ARM 기밀 연산 아키텍처에서의 안전하고 효율적인 메모리 공유

유준승[1], 백윤흥[2]
[1]서울대학교 전기정보공학과 석박통합과정, 반도체공동연구소
[2]서울대학교 전기정보공학과 교수, 반도체공동연구소
jsyou@sor.snu.ac.kr, ypaek@snu.ac.kr

Secure and Efficient Memory Sharing on ARM Confidential Compute Architecture

Junseung You[1], Yunheung Paek[1]
[1]Dept. of Electrical and Computer Engineering and Inter-University Semiconductor Research Center (ISRC), Seoul National University

요 약

원격 컴퓨팅 환경에서 오프로딩된 사용자의 코드와 데이터를 악의적인 내부 위협자(클라우드 운영 체제 등)으로부터 안전하게 지켜주는 하드웨어 신뢰실행환경은 보안성을 위하여 사용되는 메모리 물리 주소가 하나의 실행환경에 귀속되는 공간적 격리(spatial isolation) 모델을 사용한다. 허나 이러한 메모리 모델은 상호작용하는 신뢰실행환경 프로그램들 사이 메모리 공유를 허락하지 않으며, 이는 성능 및 기존 어플리케이션과의 호환성에서의 문제를 야기한다. 본 논문에서는 최근 ARM사에서 발표된 새로운 신뢰실행환경인 기밀 컴퓨팅 아키텍처를 분석하여 메모리 공유 가능성을 파악하고, 공유가 단순히 허용되어 있을 때의 보안 문제와 이에 대한 기본적인 해결책 및 그 한계점을 제시한다.

1. 서론

클라우드 컴퓨팅의 수요와 사용이 증가함에 따라 원격 환경에 오프로딩된 사용자의 코드 및 데이터에 대한 무결성과 기밀성을 보장하기 위한 핵심 보안 기술로 신뢰실행환경이 각광 받고 있다. 이에 따라 프로세서 제조사들(인텔, AMD 등)에서 서로 다른 하드웨어 기반 신뢰실행환경 기술들을 제공하고 있다. 각 기술들의 세부 메커니즘은 상이하지만, 이들은 사용되는 물리 메모리를 하나의 실행환경(이하 인클레이브(enclave))에 귀속시키는 공간적 격리 메모리 모델을 기반으로 한다. 예를 들어, 인텔 사의 신뢰실행환경 기술인 Software Guard Extensions (SGX)의 경우 Enclave Page Cache Map Entrty (EPCM)라는 물리 페이지 별 메타 데이터를 관리하여 각 페이지의 주인 인클레이브 아이디를 저장한다 [1]. 메모리 접근 시, 하드웨어는 EPCM 메타 데이터를 읽어 현재 접근을 요청하는 인클레이브가 해당 메모리 페이지의 주인임이 확인될 시에만 접근을 허용한다.

공간적 격리 메모리 모델은 운영체제 및 타 어플리케이션들로부터의 인클레이브 접근을 원천적으로 차단하지만, 이에 따라 인클레이브 메모리의 공유를 허용하지 않음으로써 안에서 실행되는 어플리케이션의 성능과 호환성을 크게 떨어뜨린다. 예를 들어 인클레이브 내에서 머신 러닝 모델을 제공하는 프레임워크의 경우, 모델을 인클레이브 간 공유하지 못하기 때문에 서로 다른 유저의 입력과 결과를 제공하기 위해 유저 별로 인클레이브 및 모델을 관리해야 하며, 이로 인해 더 많은 메모리와 결과 반환 시간을 필요로 한다.

이러한 기존 신뢰실행환경의 한계점을 극복하기 위해 인클레이브 간 메모리 공유를 안전하게 허용하고자 하는 연구들이 진행되었으나[2, 3], 프로토타입이 RISC-V 기반으로 설계되어 상업적/실용적인 기술로의 발전으로는 한계가 있었다. 이에 따라, 본 논문에서는 최근 ARM사에서 발표한 ARMv9 프로세서들에서부터 제공될 신뢰실행환경 기술인 기밀 연산 아키텍처(Confidetial Compute Architecture)에서 인클레이브 간 메모리 공유가 가능한지 분석한다. 이와 더불어 메모리 공유가 단순하게 허용되었을 때의 보안 문제를 파악하고, 이에 대한 해결책 및 그 한계점을 제시한다.

2. 배경이론

2-1. ARM Exception Levels (ELs)

ARM에서의 프로그램은 4가지 중 하나의 Exception Level(이하 EL)을 지닌다. EL은 프로그램의 실행 권한(privilege)를 나타내며, 더 높은 실행 권한을 가질수록 더 많은 시스템 자원 및 설정을 관리할 수 있다. 기본적으로 EL0에서는 유저 어플리케이션이 실행되며, EL1에서는 커널/운영체제, EL2에서는 하이퍼바이저, 그리고 EL3에서는 펌웨어 및 베어메탈 어셈블리/코드 등이 실행된다.

2-2. ARM Security State

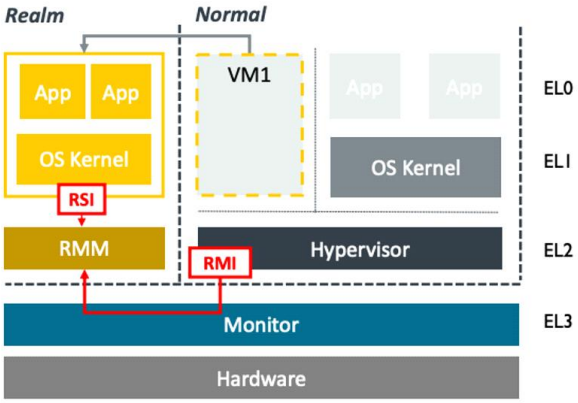
ARM Security State(이하 보안 상태)는 ARMv8까지에서는 Normal State과 Secure State으로 나뉜다. Normal State에서 실행되는 일반 어플리케이션 및 운영체제는 Secure State에서 사용되는 Secure 메모리를 접근할 수 없다. 이에 따라 Secure State에서는 Trusted OS 및 security-critical 어플리케이션이 실행된다. ARM은 TrustZone 기술을 활용하여 두 상태를 분리 및 관리하며, 두 상태 간 변환은 EL3에서 실행되는 Secure Monitor라는 시스템 부팅 시 보안성이 검증되는 요소를 통해 이루어진다.

2-3. ARM 가상화

ARM은 2단계 주소변환을 통해 하드웨어 기반 가상화를 지원한다. 기존에 메모리 가상주소가 운영체제의 페이지 테이블을 통해 바로 물리주소로 변환되었던 것과는 다르게, ARM 가상화 기능을 사용하면 VM OS의 페이지 테이블은 가상 주소를 중간물리주소(Intermediate Physical Address, 이하 IPA)로 변환한다. 2단계(Stage-2) 주소변환은 EL2에서 실행되는 하이퍼바이저가 관리하는 2단계 페이지 테이블을 통해 이루어지며, IPA를 물리주소로 변환한다.

2-4. ARM 기밀 연산 아키텍처

ARM 기밀 연산 아키텍처(이하 CCA)는 ARM사에서 발표한 새로운 신뢰실행환경 기술로 [4], 기존 ARM TrustZone [5] 기술을 보완하여 ARMv9.2 이후 프로세서들부터 이를 지원한다. CCA는 기본적으로 새로운 보안 상태(state)인 Realm을 제공하며, 가상화를 기반으로 인클레이브를 가상 머신(Virtual Machine, 이하 VM) 단위로 생성 및 관리한다. 즉, CCA에서 인클레이브는 Realm VM으로 제공되며 그 안의 내용에 대한 무결성과 기밀성을 보장한다. CCA를 구성하는 시스템 요소는 (그림 1)과 같다.



(그림 1) ARM CCA 시스템 구성 요소

ARM RMM. Realm VM의 실행을 지원하기 위하여 CCA는 Realm Management Monitor(RMM)이라는 소프트웨어를 제공한다. RMM은 Realm 보안 상태의 EL2에서 실행되며, 2단계 페이지 테이블을 통해 Realm VM간을 격리한다.

ARM RME. CCA는 Realm Management Extension(이하 RME)라는 하드웨어 추가 기능을 통해 구현된다. RME는 기존 TrustZone 기능을 연장하여 Realm 보안 상태를 제공함과 동시에 EL3의 시스템 요소들이 Root 보안 상태에서 실행되게끔 지원한다. 이와 더불어 메모리 접근 시 아래 설명될 추가 보안 체크를 통해 보안 상태 간 격리를 제공한다.

Granule Protection Check. RME는 메모리 접근이 유효한(허용되어도 괜찮은) 접근인지 Granule Protection Check(이하 GPC)를 통해 확인한다. 모든 물리페이지 주소공간(Physical Address Space, 이하 PAS)은 Normal, Secure, Realm, Root 중 하나의 보안 상태로 정해지며, 메모리 접근이 요청되었을 때의 보안 상태에 따라 GPC는 각 주소공간에 속해 있는 메모리 접근 허용 또는 불허한다. 보안 상태에 따른 접근 가능 PAS는 <표 1>과 같다.

Granule Protection Table. CCA는 GPC를 위해 각 물리메모리 페이지가 속해 있는 보안 상태를 관리하는 자료구조인 Granule Protection Table(이하 GPT)를 관리한다. GPT는 Root 보안 상태 EL3에서 실행되는 Secure Monitor을 통해서만 관리된다.

보안 상태	Normal PAS	Secure	Realm	Root
Normal	√	×	×	×
Secure	√	√	×	×
Realm	√	×	√	×
Root	√	√	√	√

<표 1> 보안 상태에 따른 접근 가능 PAS

3. ARM CCA에서의 메모리 공유

3-1. ARM CCA에서의 메모리 공유 가능성

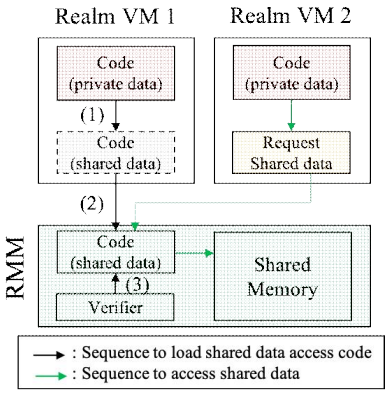
ARM CCA는 기존 신뢰실행환경 기술(Intel SGX, AMD Secure Encrypted Virtualization 등)과 다른 메모리 격리 모델을 사용한다. 보다 구체적으로, 다른 신뢰실행환경 기술은 물리메모리 페이지를 하나의 인클레이브에 귀속 시킴에 반면, CCA는 페이지를 보안 상태에 귀속 시킨다. 다시 말해, CCA는 하드웨어적으로 RME를 통해 Normal State에서 실행되는 악의적인 어플리케이션/OS/하이퍼바이저가 Realm VM의 메모리를 접근하는 것을 방어하지만, Realm State에서 실행되는 여러 Realm VM들(인클레이브들) 간의 격리는 하드웨어가 아닌 RMM 소프트웨어가 관리하는 2단계 페이지 테이블을 통해 실현한다. 이에 따라 Realm VM들은 기존의 일반적인 VM들이 하이퍼바이저를 통해 2단계 페이지 테이블에서 서로 다른 VM의 IPA를 같은 PA로 매핑함을 통해 메모리를 공유할 수 있는 것과 같이 RMM, 즉, Realm State 하이퍼바이저를 통해 Realm VM 간의 메모리 공유를 실현할 수 있다.

3-2. 메모리 공유에 따른 보안 문제

2단계 페이지 테이블에서 서로 다른 Realm VM의 IPA를 같은 PA로 매핑함을 통해 메모리를 공유하는 것은 가능하지만, 안타깝게도 이러한 공유 메커니즘은 보안 취약점을 지닌다. 보다 구체적으로, 이러한 메커니즘은 Realm VM간 공유된 메모리에 대한 공간적 격리는 지원하지만, 시간적 격리는 지원하지 못한다. 공간적 격리는 공유되는 메모리가 어떠한 Realm VM에서 접근 가능한지를 관리할 수 있음에 따라 실현되는데, 이는 2단계 페이지 테이블을 통해 자연스럽게 해결된다. N개의 Realm VM이 있을 때 서로 메모리를 공유하는 M개의(M≦N) VM에 대해서만 IPA-to-PA 매핑을 하고 나머지 VM의 IPA에 대해서는 공유메모리의 PA를 매핑하지 않으면 되기 때문이다.

이에 반면, 시간적 격리는 공유 메모리에 대한 접근이 atomic하게 이루어져야 됨을 의미한다. 다시 말해, 공유 메모리 접근에 있어서 하나의 VM이 접근하여 연산을 수행 중일 때 다른 VM이 해당 메모리를 접근할 수 없어야 된다. 이는 다른 Realm VM에서 실행되는 소프트웨어에 대하여 서로 신뢰할 수 없다는 점에 기인한다. 서로 신뢰할 수 있다면, 다른 인클레이브에서 실행 시키지 않고, 하나의 VM에서 같이 실행시켰을 것이기 때문이다. 예를 들어, 데이터베이스 X를 A와 B 어플리케이션에서 사용한다고 가정했을 때, A와 B는 서로 신뢰하지 않기 때문에 A, B, 그리고 X는 서로 다른 인클레이브($Enc_{A,B,X}$)에서 실행되어야 한다. 이 때 성능 향상을 위해 Enc_X 메모리를 Enc_A, Enc_B와 시간적 격리 없이 공유한다면 A가 X를 읽거나 쓸 때 B가 해당 메모리를 관찰 또는 변경할 수 있기 때문에 보안 문제가 발생한다.

시간적 격리는 보통 semaphore나 lock 등의 concurrent 환경 프로그래밍에서 사용되는 개념들로 실현되지만, 앞서 설명했듯이 서로 다른 VM에서 실행되는 프로그램이 해당 개념들을 올바르게 실행하는지 신뢰할 수 없기 때문에 보안 문제가 발생한다. 즉, 메모리를 공유하는 악의적인 인클레이브가 lock 등을 bypass하고 공유 메모리를 접근할시에 시간적 격리가 지켜지지 않는 것이다.



(그림 2) 신뢰하는 중재자를 통한 공유메모리 접근에 대한 시간적 격리 적용 방안

3-3. 메모리 공유에 따른 보안 문제의 해결 방안

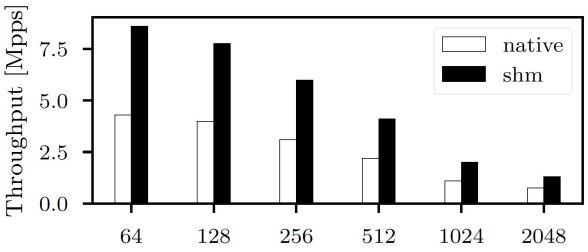
보안 문제를 해결하기 위한 시간적 격리는 공유 메모리 접근을 신뢰할 수 있는 시스템 요소를 통해 실행 시키는 것으로 해결할 수 있다. 즉, 신뢰할 수 있는 중재자(trusted coordinator)를 통한 메모리 접근을 통해 시간적 격리를 위한 primitive가 올바르게 수행될 수 있도록 하는 것이다.

ARM CCA의 경우 이러한 중재자는 Realm State EL2에서 실행되는 RMM이 맡을 수 있다. 2단계 페이지 테이블 관리를 통해 Realm VM 간 공간적 격리를 실현하는 주체임과 동시에, 시스템 부팅 시 EL3에서 실행되는 Secure Monitor와 같이 올바른 소프트웨어가 로딩되었는지 확인할 수 있으며, 일반적인 OS나 하이퍼바이저와는 다르게 코드/로직 크기가 작기 때문에 그 작동에 대한 정확성(correctness)를 사전에 검증할 수 있기 때문이다. 이를 위해 메모리 공유를 하고자 하는 인클레이브 어플리케이션들은 (1) 코드를 공유 메모리를 접근하는 부분

과 사유 메모리를 접근하는 부분으로 파티셔닝한 후, (2) RMM에 요청하여 공유 메모리 접근 코드를 전달하고, (3) RMM은 각 Realm VM 어플리캐이션 요청을 확인 후 시간적 격리가 실현되었는지 검증 후 공유 메모리 접근 코드를 로드한다. 공유메모리 접근 시에는 하이퍼바이저로 나가는 hypercall을 통해 공유메모리 접근 코드 수행을 요청하고, RMM에서 관리되는 코드를 통해 메모리 접근 후 다시 VM 실행 컨텍스트로 돌아오게 된다. 이러한 과정은 (그림 2)에 정리되어 있다.

4. 구현 및 실험 결과

Realm VM간 메모리 공유를 통한 성능 향상을 평가하기 위해 packet size에 따른 inter-VM communication throughput을 측정하였다. 메모리 공유를 하지 않는 native 성능의 경우, 메시지를 주고 받기 위해 인클레이브 간 안전한 통신 채널이 성립되어야 하기 때문에, 메시지를 암호화하여 Normal State 메모리에 복사하고, 이를 다시 수신 Realm VM 메모리에 복사 후 복호화하는 방식으로 VM끼리 통신했을 때의 성능을 측정하였다. 안타깝게도 CCA가 아직 실제 하드웨어로 존재하지 않기 때문에, ARM에서 공식적으로 지원하는 시뮬레이터인 ARM Fixed Virtual Platform(FVP) Base RevC-2-xAEMvA를 사용하여 실험을 진행하였다. 결과는 (그림 3)과 같으며, 공유 메모리를 통한 VM간 통신의 성능이 평균적으로 약 1.8배 좋게 측정되었다.



(그림 3) Inter-VM Communication Throughput

5. 한계 및 성능 향상 방안

비록 RMM을 신뢰하는 중재자로 활용한 Realm VM 사이의 메모리 공유를 통해 많은 성능 향상을 얻을 수 있지만, RMM을 사용하는 메커니즘은 매 공유 메모리 접근마다 VM을 나가야된다는 단점이 있다. 안타깝게도, 일반 VM과는 다르게 Realm VM의 경우 <표 2>와 같이 hypercall에 필요로 하는 사이클 수가 훨씬 더 많다. EL2까지만 나가면 되는 일반 VM과는 달리 Realm VM은 EL3까지 내려가 Realm 관련 자료구조, 레지스터 및 VM 상태를 저장해야 하기 때문이다. 이는 VM exit을 야기하지 않고 격리된 환경에서 신뢰할 수 있는 공유 메모리 접근 코드를 실행 시킬 수 있는 연구를 통해 해결해야 할 것이다.

VM	Normal (KVM)	CCA VM
Cycles	362	1,865

<표 2> VM 종류에 따른 hypercall시 소요 사이클

6. 결론

본 논문에서는 신뢰실행환경 성능 향상을 위해 필요한 메모리 공유가 기존 기술들에서는 불가능했음에도 불구하고 ARM의 새로운 기술인 CCA에서는 가능함을 분석하고, 이를 단순하게 적용했을 때의 보안 문제 및 신뢰하는 중재자를 통한 해결 방안을 제시한다. 허나, 이러한 해결 방안은 여전히 향상 가능한 여지가 있기 때문에, 이에 대한 연구가 계속되어야 할 것이다.

사사

본 연구는 2024년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원(IITP-2023-RS-2023-00256081), 한국연구재단(RS-2023-00277326), BK21 FOUR 정보기술 미래인재 교육연구단, 그리고 반도체 공동연구소의 지원을 받아 수행된 연구임.

참고문헌

[1] Victor Costan, “Intel SGX explained”, Cryptology ePrint Archive, 2016

[2] Dayeol Lee, “Cerberus: A Formal Approach to Secure and Efficient Enclave Memory Sharing”, ACM SIGSAC Conference on Computer and Communications Security, Los Angles, USA, 2022, pp.1871-1885

[3] Jason Zhijingcheng Yu, “Elasticlave: An Efficient Memory Model for Enclaves”, USENIX Security Symposium, Boston, USA, 2022, pp.4111-4128

[4] Xupeng Li, “Design and Verification of the ARM Confidential Compute Architecture”, USENIX Symposium on Operating Systems Design and Implementation, Carlsbad, USA, 2022, pp.465-484

[5] Pinto Sandro, “Demystifying ARM TrustZone”, ACM Computing Surveys, 51, 6, 1-36, 2019