Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.25, NO.4, Aug. 2015 ISSN 2288-2715(Online)
http://dx.doi.org/10.13089/JKIISC.2015.25.4.873

메모리 주소 변환 공격에 대한 스케줄러 기반의 방어 방법*

장 대 희,[1†] 장 진 수,[1] 김 동 욱,[2] 최 창 호,[1] 강 병 훈[1‡]
[1]KAIST 정보보호대학원, [2]삼성전자 소프트웨어센터

Scheduler-based Defense Method against Address Translation Redirection Attack (ATRA)*

Daehee Jang,[1†] Jinsoo Jang,[1] Donguk Kim,[2] Changho Choi,[1]
Brent ByungHoon Kang[1‡]
[1]Graduate School of Information Security, KAIST,
[2]Software R&D Center, Samsung Electronics

요 약

하드웨어 기반 커널 무결성 감시 시스템들은 감시 대상으로부터 물리적으로 완전히 분리된 공간에서 수행되기 때문에 감시 시스템 자체의 안전성을 보장받는다는 장점을 가진다. 반면, 이들 감시 시스템들은 물리 메모리 주소를 기반으로 동작하기 때문에 가상-물리 메모리 주소 변환을 조작하는 공격으로부터 취약하다는 단점도 지닌다. 본 논문은 이러한 단점을 보완하기 위해 주소 변환 조작 공격에 대한 커널 스케줄러 기반의 탐지 기법을 제시한다. 제시된 탐지 기법은 커널 스케줄러가 모든 프로세스의 문맥 교환 시마다 수행된다는 점을 이용하여 프로세스 스케줄 시에 공격 여부를 검증한다. 탐지 시스템은 안드로이드 에뮬레이터와 TizenTV에서 구현되었으며, 실험을 통해 최대 10% 정도의 성능저하만 발생시키면서, 루트킷이 수행하는 주소 변환 공격을 정확히 탐지한다는 것을 확인하였다.

ABSTRACT

Since hardware-based kernel-integrity monitoring systems run in the environments that are isolated from the monitored OS, attackers in the monitored OS cannot undermine the security of monitoring systems. However, because the monitoring is performed by using physical addresses, the hardware-based monitoring systems are vulnerable to Address Translation Redirection Attack (ATRA) that manipulates virtual-to-physical memory translations. To ameliorate this problem, we propose a scheduler-based ATRA detection method. The method detects ATRA during the process scheduling by leveraging the fact that kernel scheduler engages every context switch of processes. We implemented a prototype on Android emulator and TizenTV, and verified that it successfully detected ATRA without incurring any significant performance loss.

Keywords: Kernel integrity, ATRA, Scheduler

I. 서 론

커널의 무결성을 보장하기 위한 하드웨어 기반의 시스템들은 감시 대상으로부터 물리적으로 격리되어 있다는 장점을 가지는 한편, 물리 주소에만 의존해서

접수일(2015년 7월 6일), 수정일(1차: 2015년 8월 4일, 2차: 2015년 8월 11일), 게재확정일(2015년 8월 11일)
* 본 연구는 삼성전자 소프트웨어센터와 공동으로 수행하였습니다.
† 주저자, daehee87@kaist.ac.kr
‡ 교신저자, brentkang@kaist.ac.kr(Corresponding author)

감시를 수행한다는 점 때문에 가상-물리 주소 변환을 조작 또는 변조하는 공격으로부터 취약하다는 단점을 가진다. 예를 들어, 하드웨어 기반 감시 시스템에 의해 sys_open()의 코드가 존재하는 물리 주소 영역이 보호되고 있다면, 공격자는 페이지 테이블 상의 sys_open()에 대한 가상-물리 주소 변환 정보가 mal_sys_open()을 지정하도록 변조할 수 있다. 이 경우, 공격자는 sys_open()에 대한 직접 변조를 하지 않았기 때문에 하드웨어 기반 감시 시스템을 우회하는 동시에, 악의적인 행위를 수행하는 mal_sys_open()이 대신 실행됨을 강제할 수 있다.

본 논문은 이러한 하드웨어 기반 커널 무결성 감시 시스템이 가진 주소 변환 공격에 대한 취약점을 해결하고자 커널 스케줄러 기반의 공격 탐지 기법을 제시한다. 탐지 기법은 공격자가 특정 프로세스의 주소 변환 정보를 변조하더라도, 변조된 내용이 효력을 발휘하기 위해서는 해당 프로세스가 스케줄 되어 실행되어야 한다는 점을 이용한다. 커널의 스케줄러는 모든 프로세스의 문맥 교환에 관여하기 때문에 주소 변환 공격 탐지를 위한 주요 거점이 될 수 있다. 하지만 스케줄러 코드에 삽입되는 공격 탐지 로직은 시스템 성능 저하를 고려하여 최대한 간단하게 작성되어야 한다.

이러한 사항들을 고려하여 안드로이드 에뮬레이터 상에 주소 변환 공격 탐지를 위한 프로토타입을 구현하였다. 프로토타입에 대한 평가를 위해서는 안드로이드 에뮬레이터 수정을 통해 하드웨어 기반 커널 무결성 감시 시스템의 동작을 재현하였으며, 주소 변환 공격을 수행하는 루트킷 또한 작성하였다. 마지막으로 실험을 통해 프로토타입이 루트킷을 정확하게 탐지한다는 것을 확인하는 동시에 성능 저하 또한 크지 않음을 보였다.

본 논문은 다음과 같이 구성되어 있다. 2장에서는 하드웨어 기반 커널 무결성 감시 시스템을 비롯한 관련 지식들에 대해 설명한다. 3장에서는 주소 변환 공격에 대한 탐지 기법을 제시하고, 4장에서는 제안한 기법에 대한 구현 및 평가 결과를 보여준다. 마지막으로 5장에서 결론을 논한다.

II. 사전 지식 및 관련 연구

2.1 커널 루트킷

커널 루트킷은 운영체제의 커널 권한으로 수행되는 악성 코드이다. 커널 권한을 바탕으로 수행되기 때문에 유저 권한의 프로세스를 포함하여 운영체제 상의 모든 객체들을 공격할 수 있다. 주요 악성 행위로는 커널 코드 및 데이터 조작을 통한 특정 프로세스 숨기기, 프로세스의 권한 상승, 키 로깅, 백도어 생성 등을 들 수 있다[1,2]. 또한, 바이러스 백신과 같은 탐지 소프트웨어를 무력화하거나 악성 행위 탐지를 어렵게 할 수 있다.

2.2 하드웨어 기반 커널 무결성 감시

2004년 발표된 Copilot[3]을 시작으로 루트킷으로부터 운영체제 커널의 무결성을 보장하기 위한 하드웨어 기반의 보안 솔루션들이 연구되어 왔다. PCI 카드 상에 구현된 Copilot은 감시 대상 커널의 정적 영역 (커널 코드, 시스템 콜 테이블 등)에 대한 주기적인 무결성 감시를 수행한다. Vigilare[4]와 KI-MON[5]은 독립된 보안 프로세서를 감시 대상 시스템에 삽입함으로써, 낮은 성능 저하만으로도 특정 메모리 영역에서 발생하는 쓰기 이벤트에 대한 버스 트래픽 감시를 가능하게 하였다. 이러한 하드웨어 기반 감시 시스템들은 감시 대상으로부터 완벽히 독립되어 수행됨으로써 감시 주체에 대한 공격으로부터 안전하다는 장점을 가진다.

2.3 메모리 주소 변환 공격 기법

하드웨어 기반 커널 무결성 감시 시스템들의 공통점은 감시 대상 시스템의 물리 메모리 주소를 기반으로 동작한다는 점이다. 이러한 동작 방식은 감시 대상 시스템의 가상 주소에서 물리 주소로의 변환 정보를 담고 있는 페이지 테이블에 대한 공격을 통해 우회 가능하다[6]. Fig. 1.의 예시와 같이 루트킷은 페이지 테이블에 대한 공격을 통해 하드웨어 모니터에 의해 감시되는 주소 영역 0x3000에 대한 아무런 변조 없이 0x7000 주소에 생성된 악의적인 시스템 콜 테이블을 사용가능하다. 이러한 메모리 주소 변환과 관련된 공격 방식은 크게 두 가지 경우로 나누어 볼 수 있다.

첫 번째 경우는, 페이지 테이블을 직접 변조하는 공격이다. 공격자는 메모리 상 페이지 테이블의 내용을 직접 변조하여 가상 주소에서 물리 주소로의 변환을 재구성 할 수 있다. 예를 들어, 정상적인 커널 시스템 함수가 존재하는 물리 주소 대신 악의적 행위를

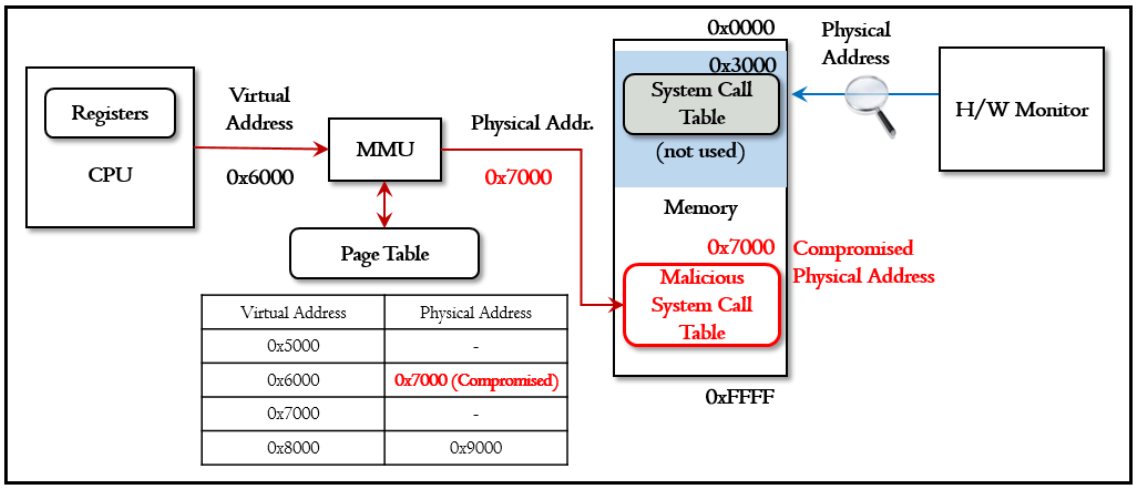

Fig. 1. Address Translation Redirection Attack (ATRA) example

수행하는 함수가 위치하는 물리 주소를 참조하도록 페이지 테이블을 조작할 수 있다. 이를 통해 공격자는 물리 주소를 기반으로 동작하는 하드웨어 기반 외부 모니터의 감시를 피하여 악의적인 함수를 실행할 수 있다.

두 번째 경우는, 레지스터를 변조하는 공격이다. 가상 주소에서 물리 주소로의 변환은 변환 시 참조되는 물리 메모리 블록의 크기나 프로세서의 종류(e.g. ARM 또는 Intel)에 따라 여러 단계의 페이지 테이블을 순차적으로 참조하여 이루어지게 된다. 이때, 최상위 페이지 테이블의 주소는 페이지 테이블 베이스 레지스터 - ARM의 경우 TTBR, Intel의 경우 CR3 - 에 저장된다. 이러한 레지스터들 역시 공격 대상이 될 수 있다. 공격자는 페이지 테이블 베이스 레지스터의 값을 조작하여 악의적으로 생성된 페이지 테이블이 주소 변환 시마다 참조되도록 강제할 수 있다.

2.4 탐지 기법 관련 연구

하이퍼바이저나 트러스트존 기반의 커널 무결성 감시 시스템에서는 메모리 주소 변환 공격에 대한 방어가 고려되고 있다. SecVisor[7]에서 보여진 바와 같이 하이퍼바이저는 기본적으로 게스트 가상 머신에 의한 페이지 테이블이나 베이스 레지스터에 대한 모든 변경 사항을 검증 가능하기 때문에 메모리 주소 변환 공격을 방어하는 것이 가능하다. TZ-RKP[8], Sprobes[9]에서는 페이지 테이블 및 주요 레지스터의 값을 변경하는 명령어들을 감시 대상 운영체제에서 모두 제거하고, 해당 명령어들을 트러스트존에서 대신 수행해 주는 방식을 사용하여 메모리 주소 변환 공격을 방어하였다.

하지만 하드웨어 기반 커널 감시 시스템을 대상으로 한 메모리 주소 변환 공격에 대한 방어 기법은 아직까지 국내외 학계에서 발표된 바가 없기 때문에 본 연구가 최초라고 볼 수 있다.

III. 방어 방법 설계

3.1 공격 모델 분석

페이지 테이블 직접 변조 공격의 경우 하드웨어 기반 감시 시스템으로 하여금 페이지 테이블이 존재하는 메모리 영역에 대한 감시를 수행하게 함으로써 방어 가능하다. 가령, 커널 코드가 존재하는 메모리 영역과 해당 영역에 대한 가상-물리 주소 변환을 위한 페이지 테이블을 주기적으로 감시하거나[3] 해당 메모리 영역에 대한 쓰기 시도를 감지하는[4,5] 방법을 들 수 있다. 그러므로 페이지 테이블 직접 변조 공격은 하드웨어 기반 감시 시스템에 의해 이미 방어 가능하다고 가정한다.

따라서 공격자는 페이지 테이블을 직접 수정하는 대신 레지스터 변조 공격을 수행함으로써 감시 시스템을 우회 할 수 있다. 공격자는 커널 권한을 가지고 있기 때문에 페이지 테이블 베이스 레지스터의 값을 변경하기 위한 어떠한 특권 명령어도 자유롭게 실행

가능하다 (Intel의 CR3나 ARM의 TTBR은 특권 명령어 수행을 통해서만 변경 가능)[10,11]. 이를 통해 공격자는 자신이 악의적으로 생성한 페이지 테이블의 주소를 페이지 테이블 베이스 레지스터의 값으로써 설정할 수 있다. 이후 모든 가상-물리 주소 변환은 공격자가 악의적으로 생성한 페이지 테이블에 의해 이루어지게 된다. 하드웨어 기반 감시 시스템들은 오직 물리 메모리 영역에 대한 감시 수행만 가능하고 레지스터 변조에 대한 감시는 수행 불가능하다. 따라서 레지스터를 활용한 공격 방식은 공격자의 관점에서 매우 효과적이라 볼 수 있다.

3.2 방어 방법 적용 환경에 대한 가정

본 논문은 주소 변환을 통해 하드웨어 기반 감시 시스템을 우회하기 위한 레지스터 변조 공격에 대한 방어 방법을 제시한다. 우리가 제안하는 방법은 이러한 목적을 달성하기 위해 아래와 같은 선행 조건을 요구한다.

첫째, 하드웨어 기반 감시 시스템은 감시 대상 운영체제 구조에 대한 사전 정보를 보유하고 있다. 예를 들어, 감시 대상 운영체제가 리눅스일 경우, 감시 시스템은 각 프로세스의 정보를 담고 있는 task_struct 구조체의 내부 구조, 모든 프로세스의 task_struct를 탐색하기 위한 연결 리스트의 시작 주소, 커널 코드 영역의 시작 주소 정보 등에 접근 가능하다[12].

둘째, 페이지 테이블 직접 변조 공격은 하드웨어 기반 감시 시스템에 의해 방어되고 있다고 가정한다. 따라서 커널 코드 영역을 지정하는 페이지 테이블과 메모리에 저장된 각 프로세스의 페이지 테이블 베이스 레지스터 정보는 하드웨어 기반 감시 시스템에 의해 보호된다.

마지막으로, 본 논문의 실험에서 사용된 메모리 주소 변환 공격의 예시와 그에 대한 방어 방법은 ARM 프로세서를 기반으로 구동되는 모바일 기기용 리눅스 운영체제를 바탕으로 구현되었다. 따라서 구현과 관련된 상세 내용은 ARM 프로세서와 리눅스에 의존적이다. 하지만 구현 상세 내용이 달라진다는 점만 제외하면 동일한 방어 방식이 Intel 프로세서 및 리눅스 이외의 운영체제를 사용하는 환경에서도 적용 가능하다.

3.3 스케줄러 기반 공격 탐지

시스템에 생성되어 있는 프로세스들은 커널 스케줄러에 의해 결정된 실행 우선순위를 바탕으로 일정 기간 동안 수행된다. 각 프로세스는 자신만의 페이지 테이블을 가지며, 페이지 테이블에 설정된 가상-물리 주소 변환 정보를 통해 시스템 메모리에 접근하게 된다. 가상-물리 주소 변환은 프로세서가 지원하는 다양한 메모리 블록의 크기(ARM 프로세서의 short-descriptor 형식의 경우, 16MB, 1MB, 64KB, 4KB를 지원)[11]와 이를 활용하는 운영 체제의 구현 내용에 따라 여러 단계의 페이지 테이블 참조를 통해 이루어 질 수 있다. 그리고 가상-물리 주소 변환 과정에서 가장 상위에 참조되는 페이지 테이블의 주소는 각 프로세스의 정보를 저장하는 메모리 상 구조체 - 리눅스의 경우 mm_struct -〉 pgd - 에 저장된다. 실행을 위해 스케줄 되는 프로세스의 pgd 값은 메모리 상 구조체로부터 추출되어 CPU의 페이지 테이블 베이스 레지스터 값으로 설정된다.

우리가 제안하는 레지스터 변조 공격에 대한 방어 방법은 프로세스 스케줄러가 모든 페이지 테이블 베이스 레지스터의 변경에 관여한다는 점을 이용한다. Fig. 2.에서 볼 수 있듯이 스케줄러는 페이지 테이블 베이스 레지스터에 설정되는 값과 메모리에 저장된 값을 비교함으로써 공격 여부를 탐지하게 된다. 여기서 메모리에 저장된 값이란 앞서 언급한 pgd 값으로써, 해당 값은 하드웨어 기반 커널 무결성 감시 시스템에 의해 보호되고 있기 때문에 항상 올바른 값을 저장하고 있다. 다만, pgd에 저장된 값은 가상 주소이며, 페이지 테이블 베이스 레지스터에 설정되는 값은 물리 주소이기 때문에 두 값의 비교 시에 적절한 가상-물리 주소 변환 과정이 필요하다. 또한, 스케줄러 코드는 프로세스 간 문맥 교환 시마다 수행되기 때문에 추가되는 탐지 코드의 복잡도에 따라 시스템 전체 성능이 크게 저하 될 수 있다. 따라서 추

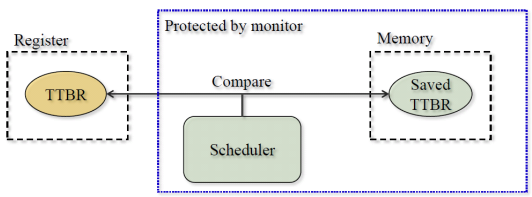

Fig. 2. TTBR manipulation detection through Scheduler-based mechanism

가되는 공격 탐지 코드는 최대한 간단해야 한다. 탐지 코드 구현과 관련된 내용은 4장에서 기술한다.

IV. 구현 및 평가

페이지 테이블 베이스 레지스터 변조 공격과 이에 대한 탐지 코드는 Nexus 5 안드로이드 환경을 제공하는 QEMU 기반의 에뮬레이터 상에서 구현되었다[13]. 하드웨어 기반 감시 시스템의 동작을 재현하기 위해서는 QEMU의 android/main.c 파일을 수정하여 TCP 방식을 통해 외부와 통신 가능한 쓰레드를 생성하였다. 해당 쓰레드는 읽어 들일 메모리 영역의 물리 주소 및 크기를 입력 받아 메모리 덤프를 생성하고 외부로 전송하게 된다. 쓰레드와의 통신을 위한 쉘 프로그램은 파이썬 코드를 이용하여 작성하였다.

4.1 탐지 시스템 구현

탐지 시스템은 ARM 프로세서를 기준으로 구현되었다. ARM의 페이지 테이블 베이스 레지스터인 TTBR은 Fig. 3.과 같은 구조를 가지고 있다. TTBR에 저장된 값은 물리 주소이며 하위 6비트에는 메모리에 대한 속성이 저장되어 있다. 따라서 가상 주소 기반의 pgd 값과 TTBR을 비교하기 위해서는 별도의 변환 과정이 요구된다.

Fig. 4.는 공격 탐지를 위해 삽입된 코드를 나타낸다. 먼저 MCR 명령 수행을 통해 TTBR에 저장된 값을 추출한다. 그리고 추출된 값에서 하위 6비트에 해당하는 속성 및 Reserved 값을 제거한 뒤에 PAGE_OFFSET을 더해준다. 이렇게 계산된 값은 메모리에 저장된 mm->pgd 값과 비교됨으로써 공격 탐지에 사용된다. 정상적인 경우라면 두 값은 동일해야 한다. 따라서 두 값이 다를 경우는 탐지 코드에 의해 공격 시도로 간주된다. 이러한 일련의 탐지 로직은 switch_mm 함수에 삽입되어 프로세스 문맥 교환이 발생하기 직전마다 수행된다.

프로토타입에서는 탐지 시스템 구현을 위해 감시 대상의 커널 코드를 직접 수정하였다. 하지만, 실제 시스템에 적용되기 위해서는 커널 코드 변경을 최소화 할 필요가 있다. 이를 위해, 커널 함수 후킹을 수행하는 동적 커널 모듈 (Loadable Kernel Module: LKM)을 활용한 탐지 로직 삽입을 고려해 볼 수 있다.

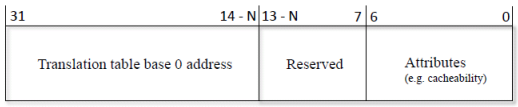

Fig. 3. Translation Table Base Register0 (TTBR0) format

```
static inline void check_atra(struct mm_struct *next){
        unsigned int ttbr0 = 0;
        __asm__("mrc p15, 0, %0, c2, c0, 0":"=r"(ttbr0));
        unsigned int ttb_base = ttbr0 & 0xFFFFF000;
        unsigned int s_ttbr = current->mm->pgd;

        if( current->mm == next ) return;       // stack already switched.

        // compare saved-ttbr to current ttbr
        if( virt_to_phys(s_ttbr) != ttb_base ){
                printk("ATRA detected!(%08x != %08x)\n", virt_to_phys(s_ttbr), ttb_base);
        }
}
```

Fig. 4. Register-based ATRA detection code

4.2 보안성 평가

패치된 스케줄러 코드의 공격 탐지 여부를 평가하기 위해서, TTBR 값을 임의의 값으로 변조하는 루트킷을 작성하였다. 루트킷은 LKM 형태로 작성 되었으며 MCR 명령어를 이용하여 TTBR의 값을 특정 값으로 변조한다.

Fig. 5.는 실험 과정에서 발생한 커널 메시지를 나타낸다. 루트킷은 현재 프로세스의 TTBR 값인 0x28b68059를 0x25300059로 바꾸고, 변경 사항을 시스템에 반영하기 위해 변환 색인 버퍼(TLB)의 값을 갱신한다. 이러한 루트킷의 공격 행위는 스케줄러에 삽입된 TTBR과 pgd의 비교 로직에 의해 정상적으로 탐지됨을 확인하였다.

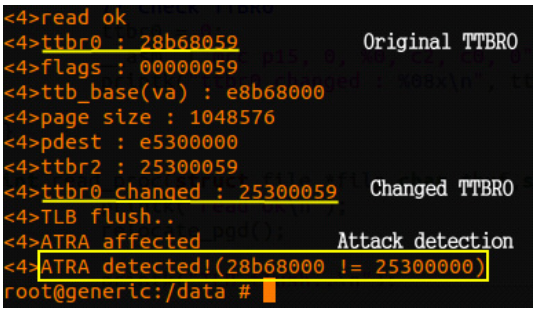

Fig. 5. Register-bound ATRA detection through scheduler-based detection mechanism

4.3 성능 평가

안드로이드의 성능 측정 시에 주로 사용되는 Oxbench[14]를 사용하여 스케줄러 코드에 삽입된

Table 1. Result of microbenchmark measured in Android emulator

Benchmark Type	Original Kernel	Patched Kernel	Overhead
execve()	2862 [1)]LPS	2742 LPS	4.19%
System Call Throughput	424027 LPS	381709 LPS	9.98%
CPU Microbench	2.21 Sec.	2.44 Sec.	10.40%

Table 2. Result of macrobenchmark measured in TizenTV

Benchmark Type	Original Kernel	Patched Kernel	Overhead
CoreMark	4189.9 Score	4152.2 Score	0.90%
Streamline	3.128 CPU usage	3.144 CPU usage	0.51%

공격 탐지 코드의 실행으로 인한 성능 저하를 측정하였다. Table 1.는 execve(), System Call, CPU Microbench의 3가지 항목에 대한 성능 측정 결과를 나타낸다. 탐지 코드 삽입에 의해 5~10% 정도의 성능 저하가 나타남을 확인 할 수 있었다.

또한, 그래픽과 네트워크 접근 등과 같은 일반적인 시스템 이벤트들이 존재할 경우의 성능 저하를 측정하기 위해 공격 탐지 코드를 삼성 TizenTV에 포팅하고 성능을 측정하였다. 실험에 사용된 Tizen TV는 4개의 ARMv7 CPU를 지원하며, 운영체제로써 리눅스 커널 3.10.30 버전을 사용하였다.

Table 2.는 2개의 벤치마크 - CoreMark[15]와 Streamline[16] - 를 TizenTV에서 수행한 결과 값의 평균을 비교한다. 두 경우 모두 1% 미만의 성능 저하를 보임으로써, 일반적인 시스템 이벤트들과 함께 측정된 탐지 코드 수행에 의한 성능 저하는 크지 않음을 알 수 있었다.

V. 결 론

본 논문은 하드웨어 기반 커널 무결성 감시 시스템의 약점을 보완하기 위해 주소 변환 공격에 대한 탐지 기법을 제시한다. 제시된 기법은 프로세스 스케줄 시마다 페이지 테이블 베이스 레지스터에 대한 변조 여부를 검증함으로써 공격 여부를 탐지해낸다. 구현 및 평가는 안드로이드 에뮬레이터와 TizenTV 상에서 이루어졌으며, 최대 10% 정도의 성능 저하만으로 주소 변환 공격을 수행하는 루트킷을 탐지 가능함을 확인하였다.

1) LPS: loop-per-seconds

References

[1] O.S. Hofmann, A.M. Dunn, Sangman Kim, I. Roy, and E.Witchel, "Ensuring operating system kernel integrity with OSck," Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems, pp. 279-290, Mar. 2011.

[2] Y. Liu, Y. Xia, H. Guan, B. Zang, and H. Chen, "Concurrent and consistent virtual machine introspection with hardware transactional memory," 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA), pp. 416-427, Feb. 2014.

[3] N.L. Petroni Jr., T. Fraser, J. Molina, and W.A. Arbaugh, "Copilot - a coprocessor-based kernel runtime integrity monitor," Proceedings of the 13th USENIX Security Symposium, pp. 179-194, Aug. 2004.

[4] Hyungon Moon, Hojoon Lee, Jihoon Lee, Kihwan Kim, Yunheung Paek, and Brent Byunghoon Kang, "Vigilare: toward snoop-based kernel integrity monitor," Proceedings of the 2012 ACM conference on Computer and communications security, pp. 28-37, Oct. 2012.

[5] Hojoon Lee, Hyungon Moon, Daehee Jang, Kihwan Kim, Jihoon Lee, Yunheung Paek, and Brent Byunghoon Kang, "KI-Mon: A hardware-assisted event-triggered monitoring platform for

mutable kernel object," Proceedings of 22nd USENIX Security Symposium, pp. 511-526, Aug. 2013.

[6] Daehee Jang, Hojoon Lee, Minsu Kim, Daehyeok Kim, Daegyeong Kim and Brent Byunghoon Kang, "ATRA: Address translation redirection attack against hardware-based external monitors." Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pp. 167-178, Nov. 2014.

[7] A. Seshadri, M. Luk, N. Qu, and A. Perrig, "SecVisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity OSes," Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles, pp. 335-350, Dec. 2007.

[8] A.M. Azab, P. Ning, J. Shah, Q. Chen, R. Bhutkar, G. Ganesh, J. Ma, and W. Shen, "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, pp. 90-102, Nov. 2014.

[9] X. Ge, H. Vijayakumar, and T. Jaeger, "Sprobes: enforcing kernel code integrity on the TrustZone architecture," Proceedings of the Third Workshop on Mobile Security Technologies (MoST) 2014, May. 2014.

[10] INTEL, INC. "Intel R 64 and IA-32 Architectures Software Developer's Manual," Volume 3b: System Programming Guide (Part 2), pp. 14-19, 2013.

[11] ARM, "Architecture reference manual (ARMv7-A and ARMv7-R edition)." ARM DDI C, 406, 2008.

[12] D.P. Bovet and M. Cesati, "Understanding the Linux kernel," O'Reilly Media, Inc., 2005.

[13] Android, "Building Kernels," https://source.android.com/source/building-kernels.html

[14] 0xbench, "Comprehensive Benchmark Suite for Android," https://code.google.com/p/0xbench/wiki/Benchmarks

[15] CoreMark, "Industry-Standard Benchmarks for Embedded Systems," http://www.eembc.org/coremark/download_coremark.php

[16] ARMDS, "Streamline Performance Analyzer," http://ds.arm.com/ds-5/optimize/

〈저 자 소 개〉

장 대 희 (Daehee Jang) 학생회원
2012년 2월: 한양대학교 컴퓨터공학과 졸업
2014년 2월: KAIST 정보보호대학원 석사
2014년 9월~현재: KAIST 정보보호대학원 박사과정
〈관심분야〉 시스템 보안

장 진 수 (Jinsoo Jang) 학생회원
2007년 8월: 아주대학교 정보및컴퓨터공학부 졸업
2008년 2월~2012년 8월: 앰코테크놀러지코리아 전산팀
2014년 8월: KAIST 정보보호대학원 석사
2014년 9월~현재: KAIST 정보보호대학원 박사과정
〈관심분야〉 시스템 보안

김 동 욱 (Donguk Kim) 정회원
2007년 8월: 고려대학교 산업시스템정보공학과 졸업
2007년 7월~2008년 8월: 한국생산성본부 연구원
2014년 2월: KAIST 산업및시스템공학과 박사
2014년 3월~현재: 삼성전자 소프트웨어센터 재직
〈관심분야〉 정보보호, 시스템 보안

최 창 호 (Changho Choi) 학생회원
2012년 8월: 한동대학교 전산전자공학부 졸업
2014년 8월: KAIST 정보보호대학원 석사
2014년 9월~현재: KAIST 정보보호대학원 박사과정
〈관심분야〉 시스템 보안

강 병 훈 (Brent Byunghoon Kang) 종신회원
1993년: 서울대학교 컴퓨터공학과 학사
1995년: 美 메릴랜드 주립 대학교 컴퓨터 공학 석사
2004년: 美 UC 버클리대학교 컴퓨터 공학 박사
2010년~2013년: 美 조지 메이슨 대학교 부교수
2013년~현재: KAIST 정보보호대학원 부교수
〈관심분야〉 시스템 보안, 신뢰 실행 환경