되도록 하여, 수행된 전체 코드 블록 수가 특정 횟수를 초과하게 되면 프로그램이 종료된다. 이를 통해 프로그램의 내용과 상관없이 모두 동일한 수행시간을 갖게 되어, 공격자로부터 어떠한 타이밍 정보나 접근 패턴을 유출시키지 않고 프로그램을 안전하게 실행할 수 있다. 그러나 OBFUSCURO[1]는 모든 코드 및 데이터를 캐시 라인 크기로 작게 만들어서 보안성이 높지만, 데이터 읽기 또는 쓰기를 수행하는데 오랜 시간이 걸리는 ORAM을 64바이트의 작은 코드 블록 하나 수행할 때마다 매번 2번씩 접근해야하기 때문에 ORAM 접근 횟수에 따른 성능 오버헤드가 굉장히 크다.

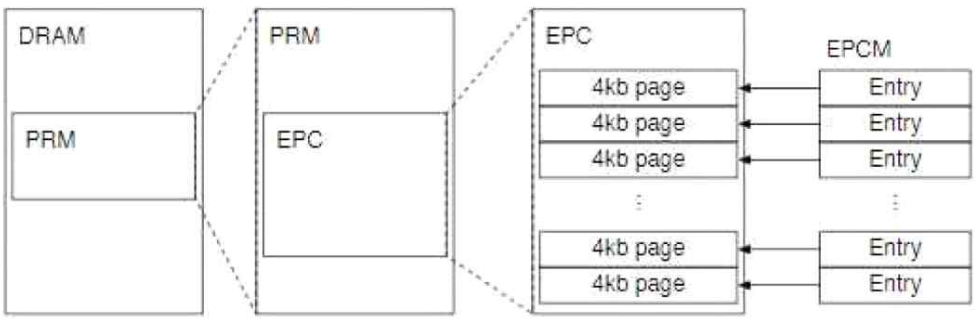
본 논문에서는 OBFUSCURO[1]가 실제 클라우드 컴퓨팅에 적용 가능

하도록 ORAM으로부터 Scratchpad에 적재되는 코드 및 데이터 크기를 늘려 전체적인 ORAM 접근 횟수를 줄임으로써 성능 오버헤드를 줄이고, Intel TSX를 통해 Scratchpad 내의 접근 패턴을 보호하여 기존의 높은 보안성을 유지하는 방법을 제안한다.

제 2 장 배경

제 1 절 Intel SGX(Software Guard Extensions)

Intel SGX는 6세대 프로세서인 '스카이레이크'에서부터 적용된 기술로 코드와 데이터의 기밀성 및 무결성을 보장하기 위해 x86 ISA(Instruction Set Architecture)를 확장한 것이다. SGX는 Enclave라는 독립된 공간에서 운영체제나 하이퍼바이저의 간섭 없이 코드 및 데이터를 수행할 수 있는 하드웨어 기반 신뢰실행환경을 제공한다. 이러한 메모리 영역은 부팅 시에, 프로세서가 DRAM 내부에 128MB 크기의 Enclave Page Cache(EPC) 영역을 미리 확보해 둔다.[3] EPC는 다른 소프트웨어로부터의 접근을 차단하며, 4KB 페이지 당 Enclave Page Cache Map(EPCM)을 통해 페이지 할당 및 해제를 수행한다. Enclave모드로 진입하기 위해서는 EENTER 명령어를 수행하고, 수행 중 Interrupt 혹은 Exception을 통해 Asynchronous Enclave Exit(AEX)가 발생할 경우 프로세서는 현재 수행 상태를 State Save Area(SSA)에 저장하고 ERESURE 명령어를 통해 재개한다. 만약 페이지 폴트가 발생할 경우 OS에 의해 Exception 핸들러가 문제를 해결하게 되므로, AEX가 Exception의 정확한 주소를 숨기더라도 악의적인 OS는 여전히 Exception에 대한 정보를 유추해낼 수 있다.



[그림 2-1] Intel SGX의 EPC 구조

앞서 언급했듯이, 최근 Intel SGX가 여러 부채널 공격에 취약하다는 연구결과가 발표되었는데 가장 대표적인 페이지 폴트와 캐시에 기반한 부채널 공격에 대해 알아보고자 한다. 먼저, 페이지 폴트에 기반한 부채널 공격인 Controlled-channel 공격[5, 6]은 악의적인 OS가 Enclave 프로그램의 페이지 테이블을 조작하여 Enclave 프로그램이 어느 메모리 페이지를 접근하려하는지 알아내는 공격이다. 페이지 테이블 엔트리(PTE)의 reserved bit를 설정하고 페이지 폴트를 감시하게 되는데. 일반 실행환경(REE)이 아닌 신뢰실행환경에서 발생한 페이지 폴트이기에 악의적인 OS는 오직 페이지 프레임 숫자만 알 수 있어 페이지 폴트의 순서들을 분석하여 공격을 시도한다. 다음으로, 캐시 부채널 공격은 캐시 적중과 미스에 따른 접근 지연시간에 대한 차이를 이용하여 희생자의 메모리 접근 패턴을 유추하는 공격이다. 주로 Prime+Probe 공격으로, Prime단계에서는 공격자가 캐시를 본인이 원하는 데이터로 채워 넣은 후 Probe단계에서 동일한 데이터를 접근하고자한다. 만약 희생자가 캐시 미스를 발생시켜 공격자의 데이터를 캐시에서 제거시켰다면, 공격자가 Probe단계에서 접근 지연시간이 길어지는 것을 통해 희생자가 접근한 영역을 유추하여 공격을 시도한다.

제 2 절 Intel TSX(Transactional Synchronization Extensions)

Intel TSX는 하드웨어 트랜잭션 메모리(HTM)을 구현한 것으로 상호 배제를 위한 lock을 얻을 때 오버헤드를 줄이고 동시프로그래밍을 단순화하기 위해 제안되었다. HTM을 사용하게 되면, 스핀락 또는 뮤텍스와 같은 소프트웨어 기반 lock 없이 중요한 영역에서 트랜잭션하게 스레드를 수행할 수 있다. 트랜잭션이 충돌 없이 수행을 마치면, 모든 읽기 및 쓰기는 메모리에 커밋된다. 충돌이 생길 경우, 모든 읽기 및 쓰기는 롤백되며 실제 메모리에 노출되지 않고 예외처리기(abort handler)가 호

출된다. 예외처리기는 트랜잭션을 재시도할지 여부를 결정한다. Intel TSX는 Hardware Lock Elision(HLE)와 Restricted Transactional Memory(RTM) 두가지 인터페이스를 지원하는데, 본 논문에서는 RTM을 사용한다. Intel TSX는 4가지 명령어(XBEGIN, XEND, XABORT, XTEST)를 제공한다. 스레드는 XBEGIN을 통해 트랜잭션을 시작하고 XEND를 통해 종료한다. XABORT를 통해 트랜잭션을 중단하고 XTEST를 통해 현재 트랜잭션이 실행 중인지 판단한다.

```
1  //begin a transaction
2  if (_xbegin() == _XBEGIN_STARTED){
3      //execute a transaction
4      [code]
5      //atomic commit
6      _xend();
7      //end transaction
8  } else{
9      //abort
10 }
11
```

[그림 2-2] Intel TSX의 기본 예제 코드

[그림 2-2]는 TSX의 가장 기본적인 코드로 _xbegin() 함수를 통해 트랜잭션을 시작한다. 이때 _xbegin()이 _XBEGIN_STARTED를 반환한다면 if 구문 안을 수행하고 충돌이 없을 시에 _xend() 함수를 통해 atomic하게 모든 결과값을 커밋하고 트랜잭션을 종료하게 된다. 그러나, 충돌 또는 예외가 발생할 경우 트랜잭션은 롤백되어 else 구문 안의 예외처리부분을 수행하게 된다.

이러한 Intel TSX 기술을 보안 관점에서 살펴보자. 트랜잭션 영역에서 캐시 미스에 의한 abort가 발생하면, 이전까지 수행된 중간결과값들이 L1 캐시에서 L3 캐시(shared cache)로 커밋되지 않고 롤백되어 흔적이 남지 않아 캐시 부채널 공격으로부터 안전하다. 또한, 트랜잭션 영역에서 페이지 폴트에 의한 abort가 발생하면, OS에 알리지 않고 처리하기

때문에 Controlled-channel 공격으로부터 안전하다.

제 3 절 Oblivious RAM

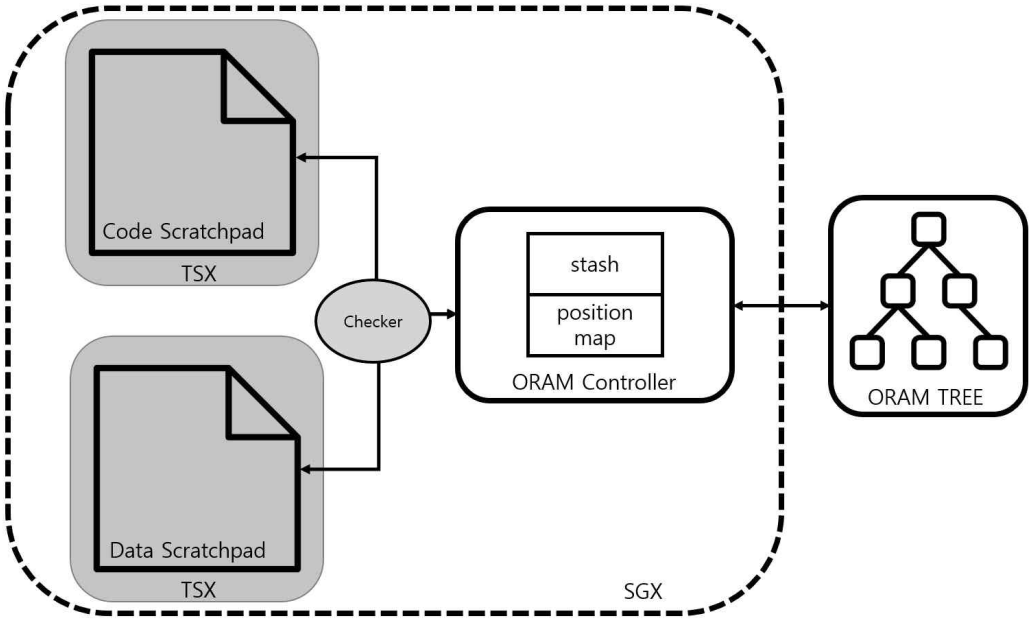
Oblivious RAM(ORAM)은 신뢰할 수 없는 원격 서버에 위치하는 메모리 영역에 안전하게 접근 가능하도록 하는 암호화 기술이다. ORAM은 공격자들로부터 프로그램의 메모리 접근 패턴이 유출되는 것을 막아낸다. 하나의 메모리 주소만을 접근하는 기존과는 다르게, 여러 메모리 주소를 접근하여 접근 패턴을 난독화하며, 접근한 메모리 영역은 랜덤 시드를 이용해 다시 섞고 재암호화한다. 이러한 ORAM은 여러 가지 방법으로 구현되는데, 가장 최악의 성능을 나타내는 ORAM은 하나의 메모리 접근이 발생할 때마다 모든 ORAM의 메모리 접근을 하는 것이다. 본 논문에서는 기존 ORAM에서 개선된 Path ORAM[7]을 사용한다. Path ORAM은 이진 트리에 암호화된 메모리를 저장하며 실제 블록과 더미 블록으로 구성되어 공격자가 구별하지 못하게 한다. Path ORAM은 Position map과 Stash라는 2개의 자료구조를 갖는다. Position map은 정수형 배열로 ORAM 트리에서 실제블록을 찾을 수 있는 경로를 단말노드(leaf node)의 인덱스 정보를 저장하여 나타낸다. Stash는 ORAM 트리에서 원하는 블록을 찾아가는 경로에 속한 모든 블록을 읽어 들인 후 찾고자 했던 블록을 Stash에 저장해두는 버퍼와 같다.

제 3 장 가정 및 공격 모델

본 논문에서는 OBFUSCURO[1]와 동일하게 클라우드 컴퓨팅 서비스의 사용자가 보안에 민감한 정보를 갖고 있는 SGX 프로그램을 클라우드 상에서 원격계산을 하는 상황을 가정한다. 또한, 컴파일러와 CPU만을 신뢰하고 나머지 운영체제, 하이퍼바이저, MMU와 같은 소프트웨어 및 하드웨어를 신뢰하지 않는다. 사용자는 수행하려는 프로그램의 어떠한 정보도 공격자에게 노출시키지 않으려 하고, 공격자는 운영체제나 하이퍼바이저와 같이 높은 권한을 갖는 소프트웨어 혹은 페이지 테이블, 캐시, 분기 예측 유닛을 장악하여 접근 패턴 및 전체 수행시간의 감시를 통해 사용자들의 SGX 프로그램에 대한 정보를 탈취하려 한다. 이외의 Spectre나 Meltdown 공격[10, 11]은 현재 Hyperthreading 기능을 비활성화하여 방어하도록 Intel에서 권고하고 있다.

제 4 장 디자인 및 구현

제 1 절 디자인 개요



[그림 4-1] 디자인 개요

[그림 4-1]과 같이 본 논문의 디자인은 크게 3단계로 나눌 수 있다. 먼저 ORAM으로부터 Scratchpad에 코드 및 데이터 크기를 캐시라인 크기인 64바이트보다 더 많이 가져와 Checker를 통해 전체적인 ORAM 접근 횟수가 줄어들어 성능을 높이는 것이다. 이때 ORAM으로부터 가져오는 크기는 Controlled-channel 공격에 위협을 받을 수 있기 때문에 최대 4KB까지만 증가시켜야 한다. 두 번째로 Scratchpad 내부에서 발생할 수 있는 캐시 부채널 공격을 Intel TSX를 적용하여 방어한다. 마지막으로 ORAM에 접근하는 타이밍 간격을 일정하게 맞춰주기 위해 ORAM을 코드용, 데이터용 2개로 나누지 않고 하나의 ORAM으로 구현하여 공격자로부터 타이밍 정보가 유출되지 않게 하였으며, SGX 프로그램 이후 더

미 코드 블록이 수행되도록 하고 전체 ORAM 접근 횟수가 특정 횟수를 초과하게 되면 프로그램을 종료시킨다. 이를 통해 어떤 프로그램이든지 동일한 전체 수행시간을 갖도록 하여 공격자에게 어떠한 타이밍 정보도 유출하지 않는다.

제 2 절 ORAM 접근 횟수

2장에서 언급했듯이, ORAM에 대한 접근 지연시간은 굉장히 느리다. OBFUSCURO[1]에서는 모든 코드 및 데이터가 64바이트로 작게 나뉘어 ORAM 트리에 저장된다. 그리고 각 코드용 Scratchpad와 데이터용 Scratchpad에 64바이트만큼의 코드와 데이터를 각각의 ORAM으로부터 읽어 들여와 프로그램을 수행하게 되는데, 이는 프로그램의 크기가 클수록 더 많은 ORAM 접근 횟수가 필요하게 되며, 성능 오버헤드가 상당하게 된다. 때문에, 본 논문에서는 128바이트, 256바이트 512바이트, 1024바이트, …, 4096바이트까지 ORAM으로부터 Scratchpad에 가져오는 코드 및 데이터 크기를 증가시키도록 구현하였다. 예를 들어. 코드용 Scratchpad에 128바이트의 코드가 적재된다면, 첫 번째 코드 기본 블록 수행이 끝난 후 두 번째 기본 블록에 접근하려 할 때 먼저 Checker를 통해 다음 코드 블록이 현재 Scratchpad에 존재하는지를 판단한다. Checker는 다음 코드 블록이 현재 Scratchpad에 존재한다면 ORAM으로 접근하지 않고 다음 코드 블록 위치로 점프하도록 하고, 반대의 경우 ORAM 접근을 하도록 구현하였다. 이는 결국 Scratchpad 내부의 지역성을 향상시켜 ORAM 접근횟수를 줄임으로써 전체적인 성능 오버헤드가 줄어들게 된다.

그러나, 연속적인 주소를 갖는 코드 기본 블록들과 데이터는 전역변수가 저장되는 데이터영역과 지역변수가 저장되는 스택영역으로 나뉘는데, 이 두 영역 간의 메모리 위치 차이가 4KB를 초과한다. 따라서 코드용 Scracthpad에서 지역성을 향상시킨 정도만큼 데이터용 Scratchpad에서 지역성이 증가하지 않는다. 매번 스택영역을 접근하는 중간에 데이터

영역도 접근하게 되므로 ORAM 접근 횟수가 크게 줄어들지 않는다.

제 3 절 Intel TSX 적용

Scratchpad에 적재된 코드와 데이터의 크기가 64바이트보다 커졌으므로, 코드 수행 중 접근패턴이 드러나는 취약점이 발생할 수 있다. 본 논문에서는 Intel TSX를 활용해 Scratchpad 내부를 트랜잭션하게 수행되도록 한다. Scratchpad에서 코드가 수행되는 동안 외부로부터 캐시 미스가 발생하게 되면, Abort가 발생하여 L1 캐시에 존재한 값들이 L3 캐시로 커밋되지 않고 롤백되어 캐시 부채널 공격을 막게 된다. TSX를 통해 Scratchpad 내부의 접근패턴이 드러날 위협이 없으므로, LLVM 컴파일러에서 생성하는 더미 데이터 접근을 무시할 수 있다. LLVM 수정과 어셈블리 코드 삽입을 통하여, 트랜잭션 영역은 매번 ORAM 컨트롤러에서 Scratchpad로 점프하기 이전에 XBEGIN을 삽입하고, ORAM 컨트롤러에서 ORAM에 접근하기 이전에 XEND를 삽입하도록 구현하였다. 기존의 nop으로 채워져 있던 기본 블록에 TSX 명령어를 대신 삽입한 것이기 때문에, 명령어 추가에 따른 성능 오버헤드는 발생하지 않는다.

제 4 절 수행시간 정규화

Scratchpad 내의 지역성이 증가함에 따라, 코드용 ORAM과 데이터용 ORAM에 대한 접근이 일정한 시간 간격으로 접근하지 않게 된다. 코드용 ORAM에 처음 접근하여 128바이트 크기의 코드를 코드용 Scratchpad에 적재한 다음 데이터용 ORAM에 2번 접근한 후에 다시 코드용 ORAM에 접근하게 될 수도 있고, 데이터용 ORAM에 1번 접근한 후에 코드용 ORAM에 접근할 수도 있다. ORAM 접근에 따른 지연시간이 기본 블록 수행시간에 비해 굉장히 크고 데이터용 Scratchpad는 스택영역으로 인해 지역성이 낮기 때문에, 코드용 ORAM에 대한 접근 간격은 데이터용 ORAM을 몇 번 접근하는가에 따라 계속해서 달라진다.

물론 데이터용 ORAM에 대한 접근 간격도 마찬가지이다. 이는 공격자에게 타이밍 정보를 유출시킬 수 있게 되므로, ORAM을 하나로 통합하여 구현해야한다. ORAM을 하나로 통합하게 되면 코드와 데이터 구별 없이 ORAM의 입장에서는 일정한 간격으로 접근되기 때문에 공격자로부터 타이밍 정보를 유출시키지 않을 수 있다.

또한, 어떤 프로그램이든지 전체 수행시간을 같게 하여 공격자로부터 타이밍 정보를 숨기기 위해, 수행하려 하는 SGX 프로그램이 끝난 뒤에는 더미 프로그램이 수행되도록 하여 전체 프로그램이 미리 설정해둔 ORAM 접근 횟수에 도달하게 되면 전체 프로그램 수행을 종료시키도록 한다. ORAM 접근 횟수에 따른 지연시간이 프로그램 수행시간에 가장 지배적이기 때문에 이를 활용하여 전체 프로그램 수행시간을 정규화하였다. SGXv2가 적용된 프로세서에서는 rdtsc()를 사용하여 더 정확하게 프로그램 수행시간을 정규화시킬 수 있을 것이다.

제 5 장 실험 결과

제 1 절 실험 환경

실험을 진행하기 위해 사용된 PC의 스펙은 Intel(R) Core(TM) i5-8500 CPU @3.00GHz, 16GB RAM (128MB for EPC)을 사용하였고, Ubuntu 16.04.5 LTS with Linux 4.15.0-122-generic 64-bit 시스템과 Intel에서 제공하는 SGX SDK와 SGX driver를 사용하였다. 사용한 프로세서의 Hyperthreading 기능을 지원하지 않는다. 또한, Intel TSX 기능이 비활성화되도록 마이크로코드 패치가 되어있어 마이크로코드 롤백을 통해 TSX 기능을 활성화시켜 실험을 진행하였다.

제 2 절 성능 오버헤드

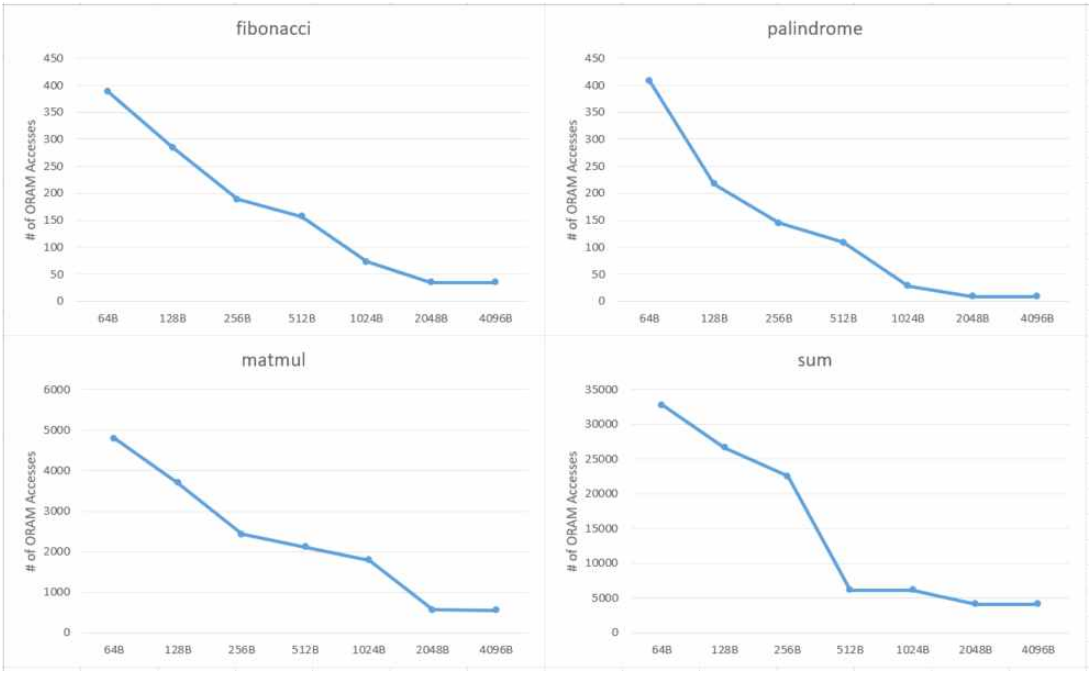
fibonacci	Exec_Time(ns)	ORAM Access
64B	47,443,310	388
128B	34,304,331	285
256B	23,567,056	189
512B	20,125,885	157
1024B	10,479,906	73
2048B	7,026,607	35
4096B	8,091,581	35

palindrome	Exec_Time(ns)	ORAM Access
64B	49,184,136	408
128B	26,454,296	229
256B	17,835,655	145
512B	13,846,976	109
1024B	4,364,282	29
2048B	4,027,765	9
4096B	3,770,137	9

matmul	Exec_Time(ns)	ORAM Access
64B	565,231,005	4792
128B	434,022,994	3705
256B	292,644,734	2437
512B	261,413,973	2113
1024B	230,243,228	1789
2048B	99,969,222	563
4096B	111,935,135	551

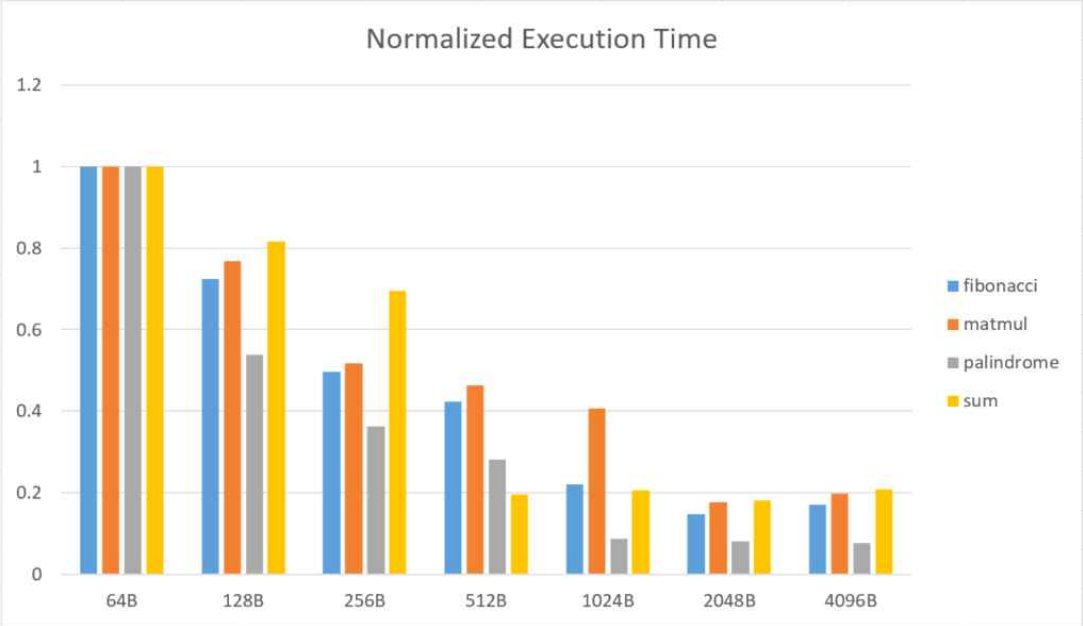
sum	Exec_Time(ns)	ORAM Access
64B	3,788,325,808	32816
128B	3,088,807,722	26657
256B	2,634,864,515	22549
512B	746,499,398	6177
1024B	783,321,565	6161
2048B	688,024,544	4105
4096B	794,384,299	4105

[표 5-1] 성능 결과 표



[그림 5-1] ORAM 접근 횟수 그래프

성능은 전체 수행시간을 기준으로 비교하였으며, 4개의 벤치마크(Fibonacci, Matrix Multiplication, Palindrome, Sum)를 통해 결과를 나타내었다. 또한, 벤치마크 프로그램 코드가 다 수행된 후에 더미 프로그램은 수행되지 않도록 하여 벤치마크 프로그램 코드만의 성능 개선 효과를 확인하였다. 실험은 Scratchpad에 적재하는 코드 및 데이터의 크기를 64바이트, 128바이트, 256바이트, 512바이트, 1024바이트, 2048바이트, 4096바이트 일 때 전체 수행시간을 측정하였으며, 128바이트 이후부터는 본 논문에서 구현한 디자인으로 실험이 진행되었다.



[그림 5-2] 성능 비교 그래프

4096바이트까지 늘려가며 실험했을 때, [표 5-1]과 [그림 5-1]에서 볼 수 있듯이 2048바이트까지만 ORAM 접근 횟수가 감소하게 되고, 4096바이트의 경우 2048바이트의 경우와 동일한 ORAM 접근 횟수를 갖는다. 4096바이트의 경우 같은 ORAM 접근 횟수를 가지더라도 더 많은 양의 데이터를 가져와야 하므로, 2048바이트보다 오히려 성능이 하락한 것을 확인할 수 있다. [그림 5-2]는 64바이트의 경우를 기준으로 수행시간을 정규화하여 나타낸 성능 비교 그래프이다. 4개의 벤치마크 중에서 성능 향상을 가장 많이 나타낸 벤치마크는 palindrome으로 약 13배이고 가장 적은 성능 향상을 나타낸 벤치마크는 matrix multiplication으로 약 5배이다. palindrome 벤치마크의 경우 전역변수만을 사용하기 때문에, 데이터영역과 스택영역을 번갈아 가면서 접근할 필요가 없어 데이터용 Scratchpad 내의 지역성이 증가해 높은 성능향상을 보인다. 반면에, matrix multiplication 벤치마크의 경우 데이터 접근이 굉장히 많고, 데이터영역과 스택영역을 모두 접근해야 하므로 성능향상 정도가 낮다.

실험에 사용된 벤치마크 자체가 굉장히 작은 크기를 지니고 있어 큰 성능이득을 보지 못하였는데, 추후에 nbench와 같은 더 큰 벤치마크를 사용하여 실험해본다면 더 많은 성능이득을 얻을 수 있을 것으로 예상된다. 또한, FPGA와 같은 외부 하드웨어를 적용시킨다면, ORAM으로 인한 근본적인 성능하락 문제를 개선 시킬 수 있을 것으로 보인다.

제 6 장 결론

본 논문은 Intel TSX를 활용하여 Intel SGX의 부채널 공격에 대한 취약점을 완벽하게 극복한 블랙박스 실행을 구현하였으며, ORAM 접근 횟수를 줄여 기존 연구의 성능을 개선시켰다. Intel TSX가 적용되어 코드가 수행되는 동안 악의적인 OS로부터의 페이지 폴트 및 캐시 부채널 공격으로부터 안전하게 보호한다. 그리고 ORAM으로부터 가져오는 코드 및 데이터 크기가 커져 Scratchpad 내의 지역성이 증가하고 ORAM 접근 횟수가 줄어듦에 따라 전체적인 성능이 향상되는 것을 실험을 통해 확인하였다. 이를 통해 클라우드 컴퓨팅 서비스가 적용 가능한 성능으로 안전한 블랙박스 실행을 제공할 수 있다는 가능성을 제시하였으며, 나아가 FPGA와 같은 외부 하드웨어를 접목한다면 더 많은 성능 이득을 기대할 수 있을 것이다.

참 고 문 헌

[1] A. Ahmad, B. Joe, Y. Xiao, Y. Zhang, I. Shin, and B. Lee. OBFUSCURO: A Commodity Obfuscation Engine on Intel SGX. In Proceedings of the 2019 Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA, Feb. 2019.

[2] T. Dillon, C. Wu, and E. Chang, “Cloud Computing: Issues and Challenges,” in IEEE International Conference on Advanced Information Networking and Applications, 2010.

[3] V. Costan and S. Devadas. "Intel SGX Explained." IACR Cryptology ePrint Archive, Tech. Rep., 2016.

[4] O. Oleksenko, B. Trach, R. Krahn, M. Silberstein, and C. Fetzer. Varys: Protecting SGX enclaves from practical side-channel attacks. In 2018 USENIX Annual Technical Conference (USENIX ATC), 2018.

[5] Y. Xu, W. Cui, and M. Peinado. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In Proceedings of the 36th IEEE Symposium on Security and Privacy (Oakland), San Jose, CA, May 2015.

[6] J. Van Bulck, N. Weichbrodt, R. Kapitza, F. Piessens, and R. Strackx. Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution. In Proceedings of the 26th USENIX Security Symposium (Security), Vancouver, BC, Aug. 2017.

[7] E. Stefanov, M. van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas, “Path oram: An extremely simple oblivious ram protocol,” in Proceedings of the 20th ACM Conference on Computer and Communications Security (CCS), Berlin,

Germany, Oct. 2013.

[8] M. Herlihy and J. Moss, “Transactional memory: Architectural support for lock-free data structures,” in Proceedings of the 20th ACM/IEEE International Symposium on Computer Architecture (ISCA), San Diego, CA, USA, 1993.

[9] M.-W. Shih, S. Lee, T. Kim, and M. Peinado. T-SGX: Eradicating Controlled-Channel Attacks Against Enclave Programs. In Proceedings of the 2017 Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA, Feb. 2017.

[10] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. H. Lai, “SGXPECTRE Attacks: Stealing Intel Secrets from SGX Enclaves via Speculative Execution”, IEEE EuroS&P, 2019

[11] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx. Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution. In Proceedings of the 27th USENIX Security Symposium (Security), Baltimore, MD, Aug. 2018

Abstract

A Study on ORAM Acceleration Technique using Intel TSX

Jiwon Chang

Electrical and Computer Engineering

The Graduate School

Seoul National University

Although the privacy issue is becoming more and more important as the era of cloud computing and big data, security problems are likely to occur because users' data are computed in a decrypted form within the cloud service. Accordingly, Intel SGX ensures program confidentiality and integrity through Encalve, a hardware-based trust execution environment such as a black box, to secure users' sensitive data. However, researchers have identified that the Intel SGX is vulnerable to side-channel attacks which are based on timing and access patterns. The OBFUSCURO proposed program obfuscation which protects the SGX enclave against information leakage through all side-channels. But, because of a number of ORAM accesses, its performance overhead is huge.

In this paper, we applied Intel TSX to OBFUSCURO, which utilizes

Intel SGX and ORAM, for improving the performance by reducing the number of ORAM accesses while maintaining the security.

keywords : Black Box Execution, Intel SGX, Intel TSX, ORAM
Student Number : 2019-29139