모호한 메모리 접근 알고리즘의 클라우드 서비스에의 적용

용인호*
*성균관학교 컴퓨터공학과
yongih12@skku.edu

Applying Oblivious Memory Primitive to Real World Cloud Services

In-Ho Yong*
*Dept. of Computer Engineering, Sungkyunkwan University

요 약

ZeroTrace는 클라우드 서버의 메모리 접근을 모호하게 하기 위해 개발되었으나, 실제 클라우드 서비스에 적용되기 위해서 해결되어야 할 두 가지 문제점이 발견되었다. 이번 연구에서는 이를 해결한 CloudZeroTrace를 제시한다. ZeroTrace 인터페이스를 호출하는 부분에 뮤텍스 락 매커니즘이 적용되었으며, IDmap이라는 새로운 자료구조를 통해 다양한 자료형을 통한 데이터의 인덱싱 기능이 추가되었다.

1. 서론

고가의 대용량 저장소 또는 강력한 연산 장치를 구매하지 않고도 필요한 때에 일정량의 비용을 지불하고 사용할 수 있다는 장점 때문에 클라우드 컴퓨팅은 IT 서비스의 한 부문으로서 2020년 기준 300만 U.S. 달러 이상의 시장규모를 이루게 되었다[1]. 클라우드 컴퓨팅이 대중화되면서 클라우드 컴퓨팅 보안도 함께 이슈화되었다. 클라우드 서비스 제공자는 클라우드 서버에 대한 모든 권한을 가지고 있기 때문에 클라우드 서비스 제공자가 악의적인 경우 매우 간단하게 사용자의 정보가 유출될 수 있다. 이에 대한 해결책으로 Intel은 새로운 하드웨어 명령어 집합인 Software Guard Extension (Intel SGX)[2]를 발표하였다. 하지만 최근 Intel SGX에 대한 여러 부채널 공격들[3, 4, 5]이 발표되어 단순히 Intel SGX를 클라우드 서비스에 적용하는 것만으로는 클라우드 서버에서 사용자의 데이터가 안전하게 보호되지 않는다는 것이 밝혀졌다. Zero Trace[6]는 이러한 상황에서 Intel SGX에 대한 부채널 공격들을 방어하고자 모호한 메모리 접근 알고리즘인 ORAM 알고리즘들[7, 8]을 메모리의 크기가 제한된 Intel SGX에 소프트웨어적으로 구현하였다.

ZeroTrace는 클라우드 서버의 메모리 접근을 모호하게 하기 위해 개발되었으나, 소스코드를 분석한 결과 실제 클라우드 서비스에 적용되기 위해서 해결되어야 할 두 가지 문제점이 발견되었다.

P1. 스레드 동기화. 한 명의 사용자가 여러 개의 스레드를 통해 동시에 메모리를 접근하는 경우에 대한 고려가 되어있지 않음.

P2. 정수로 제한된 인덱스. 현실 세계를 데이터로 모델링하게 되면 다양한 자료형을 통해 데이터들이 인덱싱될 수 있지만, 현재 ZeroTrace에서는 데이터에 대해 정수형의 인덱스만을 제공함.

따라서 이번 연구에서는 P1, P2의 해결책을 ZeroTrace에 추가한 버전인 CloudZeroTrace를 제시한다.

2. 배경지식

2-1. Intel SGX

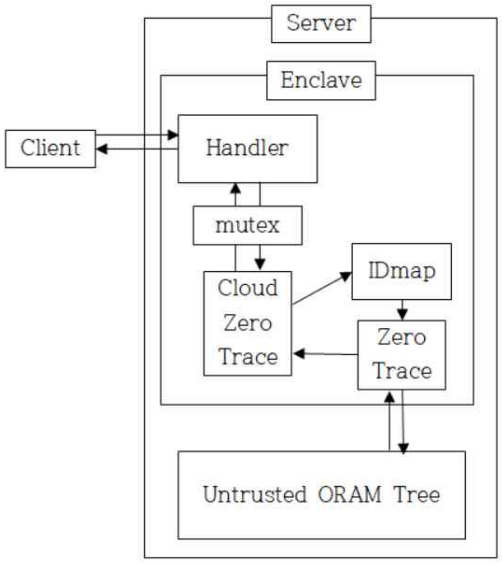
Intel SGX[2]는 메모리의 일부를 PRM(Processor Reserved Memory)으로 분리하여 Enclave단위로 관리하며, Enclave 초기화 과정에서 설정된 키가 없으면 시스템에 대한 모든 권한을 가지고 있더라도 해당 메모리에 접근할 수 없다. 또한 제한된 메모리의

크기 때문에 어플리케이션이 Intel SGX를 지원하기 위해서는 어플리케이션의 보안에 민감한 최소한의 부분(trusted code)만 Enclave 내부에 구현하고 나머지 부분(untrusted code)은 Enclave 외부에 구현하여야 한다. 두 부분간의 인터페이스는 'ecall'과 'ocall'를 통해 직접 정의해주어야 한다.

2-2. ZeroTrace

ZeroTrace[6]는 모호한 메모리 접근 알고리즘인 ORAM 알고리즘을 Intel SGX 내부에 구현한 라이브러리이다. ORAM의 모호한 메모리 접근을 위한 자료구조인 Position map과 Stash만 보안에 민감한 부분으로 정의하고 나머지 부분은 Enclave 외부에 구현되어 있다. 또한 x86 CMOV 명령어를 사용함으로써 소프트웨어적인 모호한 메모리 컨트롤러 알고리즘을 구현하였다.

3. CloudZeroTrace



(그림 1) CloudZeroTrace 구조.

CloudZeroTrace의 구조는 전체적으로 ZeroTrace의 구조와 동일하다. 서론에서 언급된 P1을 해결하기 위해 ZeroTrace 인터페이스를 호출하는 부분에 뮤텍스 락 매커니즘이 적용되었으며, P2를 해결하기 위해 IDmap이라는 새로운 자료구조가 추가되었다. IDmap은 정수 이외의 자료형을 인덱스로 사용할 수 있으면서도 ZeroTrace 인터페이스를 변경 없이 사용하기 위해 추가되었다. IDmap에는 데이터의 인덱스와 ORAM의 데이터 저장 단위인 블록의 인덱스 간의 매칭 정보가 저장되어 있다. ZeroTrace자체를 변경하여 블록의 인덱스의 자료형을 정수 이외의 자료형도 가능하도록 변경하게 되면, ZeroTrace가 보증하는 보안성이 깨지게 되고 변경된 시스템에 대한 새로운 보안성을 검증해야 하기 때문에 효율적인 설계를 위해 새로운 자료구조를 추가하는 방식으로 구현이 진행되었다. 이 때, 데이터의 인덱스와 블록 ID 간의 매칭 정보는 공격자가 알더라도 블록 ID를 통해 데이터를 얻어내는 것은 ZeroTrace의 보안성에 의해 불가능하기 때문에 보안성을 유지해야 할 대상이 아니다. 따라서 IDmap의 조회에 있어서 보안성을 유지하기 위해 성능을 크게 저하시킬 수 있는 선형 스캔을 적용하지 않았다. 또한 IDmap은 Enclave 내부에 저장되는데, 이는 IDmap을 암호화하여 SGX 외부에 저장하게 되면 암, 복호화와 ocall에 의한 성능 저하가 예상되기 때문이다.

4. 결과분석

전체적인 실험은 4433 포트를 통해서 http 요청을 대기하는 CloudZeroTrace 프로세스를 실행시킨 뒤, localhost:4433으로 curl을 통해 요청을 보내는 방식으로 진행되었다. 실험에 사용된 코드는 https://github.com/Yong-inho/Graduation_Thesis에서 확인할 수 있다. 실험을 통해 사용자가 동시에 여러 개의 요청을 보내는 경우를 잘 처리하는 것이 확인되었으며, 읽기 후 쓰기, 쓰기 후 쓰기와 같은 동시에 동일한 메모리를 접근하는 경우 발생하는 동기화 문제도 발생하지 않는 것으로 확인되었다. 데이터의 인덱스의 크기는 빌드 옵션으로 설정할 수 있으며, 다양한 크기의 인덱스에 대해서도 실험을 통해 데이터를 문제없이 저장하고 읽어올 수 있음이 확인되었다.

5. 결론

이번 연구에서는 클라우드 서버에 아웃소싱된 데이터를 저장하는 메모리의 접근 패턴을 모호하게 하기위해 개발된 ZeroTrace를 실제로 클라우드 서버에 적용함에 있어서 부족한 부분을 분석하였다. 이를 통해 동시에 여러 개의 스레드를 통해 메모리에 접근하는 경우에 대한 동기화 문제와 데이터의 인덱싱이 정수로밖에 이루어지지 않은 문제를 발견하였다. 따라서 이를 해결하기 위해 CloudZeroTrace라는 새로운 인터페이스를 추가하고 이 인터페이스를 통해서 발견된 문제들을 해결하였다. 그 결과 메모리 접근의 모호성을 유지하면서도 사용자가 동시에 여

러 번의 메모리 접근을 요청하는 경우와 정수 이외의 자료형에 의해 데이터가 인덱싱되어야 하는 경우를 적절이 처리할 수 있었다.

참고문헌

[1] K. Mlitz, "Cloud Computing - Statistics & Facts," statista, 2021, [Online] https://www.statista.com/topics/1695/cloud-computing.

[2] S. JOHNSON, V. SCARLATA, C. ROZAS, E. BRICKELL and F. MCKEEN, "Intel® software guard extensions: Epid provisioning and attestation services.", White Paper 1, (2016), 1 - 10.

[3] J. V. Bulck, F. Piessens, and R. Strackx, "SGX-Step: A Practical Attack Framework for PRecise Enclave Excution Control," SysTEX'17: Proceedings of the 2nd Workshop on System Software for Trusted Execution, 2017, 1-6.

[4] J. V. Bulck, M. Minkin, O. Weisse, D. Genkin, B. kasicki, 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," 27th {USENIX} Security Symposium ({USENIX} Security 18), 991 - 1008.

[5] S. Lee, M. Shih, P. Gera, T. Kim, H. Kim and M. Peinado, "Inferring Fine-grained Control Flow Inside {SGX} Enclaves with Branch Shadowing," 26th {USENIX} Security Symposium ({USENIX} Security 17), 557-574.

[6] S. Sasy, S. Gorbunov, and C. W. Feltcher, "ZeroTrace: Oblivious Memory Primitives from Intel SGX," In Network and Distributed System Security Symposium (NDSS), 2018.

[7] E. Stefanov, M. V. Dijk, E. Shi, T.-H. H. Chan, C. Fletcher, L. Ren, X. Yu, and S. Devadas, "Path ORAM: An Extremely Simple Oblivious RAM Protocol," J. ACM 65, 4, Article 18 (August 2018), 26 pages.

[8] X. Wang, H. Chan, and E. Shi, "Circuit oram: On tightness of the goldreich-ostrovsky lower bound," in Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security, 2015, pp. 850 - 861.