정 보 보 호 학 회 지
제 33 권 제5호, 2023. 10

모바일 기기의 신뢰연산을 위한 디버깅 기술 활용

한 승 균*, 장 진 수*

요 약

중요 데이터와 서비스를 격리하여 보호하기 위한 신뢰연산기술이 모바일 기기에서 널리 활용되고 있다. 신뢰연산기술은 보통 하드웨어 기반 접근제어를 통해 메모리, 레지스터, 캐시 등의 하드웨어 자원을 공격자로부터 격리하는 신뢰실행영역을 시스템 내에 생성할 수 있게 한다. 하지만 보호 대상 소프트웨어에 악용 가능한 취약점이 존재할 경우 그 보안성이 파훼될 수 있다. 따라서 신뢰실행영역 내에도 소프트웨어에 대한 공격 효율성을 최소화할 수 있는 보안 기술이 적용되어야 한다. 모바일 디바이스에 주로 적용된 ARM 아키텍처에서도 포인터 인증, 메모리 태깅과 같은 다양한 하드웨어 기반 보안 기술들이 정의되고 있으며 최신 고사양 모바일 디바이스를 중심으로 적용되고 있다. 하지만 아키텍처 버전에 따라 가용한 하드웨어 보안 기술이 상이하기 때문에 보안기술의 범용성을 향상시키기 위한 방안 또한 중요하게 연구되어야 한다. 본 고에서는 모바일 신뢰연산기술의 보안성을 향상시키기 위한 대표적인 범용 보안 기술들에 대해 소개한다. 특히 유저서비스, 운영체제, 하이퍼바이저의 보안성을 향상시키기 위해 제안된 디버깅 와치포인트 기반 보안 기술들에 대해 분석하고 그 한계와 타 아키텍처 확장 가능성에 대해서 논의한다.

Ⅰ. 서 론

신뢰연산기술은 Intel, ARM, RISC-V 등 다양한 아키텍처에서 하드웨어 보안 기술로써 제공되고 있다. 특히, 모바일 디바이스에서 주로 활용되고 있는 ARM 아키텍처 기반 보안 기술인 TrustZone[17]은 모바일 뱅킹, 운영체제, 디지털 컨텐츠 등 다양한 소프트웨어와 중요 데이터를 보호하기 위해 응용되고 있다. 전 세계 스마트폰의 시장의 규모와 ARM 프로세서의 시장 점유율을 고려할 때, TrustZone을 기반으로 한 신뢰연산기술은 가장 널리 활용되고 있는 신뢰연산기술 중의 하나로 볼 수 있다.

일반적으로 신뢰연산기술은 메모리, 캐시, 레지스터 등 소프트웨어의 실행에 필요한 자원들에 대한 하드웨어 기반 접근제어를 가능하게 한다. 이를 활용하여 시스템 내에 신뢰실행영역을 구성하고 중요 소프트웨어를 공격자로부터 격리할 수 있다. 특히, 모바일 기기의 신뢰실행영역은 별도의 신뢰운영체제와 응용 소프트웨어가 격리된 환경에 설치되어 실행되는 구조로 운용된다. 따라서 신뢰실행영역에서 실행되는 소프트웨어에 악용 가능한 취약점이 존재하지 않아야 진정한 의미의 신뢰실행영역을 구성할 수 있다.

불행히도 신뢰연산기술을 겨냥한 다양한 공격들이 연구·보고되고 있다. 대부분은 신뢰실행영역에서 실행되는 소프트웨어의 취약점을 악용한 공격들이다. 가령, 공격자는 취약점이 존재하는 신뢰 응용소프트웨어로 악성 페이로드를 전송하여 취약점을 공략하고 해당 소프트웨어를 장악한다. 또한, 획득한 신뢰 응용소프트웨어의 권한을 바탕으로 신뢰 운영체제에 대한 추가 공격을 감행한다. 따라서 신뢰실행영역에 대한 보안신뢰성을 보장하기 위해서는 신뢰실행영역 내에서 실행되는 소프트웨어의 보안성을 향상시키기 위한 노력이 요구된다.

본 고에서는 모바일 신뢰실행영역 소프트웨어의 보안성을 향상시키기 위한 몇 가지 연구와 기법들에 대해 소개한다. 특히 ARM 아키텍처의 버전 또는 모바일 디바이스의 사양별로 가용한 보안 하드웨어의 종류가 상이하다는 점을 고려하여 범용하드웨어 기반 보안기술을 구현하는 방안에 초점을 맞춘다. 아키텍처에서 제공하는 여러 범용 하드웨어 기능들(예: 시스템 성능측정, 부동소수점 연산 등) 중에서도 디버깅 기술을 이용하여 보안기술을 구현한 사례를 소개한

본 연구는 정보통신기획평가원의 지원(No.2020-0-01840, No2021-0-00724)과 한국연구재단의 지원(RS-2023-00240697)을 받아 수행된 연구임.

* 충남대학교 컴퓨터융합학부(대학원생, yakmg3000@gmail.com, 조교수, jisjang@cnu.ac.kr)

다. 구체적으로는 메모리 영역에 대한 접근을 감시하는 와치포인트 기능을 활용한 프로세스 내부의 메모리 격리, 커널 권한 메모리 접근제어, 하이퍼바이저 자가 보호 기술들에 대해 분석한다. 또한, 소개된 기술들의 한계점과 타 아키텍처 적용 가능성 등에 대해서도 논의한다.

Ⅱ. 관련연구

2.1. 모바일 신뢰연산 기술

일반적으로 ARM TrustZone과 같은 모바일 신뢰연산기술은 디바이스 제조사에 의해 폐쇄적으로 관리된다. 가령, 소프트웨어를 신뢰연산기술로 보호하기 위해서는 제조사와 관련 기술에 대한 기밀 유지 협약을 맺고 개발된 소프트웨어에 대한 제조사 주도 신뢰성 검증을 통과해야 한다. 이러한 폐쇄적인 운영 방식은 신뢰실행영역 내에서 실행되는 소프트웨어를 엄격하게 제한함으로써 그 보안성을 향상시키는 반면 신뢰연산기술에 대한 접근성을 제한한다. 따라서, 학계에서는 ARM TrustZone 이외의 다양한 범용 기술(예: 가상화)을 활용하여 신뢰실행영역을 구성하는 방법을 연구해왔다[1], [2], [3], [4], [5], [6], [7]. 특히, 이런 연구들은 메모리 및 주변장치 보호에 국한되지 않고 다중 신뢰실행영역의 병렬 생성, 성능 향상, 유연한 동적 자원 할당 등을 가능하게 하였다.

ARM TrustZone을 활용하여 입출력 장치의 보안성을 향상시키기 위한 연구도 진행되어왔다. 원격으로 기기 상태를 인증하고 특정 주변장치를 제어하거나[8], 기기 소유자에게 주변장치에 대한 신뢰할 수 있는 접근제어 기능을 제공하기 위한 기술[9] 구현을 위해 ARM TrustZone이 활용되었다.

이외에도 TZ-RKP[10]는 운영체제의 무결성 보호를 위해 페이지 테이블, 시스템 레지스터 등 시스템 중요 자원에 대한 관리를 신뢰실행영역 내에서 수행하였다. SeCReT[11]은 일반실행영역과 신뢰실행영역 간의 안전한 통신 채널을 생성하기 위한 메모리 및 암호키 보호 기술을 제안하였다. DarkneTZ[12]은 인공지능 모델 유출을 방지하기 위해 일부 추론 레이어를 신뢰실행영역 내에 격리하는 방법을 제안하였다. MyTEE[13]는 TrustZone이 부재한 환경에서 가상화 기술을 활용하여 신뢰실행영역을 생성하는 방법을 제안하였다.

2.2. 내부 권한 분리 기술

공격자로부터 프로세스를 격리하고 보호하기 위한 다양한 방법이 제안되어왔다. 가상화 기술은 공격자의 주요 타겟이 되는 OS보다 높은 권한을 기반으로 CPU와 메모리를 가상화하여 여러 가상머신을 구동할 수 있도록 한다. 특히, 가상화 지원을 위한 2단계 페이징(Stage-2 Paging)은 가상머신의 운영체제가 사용하는 물리주소인 '중간물리주소'를 실제 물리주소(머신주소)로 변환하고 메모리 영역에 대한 접근 권한 설정을 가능하게 하므로 다양한 보안기술 구현을 위해 활용되어 왔다. 가령, Trustvisor[14], Overshadow[15], Inktag[16], MyTEE[13]는 어플리케이션의 중요 로직을 보호하기 위해 가상화 기반의 메모리 격리 기술을 활용하였다.

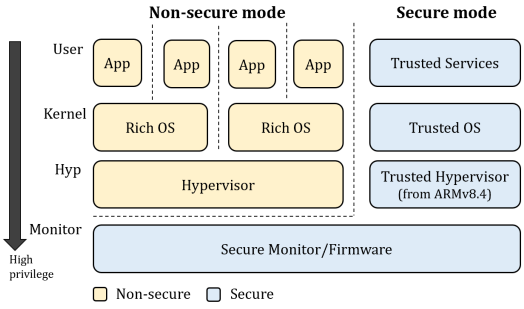
한편, 하드웨어 보안 기술인 Intel SGX[18]와 ARM TrustZone[17] 역시 소프트웨어의 중요 로직을 격리하고 보호 가능하게 한다. RISC-V 아키텍처에서도 AP-TEE Extension[19], 페이지 워커 수정[20], 엔클레이브 필터 엔진[21], 태그 엔진[22] 등 다양한 하드웨어 보안기술들이 연구되고 있다. 특히, Keystone[23]은 RISC-V에서 기본적으로 제공하는 하드웨어 보안기능인 PMP(Physical Memory Protection) 을 활용하여 소프트웨어의 중요 로직을 격리하여 실행할 수 있도록 하였다.

앞서 언급된 기술들은 프로세스 수준의 메모리 격리를 보장한다. 따라서 HeartBleed[24]와 같이 동일한 프로세스 또는 보호(격리) 단위 내에서 발생하는 공격에는 여전히 취약하다. 이러한 문제점을 해결하기 위해 Secage[25]는 어플리케이션을 여러 구역으로 나누고 2단계 페이징을 활용한 하이퍼바이저 기반의 내부 권한 분리를 수행하였다. Shred[26]는 메모리 도메인과 도메인 접근제어 레지스터(DACR, Memory domain and domain access control register)를 활용하여 세분화된 보안 메모리를 제공하였다.

Ⅲ. 배경 지식

3.1. ARM 아키텍처

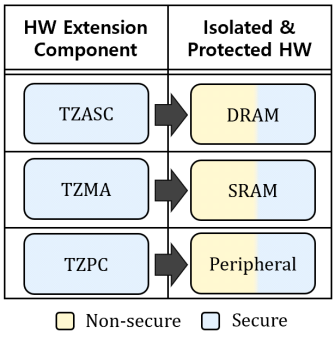
[그림 1]은 ARM 아키텍처 기반 CPU에서 제공하는 보안 상태 및 모드를 나타낸다. 모니터 모드에서 설정 가능한 시스템 레지스터 중 하나인 SCR_EL3는



〔그림 1〕 ARM 권한 및 CPU 보안 상태

CPU의 보안 상태를 설정하기 위한 플래그를 제공한다. 따라서 모니터 모드에서 실행되는 소프트웨어는 해당 플래그 조작을 통해 CPU 상태를 보안 및 비보안 상태로 변경할 수 있다. 한편, 유저, 커널, 하이퍼바이저 모드는 ARMv8.4 이상의 CPU의 보안/비보안 상태 모두에서 지원된다.

CPU의 보안 상태와 함께 TrustZone 하드웨어 확장 구성 요소([그림 2])를 활용하여 신뢰실행영역을 구성할 수 있다. TZASC와 TZMA는 각각 DRAM과 SRAM에 보안/비보안 영역을 구성할 수 있게 해준다. TZPC는 주변장치를 보안 전용으로 설정할 수 있도록 한다. 가령, CPU가 보안 상태일 때만 특정 중요 하드웨어 자원(예: 키패드)에 대한 접근이 허용되도록 시스템을 설정할 수 있다.



〔그림 2〕 TrustZone 확장 기능

3.2. ARM 디버깅 기술

본 절에서는 ARM에서 제공하는 하드웨어 디버깅 기능 중 하나인 와치포인트에 대해 소개한다.

3.2.1. 자가(Self-hosted) 디버깅

ARM과 x86과 같은 상용 프로세서 아키텍처에서는 두 가지 타입의 디버깅 메커니즘을 지원한다. 외부 디버그는 JTAG 디버거와 같은 외부 하드웨어 장비를 사용하여 디버깅을 수행하는 방식이다. 반면, 자가 디버그는 프로세서에서 발생한 디버그 관련 예외를 활용한다. 디버그 예외는 OS 커널과 같은 상위 권한을 가진 소프트웨어에 의해 처리된다. 가령, 중단점(breakpoint) 레지스터를 사용하면 중단점 예외를 발생시킬 명령어의 주소를 설정할 수 있다. 와치포인트 레지스터를 사용하면 특정 메모리 영역에 대한 접근이 와치포인트 예외를 발생시키도록 할 수 있다. 즉, 와치포인트와 중단점은 각각 데이터 접근과 명령어 실행에 대한 디버그 예외를 발생시킨다.

ARM 아키텍처에서는 (1) 중단점 명령어 예외, (2) 소프트웨어 스텝 예외, (3) 중단점 예외, (4) 와치포인트 예외의 4가지 디버그 예외를 지원한다.

중단점 명령어 예외는 64bit ARM 아키텍처에서 brk 명령어를 실행하면 활성화된다. 이와 같은 명령어를 특정 코드 영역 중간에 삽입해야 하므로 소프트웨어 중단점이라고도 한다. 소프트웨어 스텝 예외는 MDSCR_EL1(Monitor Debug System Control Register)[27] 레지스터의 SS(Single Stepping) 플래그를 세팅함으로써 활성화할 수 있으며, 단일 명령어를 실행한 후 다음 명령어가 실행될 때 디버그 예외를 발생시킨다. 중단점 예외와 와치포인트 예외는 MDSCR_EL1 레지스터의 MDE(Monitor Debug Events) 비트를 설정하여 활성화할 수 있다.

기본적으로 모든 예외는 커널로 라우팅 되어 처리되지만, 하이퍼바이저 전용 MDCR_EL2(Monitor Debug Configuration Register)[28] 레지스터의 TDE(Trap Debug Exceptions) 비트를 활성화하면 하이퍼바이저 모드로 모든 디버그 예외를 라우팅할 수

〔표 1〕 디버그 예외 라우팅을 위한 제어 플래그 설정

	디버그 예외가 생성된 모드:		
	User	Kernel	Hypervisor
	트랩 & 핸들링할 모드:		
NS=1, TDE=0	Kernel	Kernel	Hypervisor*
NS=1, TDE=1	Hypervisor	Hypervisor	Hypervisor
NS=0, TDE=X	Kernel	Kernel	N/A
* 중단점 명령어 예외만 허용			

있다[표 1]. 보안 모드에서는 TDE 값에 상관없이 디버그 예외가 (보안)커널로 라우팅 된다. 이외에도 디버깅 기능을 활성화하기 위해서 MDSCR_EL1 레지스터의 KDE(Local(Kernel) Debug Enable) 비트를 활성화하고 PSTATE(Process State) 레지스터의 디버그 예외 마스크 플래그(D)를 지워야 한다. 가령, 하이퍼바이저로 디버그 예외를 라우팅하기 위해서 PSTATE.D=0, MDSCR_EL1.KDE=1, MDCR_EL2.TDE=1 로 설정해야 한다. 본 연구에서는 이러한 기능들을 커널 및 하이퍼바이저 레벨의 보안 어플리케이션을 구현하는 데 활용한다.

3.2.2. 와치포인트 예외

와치포인트 예외를 생성하기 위해, MDSCR_EL1.MDE 외에도 와치포인트 주소 설정을 위한 DBGWVR(n)_EL1(Debug Watchpoint Value Register, n=0-15)[29] 레지스터와 와치포인트 세부 제어를 위한 DBGWCR(n)_EL1(Debug Watchpoint Control Register, n=0-15)[30]를 설정해야 한다. 하나의 와치포인트를 설정하기 위해 동일한 인덱스(n)를 가진 각각의 레지스터가 함께 설정되어야 한다. ARMv8에서는 최대 16개의 와치포인트 레지스터쌍을 구성할 수 있으며, Juno ARM 개발 보드[Juno]에서는 4개의 레지스터쌍을 지원한다(DBGWVR0_EL1-DBGWCR3_EL1, DBGWVR3_EL1 – DBGWCR3_EL1 지원).

DBGWVR(n)_EL1 레지스터는 모니터링하고자 하는 시작 주소를 저장하며, 모니터링되는 영역의 크기에 따라 워드 또는 더블워드 크기에 맞춰 정렬되어야 한다. DBGWCR(n)_EL1 레지스터는 인덱스에 상응하는 DBGWVR(n)_EL1 레지스터에 저장된 주소에 대해 모니터링 크기, 읽기/쓰기 모드, 예외 권한, 활성화 제어와 같은 속성을 설정한다. DBGWCR(n)_EL1 레지스터의 BAS[12:5] (Byte Address Select), MASK[28:24] 플래그는 설정된 와치포인트 시작 주소로부터 감시할 크기를 설정하는 데 사용된다. 크기가 8바이트 이하인 경우 BAS 플래그가 사용된다. 해당 플래그는 8비트 크기이며 각 비트는 와치포인트 값 레지스터의 주소에서 시작하는 더블워드의 바이트를 선택한다. 가령, 와치포인트 값 레지스터가 0x1000이고, BAS[5]==1인 경우 0x1005에 액세스하면 와치포인트 예외가 생성된다. MASK 플래그는 2의 거듭제곱 크기의 주소 범위를 모니터링 하는 데 활용된다. 최대 5비트 길이를 지원하므로 최소 8바이트에서 최대 2기가 바이트까지의 크기를 지정할 수 있다. 가령, MASK가 0b01100이고 DBGWVR(n)_EL1 레지스터가 0x80000000로 설정된 경우 모니터링 크기는 212이므로 0x80000000에서 0x80001000 사이의 모든 주소에 대한 접근 시 예외가 발생한다.

예외가 생성될 수 있는 권한 모드는 SSC[15:14](Security state control), HMC[13](Higher Mode Control), PAC[2:1](Privilege of Access Control) 플래그의 조합에 따라 정의된다. 모니터 모드를 제외하고 두 보안 상태의 모든 모드에서 예외를 생성할 수 있다. [표 2]는 비보안 상태의 단일 모드에서 명시적으로 디버그 예외를 생성하기 위한 플래그 조합을 나타낸다. LSC[4:3](Load/Store Control) 플래그는 모니터링할 액세스 타입이 읽기인지 쓰기인지, 혹은 읽기/쓰기 둘 다 인지 여부를 결정한다[표 3]. 마지막으로 DBGWCR(n)_EL1 레지스터의 최하위 비트는 와치포인트의 활성화 유무를 나타낸다.

〔표 2〕 비보안 상태에서 특정 권한 모드를 모니터링하기 위한 와치포인트 제어 레지스터의 플래그 설정

HMC	SSC	PAC	보안 상태	모니터링할 모드:
0	01	10	비보안	User
0	01	01	비보안	Kernel
1	11	00	비보안	Hypervisor

〔표 3〕 DBGWCR(n)_EL1.LSC 비트에 따른 와치포인트 예외가 발생할 수 있는 접근 유형

LSC	와치포인트 접근 유형:
01	Load
10	Store
11	Load & Store

Ⅳ. 와치포인트 기반 보안 기술

본 장에서는 전술된 와치포인트 기능 중 보안 목적으로 사용될 수 있는 유용한 기능을 분류한다. 또한, 신뢰실행영역의 유저, 커널, 하이퍼바이저의 보안성 향상을 위해 적용될 수 있는 와치포인트 기반 보안기술들을 소개한다.

4.1. 유용한 특성

(1) 유용한 보안 속성 : 다음에 열거된 와치포인트의 특성들은 보안 어플리케이션 구축에 유용하게 활용될 수 있다.

(특성1) 프로세서별 구성. 와치포인트 레지스터는 각각의 프로세서별로 지원된다. 즉, 스레드 기반으로 와치포인트 기반 보안 솔루션을 설계할 수 있다.

(특성2) 권한 인식 감시. 와치포인트 예외는 현재 권한에 따라 상이한 동작을 수행하도록 구성할 수 있다. 가령, 특정 주소 범위가 사용자와 커널에서 액세스할 수 있더라도 커널 권한 액세스만 예외를 생성하는 방식으로 와치포인트를 구성할 수 있다.

(특성3) 비특권 명령어와의 호환성. 비특권 명령어의 실행은 (***특성2***)의 와치포인트 감시 영역에 대한 권한 설정에 영향을 받는다. 가령, 커널 권한 접근에 대한 감시를 수행하는 영역을 비특권 메모리 읽기(예: ARM의 ldtr 명령어) 명령어로 접근하면 예외가 생성되지 않는다.

(특성4) 명령어 실행에 무관한 감시. 와치포인트는 일종의 데이터 중단점이므로 와치포인트 영역에 존재하는 명령어를 실행할 경우 별도의 예외가 발생하지 않는다. 이러한 속성은 실행 전용 메모리(XOM, Execute Only Memory)를 구현하는데 적합하다.

(특성5) 자가 관리 가능한 와치포인트 예외. 권한 있는 특정 모드(예: 커널 및 하이퍼바이저)로 와치 포인트 예외를 라우팅하고 처리하도록 구성되어있더라도 해당 모드에 대한 와치포인트 감시 활성화는 허용된다. 이는 예외 처리 중에 문제가 발생하지 않도록 예외가 발생하면 와치포인트를 자동으로 비활성화(PSTATE.D 플래그 설정)하는 방식으로 지원된다.

(2) 와치포인트 구성 요구 사항 : 와치포인트가 의도한 대로 작동하도록 보장하려면 다음과 같은 요구 사항을 충족해야 한다.

(R1) 모니터링 주소 크기 및 정렬. 와치포인트 설정을 위해 감시 크기 및 주소 정렬 요구 사항을 엄격하게 준수해야 한다. 요구 사항을 충족하지 못하면 와치포인트가 활성화되지 않는다.

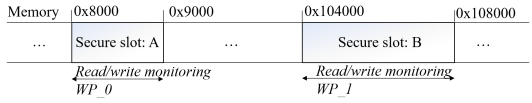
4.2. 프로세스 내부 메모리 격리[31]

4.2.1. 보안 메모리 생성

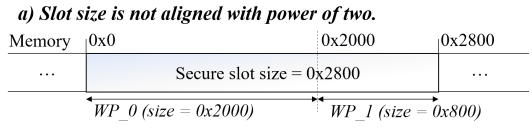
와치포인트는 CPU별로 구성할 수 있으므로 이를 활용하여 특정 메모리 영역에 대한 와치포인트 읽기/쓰기 감시를 활성화하여 스레드 내 보안 메모리를 생성할 수 있다. 보호되는 영역에 대한 접근으로 인해 생성된 와치포인트 예외는 커널로 라우팅되어 처리된다. 보안 메모리의 안전성을 위해 현재 발생한 예외가 합법적인(허용된) 코드에 생성된 경우에만 와치포인트 감시가 비활성화되어야 한다.

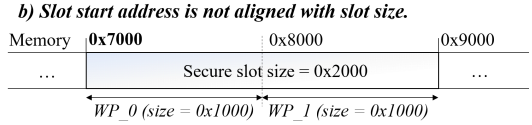
[그림 3]과 같이 슬롯 하나당 와치포인트 하나를 할당하는 간단한 운용 방식의 경우, 생성 가능한 보안 메모리 슬롯의 수는 사용 가능한 와치포인트 수[1]로 제한된다. 또한, [그림 4]에서와 같이 보안 슬롯의 크기나 시작 주소가 2의 거듭제곱이 아닌 경우 보안 영역을 생성하기 위해 하나 이상의 와치포인트가 필요하게 된다.

따라서 [31]에서는 가능한 보안 메모리 슬롯 생성 수(Nslot)를 최대화하기 위해 예약된 선형 메모리 영역 내에서 크기와 주소가 2의 거듭제곱인 동시에 모든 슬롯의 크기가 동일하도록 슬롯을 할당한다. 또한,



〔그림 3〕 할당 가능한 보안 슬롯의 개수는 사용 가능한 와치포인트 개수와 동일



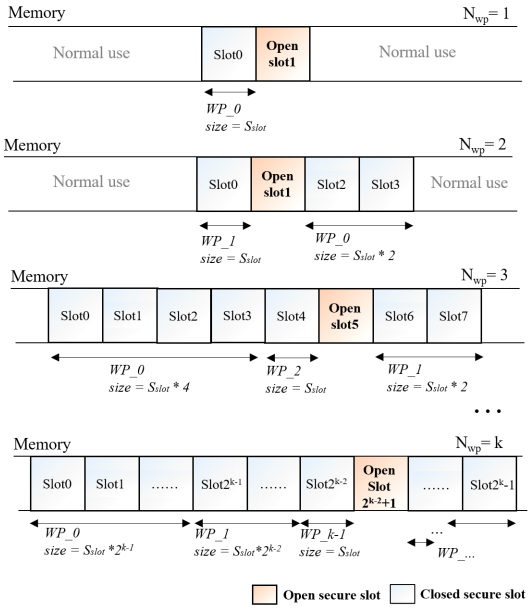


〔그림 4〕 a) 슬롯의 크기가 2의 거듭제곱이 아닌 경우 또는 b) 슬롯의 시작 주소가 슬롯의 크기에 비례하여 정렬되지 않은 경우, 보안 슬롯을 보호하려면 2개 이상의 와치포인트 설정이 필요함

1) SoC(System On a Chip)에 따라 상이함

다음과 같은 조건을 만족하도록 한다. ㄱ) 전체 슬롯 크기는 사용 가능한 와치포인트로 처리할 수 있어야 하며, ㄴ) 특정 슬롯이 사용될 때(해당 영역에 대한 와치포인트를 일시적으로 비활성화), 나머지 슬롯은 사용 가능한 와치포인트를 구성하여 모니터링할 수 있어야 한다. ㄱ)을 만족시키기 위해서는 예약된 영역의 크기가 Nwp(지원되는 와치포인트 수) * 2GB(각 와치포인트의 최대 모니터링 크기)이내여야 한다. 가령, N_{wp}가 4일 경우 감시 가능한 영역의 최대 크기는 8GB이다. 또한, 각 와치포인트에서 모니터링하는 특정 영역(연속적인 슬롯 그룹)의 크기는 2의 거듭제곱이어야 한다. 각 슬롯 크기는 2의 거듭제곱이므로 각 와치포인트에서 모니터링할 슬롯 수도 2의 거듭제곱이어야 한다. 즉, N_{slot}도 2의 거듭제곱이다.

[그림5]에서 볼 수 있듯이, 오픈된 슬롯을 찾는 것은 이등분된 영역의 크기가 슬롯의 크기가 같아질 때까지 슬롯을 포함하는 메모리 영역을 계속해서 이등분하는 것으로, 이진 탐색과 유사하다고 할 수 있다. 이등분은 각 단계에서 대상 슬롯을 포함하지 않는 메모리 영역 중 하나를 와치포인트로 모니터링해야 하므로 와치포인트 개수(N_{wp})만큼 이등분을 수행할 수 있다. 즉, N_{slot}이 2^{Nwp}여야 함을 의미하며, 이는 4개의



〔그림 5〕 각 슬롯의 크기가 2의 거듭제곱 크기로 설정되고 모든 슬롯 크기가 동일하다고 가정할 때, 지원되는 와치포인트 수 N_{wp}를 활용하여 생성할 수 있는 총 슬롯 수 N_{slot}은 2^{Nwp}임

와치포인트를 지원하는 개발환경에서는 최대 16개의 슬롯을 생성할 수 있음을 의미한다. 또한, 양분된 영역은 단일 와치포인트로 감시 가능해야 한다. 즉, 이등분된 영역 중 가장 큰 첫 번째 이등분 영역의 크기는 2GB 이내여야 한다. 결과적으로 각 슬롯의 크기(S_{size})는 최대 2GB/(N_{slot}의 절반)까지 할당할 수 있으며, 여기서는 256MB(2GB/8)이다.

4.2.2. 접근제어

보안 슬롯에 대한 접근제어를 위해 접근 기반에 따라 와치포인트가 적절하게 구성되어야 한다. Shreds[26]와 Wedge[34] 등과 같이 슬롯에 액세스할 수 있는 중요 코드 영역이 사전에 검증되었다고 가정한다. 중요 코드를 실행하기 전에 보안 슬롯은 접근 가능한 상태로 변경되어야 한다. 이는 접근할 슬롯을 제외한 모든 슬롯이 감시되도록 와치포인트를 구성하여 수행된다. [알고리즘1]과 같이 감시되는 슬롯 그룹을 재귀적으로 검색하여 와치포인트를 재설정함으로써 특정 슬롯을 접근 가능 상태로 변경할 수 있다. 먼저, 예약된 메모리 전체를 양분한다. 이등분된 영역 중 대상 슬롯이 포함되지 않은 영역은 와치포인트를 설정하여 감시한다. 이등분된 나머지 반대편에 대해서는 대상 슬롯을 포함하는 마지막 슬롯까지 알고리즘을 재귀적으로 실행한다. 총 슬롯 수는 2^{Nwp}이므로

```
1  setupWP (start, end, targetSlot, slotSize);
2  /* start: current block start address */
3  /* end : current block end address*/
4  /* targetSlot: target slot address*/
5  /* slotSize: individual slot size*/
6  mid = (end - start) / 2;
7  if mid < = targetSlot then
8  |   configWPRegister(start, mid);
9  |   if (end - mid) != slotSize then
10 |   |   seupWP(mid, end, targetSlot, slotSize);
11 |   end
12 else
13 |   configWPRegister(mid, end);
14 |   if (mid - start) != slotSize then
15 |   |   setupWP(start, mid, targetSlot, slotSize);
16 |   end
17 end
```

〔알고리즘 1〕 setupWP() 함수는 대상 슬롯 이외의 모든 슬롯을 모니터링하도록 와치포인트를 구성하기 위해 재귀적으로 호출된다.

모든 와치포인트를 구성하려면 N_{wp}번의 재귀호출이 요구된다.

접근한 영역에 대한 보호를 재활성화하려면 와치포인트를 재구성하여 해당 영역과 관련된 보안 슬롯을 감시해야 한다. 이는 가용한 슬롯을 찾는 과정에 비해 간단하다. 단순히 S_{size} * N_{slot}의 크기의 전체 예약 메모리에 대해 와치포인트 읽기/쓰기 모니터링을 활성화하면 된다. [31]의 시제품에서는 S_{size}를 256MB로 제한하므로 예약된 메모리의 최대 크기는 4GB(256MB * 16)까지 가능하다. 즉, 각각 2GB 모니터링을 하는 최대 2개의 와치포인트를 구성하면 된다.

4.2.3. 구성 요소 및 사용법

[31]에서 제시된 방안으로 스레드 내 보안 메모리를 생성하고 안전하게 사용하기 위해 설계된 사용자 라이브러리와 커널의 핵심 구성 요소를 소개한다.

사용자 라이브러리. 와치포인트 기반 스레드 내 보안 메모리 생성을 지원하기 위한 사용자 라이브러는 5개의 API를 제공한다. *initSlotAll*은 모든 슬롯(S_{slot} * N_{slot}) 할당에 필요한 전체 메모리 크기를 계산하고 메모리를 예약한다. 메모리 예약 전에 와치포인트 기반 감시를 위한 주소 정렬(2의 거듭제곱 주소) 조건을 만족시키기 위해 매개변수 SlotSize는 2의 거듭제곱으로 반올림된다. 예약된 메모리도 내부적으로 *memalign* API를 사용하여 크기를 정렬한다. *enterCriticalRegion* 및 *exitCriticalRegion*은 SlotNum을 매개변수로 해당 슬롯을 여닫는데 필요한 와치포인트 레지스터인 DBGWCR(n)_EL1과 DBGWVR(n)_EL1의 설정값을 생성한다. 생성된 레지스터 설정값은 커널 드라이버로 전달된다. 추가로 *enterCriticalRegion*은 호출되는 함수의 스택 주소를 임의의 주소로 변경한다. *exitCriticalRegion*은 함수 에필로그가 완료된 후에 원래 스택 주소를 복원한다. *wp_malloc* 및 *wp_free*는 현재 사용 중인 보안 메모리 슬롯에서 메모리를 할당하고 해제하는 역할을 수행한다.

커널 패치 및 드라이버. 와치포인트 관련 레지스터는 커널, 하이퍼바이저 모드 등 유저 모드 보다 높은 권한을 바탕으로 설정될 수 있다. [31]에서는 커널 패치 및 커널 드라이버를 구현하여 보안 메모리 생성

```
int main(int argc, char *argv[]){
  /* Variables initialization */
  unsigned char *pass_input;
  unsigned char *pass;
  ...

  initSlotAll(0x2000); // Memory reservation.
                       Each slot size is 0x2000.
  enterCriticalRegion(0);
  pass_input = (unsigned char *) wp_malloc(
    MAX_PASSWD_BUF, 0);
  pass = (unsigned char *) wp_malloc(
    MAX_PASSWD_BUF, 0);
  exitCriticalRegion(0);
  ...

  /* processing an input parameter */
  if (passlen ==0) {
   ...
   enterCriticalRegion(0);
   passlen = passwd_to_utf16((unsigned char*)
     optarg, strlen((char *)optarg),
     MAX_PASSWD_LEN, pass);
   exitCriticalRegion(0);
   ...
  }

  /* Encrypting the input stream */
  if (mode == ENC) {
   ...
   enterCriticalRegion(0);
   rc = encrypt_stream(infp, outfp, pass,
    passlen);
   exitCriticalRegion(0);
   ...
  }

 /* Cleanups */
 ...
 enterCriticalRegion(0);
 memset(pass, 0, MAX_PASSWD_BUF);
 wp_free(pass, 0);
 wp_free(pass_input, 0);
 exitCriticalRegion(0);

 return rc; // End of main.
}
```

〔코드 1〕 AESCrypt의 API 사용 예제

에 필요한 와치포인트 설정을 수행하였다. 커널 드라이버는 *ioctl*을 통해 *enterCriticalRegion* 및 *exitCriticalRegion*과 같은 사용자 API와 통신한다. API에서 구성된 와치포인트 레지스터 설정값들을 전달받아 수정된 *thread_info* 커널 데이터 구조체의 *sec_thread_mem* 변수에 복사한다. 이후 해당 프로세스 실행 시 와치포인트 레지스터를 설정함으로써 보

호 영역에 대한 감시를 수행한다.

사용 예제. [코드 1]은 제안된 방법이 오픈소스 파일 암호화 프로그램인 AESCrypt에 적용된 예시 코드이다. 파일 암호화를 위해 입력된 비밀번호를 스레드 내 보안 메모리에 위치시켜 보호하는 방법을 제시한다. 보안 슬롯 #0에는 *pass_input*과 *pass* 변수가 할당된다. 해당 변수는 원래의 소스 코드에서는 스택 변수였지만 *wp_malloc* API를 사용하여 힙 변수로 변경한다. *passwd_to_utf16* 및 *encrypt_stream*과 같은 서브루틴을 호출하기 전후에 *enterCriticalRegion* 및 *exitCriticalRegion* API를 호출해야 한다. 이러한 함수는 보호된 변수들에 접근해야 하기 때문이다. 마지막으로 *main* 함수를 종료하기 전에 *wp_free* API를 사용하여 할당된 보안 슬롯을 해지한다. 제시된 예제는 와치포인트를 활용하여 스레드 내 보안 메모리 생성이 가능함을 보인다. 세분화된 권한 분리 방식[34] 도입 및 컴파일러 툴과의 결합[26]을 통해 제안된 방식의 보안성과 사용성을 재고할 수 있을 것으로 예상된다.

4.3. 커널 권한 접근제어 기술[32]

신뢰실행영역 내의 운영체제 커널 또한 공격의 대상이 될 수 있으므로 자체적으로 보안성을 향상시킬 수 있는 방안들이 모색되어야 한다. 본 절에서는 (1) 커널 권한의 유저 메모리 영역에 대한 접근을 제어하는 특권모드 접근 방지(PAN, Privileged Access Never) 기술과 (2) 특정 메모리 영역을 읽기 권한 없이 실행 권한만 가지도록 강제하는 실행 전용 메모리(XOM, Execute Only Memory) 기술을 와치포인트를 활용하여 구현하는 방법을 소개한다.

4.3.1. 와치포인트 기반 특권모드 접근 방지

특권모드 접근 방지 기술은 커널 모드에서 사용자 영역 메모리에 접근하지 못하게 제한하는 하드웨어 보안기술로써 ARMv8.1 이상의 CPU에서만 제공된다[35]. 따라서 8.1버전 이하의 ARM CPU에서는 해당 기술을 활용할 수 없다[36]. 리눅스에서는 페이지 테이블을 활용하여 특권모드 접근 방지 기술이 제공되지 않는 기기에서 비슷한 기능을 제공한다[37]. 64비트 ARM 아키텍처 기반 리눅스는 커널 실행 중에 TTBR0(유저영역 매핑을 위한 페이지테이블 베이스 레지스터)의 값으로서 유저 영역에 대한 메모리 매핑이 존재하지 않는 테이블 주소를 설정하는 방식으로 특권모드 접근 방지 기술을 에뮬레이션한다. 이로 인해 커널 모드에서는 사용자 공간 메모리에 접근할 수 없게 된다. 하지만, 사용자 메모리에 정상적으로 접근하는 커널 작업(예: *copy_from_user*)을 지원하기 위해 일시적으로 유효한 페이지 테이블 주소를 TTBR0에 설정함으로써 유저 공간을 재매핑한다. 이렇게 리눅스에서는 특권모드 접근 방지 기능을 제공하는 하드웨어가 존재하지 않을 경우, 이를 소프트웨어적으로 에뮬레이션한다. 하지만 유저 영역에 대한 매핑이 재활성화 되어 있는 동안 커널 버그에 대한 공격이 발생할 수 있으므로 이러한 방식은 잠재적 취약하다.

와치포인트 기반 특권모드 접근 방지. 와치포인트를 활용하여 커널 모드 실행 중에 사용자 영역에 대한 접근을 제한하는 방안을 제시한다. 이는 ***특성2 및 특성3(4.1 섹션)***에 의해 달성된다. 프로세서가 커널 모드로 전환되면 사용자 공간에 대한 모든 커널 권한의 접근(읽기/쓰기)을 감시하도록 와치포인트를 구성한다(***특성2***). 또한, 커널의 합법적인 사용자 공간 작업을 지원하기 위해 사용자 공간에 접근하는 커널 함수의 읽기(*LDR*) 및 쓰기(*STR*) 명령어들을 비특권 읽기(*LDTR*) 및 쓰기(*STTR*) 명령어들로 대체한다. 와치포인트는 사용자 공간에 대한 커널 권한의 접근만 감시하도록 설정된 상태이기 때문에 비특권 읽기/쓰기 명령어들을 통한 접근은 와치포인트 예외를 발생시키지 않는다(***특성3***). 결국, 와치포인트의 특성에 대한 이해와 적절한 설정을 통해 커널 모드에서 사용자 메모리 영역에 대한 접근을 제한하는 특권모드 접근 방지 기술을 구현할 수 있다.

4.3.2. 실행 전용 메모리

실행 전용 메모리는 특정 메모리 영역을 읽어 유용한 공격 코드를 동적으로 탐색하는 공격을 방지하기 위해 도입되었다[38]. 읽기 권한 없이 실행만 가능한 메모리를 구성하기 위해 과거 연구들은 메모리 페이징과 같은 하드웨어 기능을 활용하였다. 가령, XnR[39]은 실행 영역에 대한 메모리 매핑을 우선 제거한다. 이후, 프로그램 실행 중에 제거된 영역 접근 시 발생하는 페이지 폴트를 처리(재매핑)하는 방식으

로 실행 전용 메모리를 구현하였다. Norax[40]는 페이지 테이블 엔트리의 액세스 권한 비트를 활용하여 사용자 메모리가 실행할 수 있으면서도 읽을 수 없도록 강제한다. x86에서는 가상화 지원을 위한 확장 페이지 테이블(Extended Page Table, EPT)을 활용하여 특정 메모리 페이지에 대한 읽기 권한을 제거함으로써 실행 전용 메모리를 구현하였다[41-44].

와치포인트 기반 커널 실행 전용 메모리. 와치포인트가 모니터링하는 메모리 영역에서 실행되는 명령어에서는 예외가 발생하지 않는다(***특성4***). 이 속성은 실행 전용 메모리를 구현하기에 적합하다. 와치포인트에 의한 감시 영역을 커널 텍스트 영역의 시작 주소로 설정한다. 실행 전용 메모리 구현은 읽기 감시를 수행하는 것만으로 충분하지만 커널 무결성 보호를 위해 쓰기 접근 감시도 수행하도록 와치포인트를 설정할 수 있다. 와치포인트의 구성 및 활성화는 CPU 모드가 사용자에서 커널로 전환될 때 수행된다. (***특성2***)에 의해 커널 권한의 접근만 감시하도록 와치포인트를 구성할 수 있으며, 이는 사용자 모드로 전환 시 와치포인트를 재구성(비활성화)할 필요가 없음을 의미한다.

4.4. 하이퍼바이저 자가 보호 기술[33]

가상 머신 관리를 위한 소프트웨어인 하이퍼바이저는 보안 어플리케이션 구현을 위해서도 활용되어 왔다. 가령, x86[45],[46] 및 ARM[47] 환경에서 OS 커널을 보호하기 위한 신뢰 기반의 역할을 수행하였다. 하지만, 하이퍼바이저 자체도 취약점을 수반하며 공격 대상이 될 수 있다[48]. 본 절에서는 하이퍼바이저 보호 기법의 하나로써 와치포인트 기반의 하이퍼바이저 무결성 보호 기법을 소개한다.

4.4.1. 가정 및 공격 모델

본 논문에서는 하이퍼바이저 자체가 공격자의 임의 메모리 접근 및 실행 흐름 탈취를 가능하게 하는 취약점을 수반할 수 있다고 가정한다. 또한, 공격자는 하이퍼바이저의 코드나 데이터 조작을 시도할 수 있다. 한편, 공격 대상이 되는 하이퍼바이저는 운영체제 커널의 무결성을 보호하는 역할을 수행한다고 가정한다. 이는 상용 기기인 삼성 갤럭시 폰 등에 실적용된 기술로써 운영체제 커널의 코드 및 읽기 전용 데이터의 변조를 방지한다.

(가정) 커널 무결성 보호[47]. ARMv7부터 하드웨어 지원 가상화 기능이 지원된다. 이를 활용하여 커널 무결성 보호 기술을 구현할 수 있다. 가상 머신(VM, Virtual Machine)의 물리 주소를 실제 머신 주소로 변환하는 2단계 페이징을 활용하여 커널 텍스트 영역을 읽기 전용으로 매핑함으로써 변조를 방지한다. 또한, 본래 커널에 의해 관리되는 페이지 테이블을 읽기 전용으로 매핑하여 운영체제 자체도 이를 변경할 수 없도록 한다. 그리고 페이지 테이블 관리와 중요 시스템레지스터를 조작하는 함수들에 하이퍼바이저 호출을 위한 명령어인 HVC(하이퍼콜, Hypervisor Call)를 삽입한다. 즉, 페이지테이블 및 시스템레지스터의 변경 시도는 하이퍼바이저로 전달되어 안전성이 검증된 후에 에뮬레이션 된다.

(공격 벡터1) 하이퍼바이저 모드 전용 페이지 테이블. 하이퍼바이저 모드 내에서도 가상 주소를 사용 가능하다. 일반 OS와 마찬가지로 하이퍼바이저는 MMU를 활성화하기 위해 하이퍼바이저 전용 페이지 테이블과 TTBR_EL2(하이퍼바이저용 페이지 테이블 베이스 레지스터)을 구성해야 한다. 페이지 테이블을 구성함으로써 하이퍼바이저 코드와 데이터를 읽기 전용으로 설정할 수 있으며, 데이터 실행 방지(DEP, Data Execution Prevention)과 같은 보안 기능을 활성화할 수 있다. 그러나 공격자가 하이퍼바이저 권한을 획득하면 이러한 보안 설정은 더 이상 유효하지 않다. 즉, 공격자는 하이퍼바이저의 읽기 전용 영역(코드 및 데이터)을 쓰기 가능하게 재매핑하고 자유롭게 수정할 수 있다[48].

(공격 벡터2) 2단계 페이지 테이블. 하이퍼바이저는 2단계 페이지 테이블을 관리함으로써 운영체제 텍스트(코드) 영역의 불변성을 보장할 수 있다. 하지만 하이퍼바이저 권한을 획득한 공격자는 하이퍼바이저 메모리 영역이 OS 커널에 매핑되도록 2단계 페이지 테이블을 조작함으로써 하이퍼바이저의 무결성을 파훼할 수 있다. 즉, 낮은 권한을 가진 커널에서 하이퍼바이저를 악의적으로 수정할 수 있게 된다.

4.4.2. 와치포인트 기반 하이퍼바이저 자가 보호

와치포인트를 활용하여 추가적인 하드웨어나 상위

권한의 소프트웨어 도입 없이 하이퍼바이저의 무결성을 보호하는 방법을 소개한다. 하이퍼바이저 페이지 테이블을 악용하는 첫 번째 공격 벡터를 제거하려면 하이퍼바이저 모드에서 MMU를 비활성화하면 된다. 이는 하이퍼바이저에 의한 페이지 테이블 관리 필요성을 제거함으로써 하이퍼바이저 코드를 간소화한다. 하지만, 페이징이 제공하는 보안 기능(접근 권한 제어 및 데이터 실행 방지 등)을 활용할 수 없으므로 텍스트 및 데이터를 포함한 모든 메모리 영역이 수정 가능한 상태로 변경된다. 이러한 문제를 해결하기 위해 보호 대상 영역(하이퍼바이저 코드와 데이터) 및 2단계 페이지 테이블에 대한 쓰기 접근을 와치포인트 설정을 통해 감시한다.

이를 통해 공격자가 중요한 자원을 직접 수정하는 것을 방지할 수 있다. 하지만 공격자는 실행 흐름 탈취를 통해 와치포인트 설정 명령을 악용하여 와치포인트 기반 보호를 해제할 수 있다. 이러한 문제점을 해결하기 위해 하이퍼바이저를 특권 영역과 비특권 영역으로 논리적으로 분리한다.

특권 영역은 *hyp_entry, hyp_exit, sec_func*의 세 가지 요소로 구성된다. *hyp_entry*는 하이퍼바이저 모드로의 진입을 처리하며 하이퍼바이저 예외 벡터와 프롤로그 코드로 구성된다. 예외 벡터는 현재 발생한 예외에 해당하는 예외 처리기를 호출한다. 가령, 커널이 호출한 하이퍼콜(HVC)은 하이퍼바이저 예외를 생성하고, 벡터에 의해 하이퍼바이저 예외 처리기로 전달된다. 프롤로그는 각 핸들러로 예외가 전달되기 전에 실행되는 코드로써 특권영역 내에 현재의 와치포인트 값(커널에서 설정된 값)을 저장한다. 커널의 실행 문맥(레지스터 값)과 리턴 주소는 하이퍼바이저 스택에 저장한다. 마지막으로 하이퍼바이저의 코드 및 데이터와 2단계페이지 테이블에 대한 쓰기 접근 감시를 위해 두 개의 와치포인트를 구성하고 활성화한다.

*hyp_exit*는 OS 커널로 복귀할 때 호출되며 저장되었던 커널 문맥정보(와치포인트 커널 설정 값과 일반 레지스터 및 스택)를 복원하고 *eret*(Exception Return) 명령어를 실행하여 CPU 모드를 커널로 전환한다.

*sec_func*는 와치포인트에 의해 쓰기 금지된 2단계 페이지 테이블의 정상적인 업데이트를 지원하기 위한 API이다. 비특권 영역에서 *sec_func*를 호출하면 함수에서는 내부적으로 하이퍼콜 명령어를 호출함으로써 하이퍼바이저 예외가 발생하도록 한다. 하이퍼바이저 예외가 예외 벡터에 의해 트랩되면 와치포인트가 자동으로 비활성화된다(***특성5***). 즉, *sec_func*는 와치포인트를 명시적으로 비활성화하지 않고도 쓰기 작업을 에뮬레이션할 수 있다. 와치포인트에 의한 감시가 비활성화되면 2단계 페이지 테이블에 대한 쓰기 감시 또한 비활성화되므로 테이블을 업데이트 가능하다. 와치포인트 감시는 비특권 영역으로 복귀하기 전에 다시 활성화된다.

비특권 영역에서는 운영체제의 하이퍼바이저 호출에 의해 발생한 예외(하이퍼콜 예외)를 처리한다. 하이퍼콜은 커널 무결성 보호를 위해 하이퍼바이저에 의해 에뮬레이션 되는 운영체제 고유의 작업들에 대한 실행 요청을 위해 호출된다(예: 운영체제의 시스템 레지스터 값 갱신 등). 운영체제의 요청을 처리 중 특권이 필요한 작업이 수행되어야 할 때 하이퍼콜 호출을 통해 특권 영역으로 진입하게 된다. 가령, 커널로 복귀하기 위한 *hyp_exit*나 2단계 페이지 테이블 갱신을 위한 *sec_func*이 하이퍼콜을 통해 호출된다. 특권 영역에 진입하기 위한 고유한 수단을 하이퍼콜로 제한함으로써 특권 영역에 대한 공격 벡터를 줄일 수 있다. 또한, 비특권 영역으로부터의 요청은 특권영역의 예외 벡터에 의해 검증된 후 특권영역의 서비스로 전달된다.

Ⅴ. 고 찰

5.1. 한계점

제안된 기법들의 실용성은 와치포인트의 가용성에 의존적이다. 스레드 내 보안 메모리 구성 기법의 경우, 이전 연구[25], [26]에서보다 적은 16개의 보안 슬롯만이 제공될 수 있다. 특권모드 접근 방지 기술의 경우, 리눅스 */proc* 디렉토리의 *map* 파일을 참고하여 모든 프로세스 메모리 매핑을 확인한 결과 각각 2GB를 모니터링하는 최대 3개의 와치포인트가 있으면 충분히 구현할 수 있다. 그러나 여전히 어플리케이션이나 OS 유형에 따라 더 많은 와치포인트가 필요할 수 있다. 이외에도 와치포인트가 충분히 제공되지 않는 경우 다른 하드웨어나 소프트웨어 기술을 혼합하여 사용해야 할 수도 있다. 가령, 2단계 페이징과 와치포인트를 병행하여 사용하면 16개 이상의 보안 슬롯을 제공할 수 있지만, 와치포인트만 사용하는 경우보다

훨씬 더 많은 성능 저하가 예상된다.

5.2. 타 아키텍처 적용 가능성

x86 아키텍처 또한 디버그 레지스터(DR0-DR7)를 제공함으로써 자가 디버깅을 지원한다. DR0-DR3은 중단점 값 레지스터이며, DR6는 디버그 상태 레지스터, DR7는 디버그 컨트롤 레지스터이다. DR4-DR5는 더는 사용되지 않는다. 불행하게도 가능한 모니터링 범위의 최대 크기는 각 레지스터당 8바이트로 제한된다. 그러므로, 본 고에서 소개된 와치포인트 보안 기능을 구현하기엔 적합하지 않다. 하지만, x86 아키텍처에서는 메모리 영역을 16개로 분할하고 각 영역에 대한 접근을 선택적으로 활성화 및 비활성화할 수 있는 메모리 보호 키(MPK, Memory Protection Key)[49]를 제공하기 때문에, 비슷한 보안 기능을 구현 가능할 것으로 예상된다.

RISC-V는 현재 보안 기능을 포함한 다양한 하드웨어의 사양 표준화를 진행 중이다. 제안되고 있는 또는 이미 정식 사양으로 승인된 보안 기능 중 프로세스 내부 메모리 격리와 관련된 기능은 PMP[50], ePMP[51], sPMP[52], AP-TEE[53], H-extension[50] 등이 있다. PMP는 물리 메모리 주소를 기반으로 한 접근제어를 수행한다. Intel CPU에서 특권모드의 유저 영역 접근을 제어하는 기능인 SMAP(Supervisor Mode Access Prevention)과 특권모드의 유저 영역 명령어 실행을 제어하는 SEMP(Supervisor Mode Execution Protection), 그리고 ARM 아키텍처에서의 비슷한 기능을 제공하는 특권모드 접근제어를 위한 PAN(Privileged access-never)과 실행 제어를 위한 PXN(Privileged execute-never)은 상위 권한을 가진 모드에서 그보다 낮은 권한을 가진 모드로의 접근 및 실행을 제어한다. 이와 비슷한 기능이 RISC-V에서는 ePMP(Enhance PMP)라고 명명된 기술로 제공되며 RISC-V의 특권 실행 모드 중 하나인 머신모드(M-mode)에서 유저 모드(U-mode)로의 접근을 제어하는 기능을 제공한다. sPMP(S-mode PMP)는 머신 모드에서 사용할 수 있는 PMP와 다르게, 커널 모드(S-mode) 전용으로 사용할 수 있는 PMP이다. ARM의 TrustZone, Intel의 SGX와 같이 RISC-V 환경에서 신뢰 실행 환경을 구축하기 위한 하드웨어 확장인 AP-TEE(Application-Processor Trusted Execution Environment)도 개발중에 있다. H-extension은 RISC-V의 가상화 확장 지원이다. 이외에도 머신 모드 내부의 기능을 분리하는 M-mode Isolation[54]과 하나의 시스템을 여러 도메인으로 분리하는 도메인 격리 기술에 대한 논의가 활발히 진행되고 있다[55]. 이러한 보안 기능을 활용하여(예: PMP를 활용한 프로세스 내부 메모리 격리) 본 고에서 소개된 보안 기술들이 구현될 수 있을 것으로 예상된다.

Ⅵ. 결 론

본 고에서는 ARM 아키텍처에서 디버그 와치포인트를 활용하여 소프트웨어의 보안성을 향상시킬 수 있는 다양한 연구들을 소개하였다. 와치포인트는 신뢰연산기술과 호환되기 때문에 소개된 기술들을 신뢰(기밀)연산 기술에 의해 보호되는 소프트웨어의 보안성 재고를 위해 적용 가능할 것으로 예상된다. 또한, 와치포인트가 범용 하드웨어라는 특성을 살려 다양한 기기에서 비슷한 보안기술들을 확장 구현 가능할 것이다. 특히, 디버깅, 성능측정 등의 하드웨어 기능들은 ARM, Intel, RISC-V 등 다양한 아키텍처에서 공통적으로 제공되고 있다는 사실은 고무적이다. 향후, 보안기술 구현 관점에서 가치가 있는 범아키텍처 하드웨어 기능들을 발굴하고 새롭게 활용하기 위한 노력이 요구된다.

참 고 문 헌

[1] Sun, He, et al. "Trustice: Hardware-assisted isolated computing environments on mobile devices." 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. IEEE, 2015.

[2] Jang, Jinsoo, et al. "Privatezone: Providing a private execution environment using arm trustzone." IEEE Transactions on Dependable and Secure Computing 15.5 (2016): 797-810.

[3] Cho, Yeongpil, et al. "{Hardware-Assisted}{On-Demand} Hypervisor Activation for Efficient Security Critical Code Execution on Mobile Devices." 2016 USENIX Annual Technical Conference (USENIX ATC 16). 2016.

[4] Brasser, Ferdinand, et al. "SANCTUARY: ARMi

ng TrustZone with User-space Enclaves." NDSS. 2019.

[5] Sun, Lizhi, et al. "LEAP: TrustZone Based Developer-Friendly TEE for Intelligent Mobile Apps." IEEE Transactions on Mobile Computing (2022).

[6] Azab, Ahmed M., et al. "SKEE: A lightweight Secure Kernel-level Execution Environment for ARM." NDSS. Vol. 16. 2016.

[7] Cho, Yeongpil, et al. "Dynamic Virtual Address Range Adjustment for Intra-Level Privilege Separation on ARM." NDSS. 2017.

[8] Brasser, Ferdinand, et al. "Regulating arm trustzone devices in restricted spaces." Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services. 2016.

[9] Lentz, Matthew, et al. "Secloak: Arm trustzone-based mobile peripheral control." Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services. 2018.

[10] Azab, Ahmed M., et al. "Hypervision across worlds: Real-time kernel protection from the arm trustzone secure world." Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. 2014.

[11] Jang, Jin Soo, et al. "Secret: Secure channel between rich execution environment and trusted execution environment." NDSS. 2015.

[12] Mo, Fan, et al. "Darknetz: towards model privacy at the edge using trusted execution environments." Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services. 2020.

[13] Han, Seung-Kyun, and Jinsoo Jang. "MyTEE: Own the Trusted Execution Environment on Embedded Devices." NDSS. 2023.

[14] McCune, Jonathan M., et al. "TrustVisor: Efficient TCB reduction and attestation." 2010 IEEE Symposium on Security and Privacy. IEEE, 2010.

[15] Chen, Xiaoxin, et al. "Overshadow: a virtualization-based approach to retrofitting protection in commodity operating systems." ACM SIGOPS Operating Systems Review 42.2 (2008): 2-13.

[16] Hofmann, Owen S., et al. "Inktag: Secure applications on an untrusted operating system." Proceedings of the eighteenth international conference on Architectural support for programming languages and operating systems. 2013.

[17] (2023. 9월) https://www.arm.com/technologies/trustzone-for-cortex-a

[18] (2023. 9월) https://www.intel.com/content/www/us/en/architecture-and-technology/software-guard-extensions.html

[19] https://github.com/riscv-non-isa/riscv-ap-tee

[20] Costan, Victor, Ilia Lebedev, and Srinivas Devadas. "Sanctum: Minimal hardware extensions for strong software isolation." 25th USENIX Security Symposium (USENIX Security 16). 2016.

[21] Bahmani, Raad, et al. "{CURE}: A security architecture with {CUstomizable} and resilient enclaves." 30th USENIX Security Symposium (USENIX Security 21). 2021.

[22] Weiser, Samuel, et al. "Timber-v: Tag-isolated memory bringing fine-grained enclaves to risc-v." NDSS. 2019.

[23] Lee, Dayeol, et al. "Keystone: An open framework for architecting trusted execution environments." Proceedings of the Fifteenth European Conference on Computer Systems. 2020.

[24] (2023, 9월) The heartbleed bug. [Online]. Available: http://heartbleed.com/

[25] Liu, Yutao, et al. "Thwarting memory disclosure with efficient hypervisor-enforced intra-domain isolation." Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. 2015.

[26] Chen, Yaohui, et al. "Shreds: Fine-grained execution units with private memory." 2016 IEEE Symposium on Security and Privacy (SP). IEEE, 2016.

[27] (2023, 9월) https://developer.arm.com/documentation/100442/0100/debug-registers/aarch64-debug-registers/mdscr-el1--monitor-debug-system-control-register--el1

[28] (2023, 9월) https://developer.arm.com/documentation/ddi0595/2021-03/AArch64-Registers/MDC

R-EL2--Monitor-Debug-Configuration-Register--EL2-

[29] (2023, 9월) https://developer.arm.com/documentation/ddi0595/2021-06/AArch64-Registers/DBGWVR-n--EL1--Debug-Watchpoint-Value-Registers

[30] (2023, 9월) https://developer.arm.com/documentation/ddi0595/2021-06/AArch32-Registers/DBGWCR-n---Debug-Watchpoint-Control-Registers

[31] Jang, Jinsoo, and Brent Byunghoon Kang. "In-process memory isolation using hardware watchpoint." Proceedings of the 56th Annual Design Automation Conference 2019. 2019.

[32] Jang, Jinsoo, and Brent Byunghoon Kang. "Revisiting the arm debug facility for os kernel security." Proceedings of the 56th Annual Design Automation Conference 2019. 2019.

[33] Jang, Jinsoo, and Brent Byunghoon Kang. "SelMon: reinforcing mobile device security with self-protected trust anchor." Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services. 2020.

[34] Bittau, Andrea, et al. "Wedge: Splitting applications into reduced-privilege compartments." USENIX Association, 2008.

[35] (2023, 9월) Arm architecture reference manual armv8, for armv8-a architecture profile. [Online]. Available: https://developer.arm.com/docs/ddi0487/latest/arm-architecture-reference-manual-armv8-for-armv8-a-architecture-profile

[36] (2023, 9월) Efficient application processors for every level of performance. [Online]. Available: https://www.arm.com/products/processors/cortex-a

[37] (2023, 9월) Exploit methods/userspace data usage. [Online]. Avail- able: https://kernsec.org/wiki/index.php/Exploit Methods/Userspacedata usage

[38] Snow, Kevin Z., et al. "Just-in-time code reuse: On the effectiveness of fine-grained address space layout randomization." 2013 IEEE symposium on security and privacy. IEEE, 2013.

[39] Backes, Michael, et al. "You can run but you can't read: Preventing disclosure exploits in executable code." Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. 2014.

[40] Chen, Yaohui, et al. "NORAX: Enabling execute-only memory for COTS binaries on AArch64." 2017 IEEE Symposium on Security and Privacy (SP). IEEE, 2017.

[41] Crane, Stephen, et al. "Readactor: Practical code randomization resilient to memory disclosure." 2015 IEEE Symposium on Security and Privacy. IEEE, 2015.

[42] Werner, Jan, et al. "No-execute-after-read: Preventing code disclosure in commodity software." Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security. 2016.

[43] Tang, Adrian, Simha Sethumadhavan, and Salvatore Stolfo. "Heisenbyte: Thwarting memory disclosure attacks using destructive code reads." Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. 2015.

[44] Brookes, Scott, et al. "Exoshim: Preventing memory disclosure using execute-only kernel code." Proceedings of the 11th International Conference on Cyber Warfare and Security. 2016.

[45] Seshadri, Arvind, et al. "SecVisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity OSes." Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles. 2007.

[46] Payne, Bryan D., et al. "Lares: An architecture for secure active monitoring using virtualization." 2008 IEEE Symposium on Security and Privacy (sp 2008). IEEE, 2008.

[47] (2023, 9월) Real-time kernel protection (rkp). [Online]. Available: https://www.samsungknox.com/pt-br/blog/real-time-kernel-protection-rkp

[48] (2023, 9월) Lifting the (hyper) visor: Bypassing samsungs real-time kernel protection. [Online]. Available: https://googleprojectzero.blogspot.com/2017/02/lifting-hyper-visor-bypassing-samsungs.html

[49] (2023, 9월) Memory protection keys. https://lwn.

net/Articles/643797/
[50] (2023, 9월) https://wiki.riscv.org/display/HOME/RISC-V+Technical+Specifications
[51] (2023, 9월) https://github.com/riscv/riscv-tee/blob/main/Smepmp/Smepmp.pdf
[52] (2023, 9월) https://github.com/riscv/riscv-spmp
[53] (2023, 9월) https://github.com/riscv-non-isa/riscv-ap-tee
[54] (2023, 9월) https://lists.riscv.org/g/tech-m-mode-isolation
[55] (2023, 9월) https://lists.riscv.org/g/tech-security-model

〈저자소개〉

한 승 균 (Seungkyun Han)
2020년 2월 : 한밭대학교 컴퓨터공학과 졸업
현재 : 충남대학교 컴퓨터융합학부 석·박통합과정(박사과정)
<관심분야> 시스템 보안, 신뢰 실행 환경

장 진 수 (Jinsoo Jang)
종신회원
2007년 8월 : 아주대학교 정보및컴퓨터공학부 졸업
2008년 2월~2012년 8월 : 앰코테크놀러지코리아
2014년 8월 : KAIST 정보보호대학원 석사
2018년 2월 : KAIST 정보보호대학원 박사
현재 : 충남대학교 컴퓨터융합학부 조교수
<관심분야> 시스템 보안, 신뢰 실행 환경