(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2025-0023315
(43) 공개일자 2025년02월18일

(51) 국제특허분류(Int. Cl.)
G06F 21/55 (2013.01) *G06N 3/0442* (2023.01)
H04L 9/00 (2022.01)
(52) CPC특허분류
G06F 21/55 (2013.01)
G06N 3/0442 (2023.01)
(21) 출원번호 10-2024-0106753
(22) 출원일자 2024년08월09일
심사청구일자 2024년08월09일
(30) 우선권주장
1020230104151 2023년08월09일 대한민국(KR)

(71) 출원인
고려대학교 산학협력단
서울특별시 성북구 안암로 145, 고려대학교 (안암동5가)
(72) 발명자
김호동
서울특별시 성북구 안암로 145, 고려대학교안암캠퍼스 과학도서관 618A 정보시스템보안연구실
허준범
경기도 용인시 기흥구 보정로 30, 119동 201호
(74) 대리인
김등용

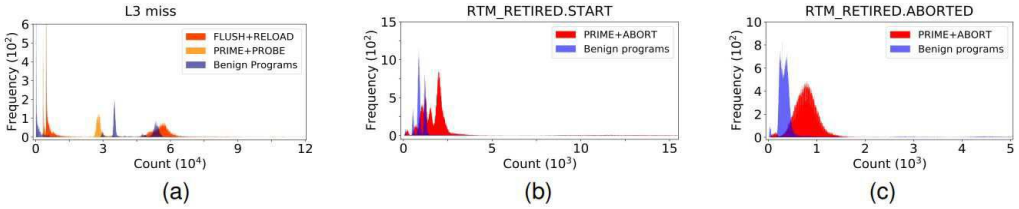
전체 청구항 수 : 총 6 항

(54) 발명의 명칭 **다종 캐시 부채널 공격 실시간 탐지를 위한 장치 및 방법**

(57) 요 약

캐시 부채널 공격(Cache side-channel attack) 실시간 탐지 방법이 개시된다. 캐시 부채널 공격 실시간 탐지 방법은 적어도 프로세서를 포함하는 컴퓨팅 장치에 의해 수행되고, 타겟 하드웨어 이벤트 정보가 획득되는 단계, 미리 학습된 딥러닝 기반의 탐지 모델의 입력 데이터가 생성되는 단계, 및 상기 탐지 모델의 출력에 기초하여, 캐시 부채널 공격 여부를 탐지하는 단계를 포함하고, 상기 타겟 하드웨어 이벤트 정보는, 미리 정해진 시간 구간 동안 미리 정해진 시간 간격으로 측정된 카운트 정보이되, L3 캐시 미스, unhalted cycle, INST_RETIRED.ANY, RTM_RETIRED.START, 및 RTM_RETIRED.ABORTED의 카운트 정보인 것을 특징으로 한다.

대 표 도 - 도6



(52) CPC특허분류
H04L 9/003 (2013.01)
G06F 2207/7219 (2013.01)

이 발명을 지원한 국가연구개발사업

과제고유번호	1711193697
과제번호	2022-0-00411-002
부처명	과학기술정보통신부
과제관리(전문)기관명	정보통신기획평가원
연구사업명	정보보호핵심원천기술개발(R&D,정보화)
연구과제명	내용기반 중요정보 분류 및 등급별 보안서비스 기술 개발
과제수행기관명	고려대학교산학협력단
연구기간	2023.01.01 ~ 2023.12.31

명 세 서

청구범위

청구항 1

적어도 프로세서를 포함하는 컴퓨팅 장치에 의해 수행되는, 캐시 부채널 공격(Cache side-channel attack) 실시간 탐지 방법에 있어서,

타겟 하드웨어 이벤트 정보가 획득되는 단계;

미리 학습된 딥러닝 기반의 탐지 모델의 입력 데이터가 생성되는 단계; 및

상기 탐지 모델의 출력에 기초하여, 캐시 부채널 공격 여부를 탐지하는 단계를 포함하고,

상기 타겟 하드웨어 이벤트 정보는, 미리 정해진 시간 구간 동안 미리 정해진 시간 간격으로 측정된 카운트 정보이되, L3 캐시 미스, unhalted cycle, INST_RETIRED.ANY, RTM_RETIRED.START, 및 RTM_RETIRED.ABORTED의 카운트 정보인,

캐시 부채널 공격 실시간 탐지 방법.

청구항 2

제1항에 있어서,

상기 탐지 모델은 상기 타겟 하드웨어 이벤트 정보를, FLUSH+RELOAD 공격, PRIME+PROBE 공격, PRIME+ABORT 공격, 및 정상 프로그램 동작 중 어느 하나로 분류하는,

캐시 부채널 공격 실시간 탐지 방법.

청구항 3

제2항에 있어서,

상기 입력 데이터가 생성되는 단계는, 상기 타겟 하드웨어 이벤트 정보를 강건한 스케일링(robust scaling)하여 상기 입력 데이터를 생성하는,

캐시 부채널 공격 실시간 탐지 방법.

청구항 4

제3항에 있어서,

상기 탐지 모델은 학습 데이터를 이용하여 학습된 LSTM(Long Short-Term Memory, 장단기 메모리) 모델인,

캐시 부채널 공격 실시간 탐지 방법.

청구항 5

제4항에 있어서,

상기 타겟 하드웨어 이벤트 정보가 획득되는 단계는, 하드웨어 성능 카운터(Hardware performance counter)를 이용하여 상기 타겟 하드웨어 이벤트 정보를 획득하는,

캐시 부채널 공격 실시간 탐지 방법.

청구항 6

제5항에 있어서,

상기 L3 캐시 미스, 상기 unhalted cycle, 및 상기 INST_RETIRED.ANY에 대한 카운트 정보는 고정 기능 카운터(Fixed-function counter)를 이용하여 획득되고,

상기 RTM_RETIRED.START 및 상기 RTM_RETIRED.ABORTED에 대한 카운트 정보는 프로그래머블 카운터(Programmable counter)를 이용하여 획득되는,

캐시 부채널 공격 실시간 탐지 방법.

발명의 설명

기 술 분 야

[0001] 본 발명은 다종 캐시 부채널 공격들을 탐지하는 방법에 관한 것으로, 특히 서로 다른 하드웨어 이벤트를 활용하여 다종의 캐시 부채널 공격을 실시간으로 탐지하는 장치 및 방법에 관한 것이다.

배 경 기 술

[0002] 캐시 부채널 공격(Cache Side-Channel Attack)은, 프로그램이 실행되는 도중에 캐시에서 발생하는 흔적을 획득 및 분석하여 정보를 획득하는 공격을 의미한다. 예를 들어, 캐시는 CPU가 프로그램을 실행하기 위해 데이터를 적재하는 공간으로, 접근하고자 하는 데이터가 캐시에 존재할 경우 더 짧은 시간에 접근가능한 캐시 히트(cache hit)가 발생하고, 존재하지 않을 경우 더 긴 시간이 소요되는 캐시 미스(cache miss)가 발생한다. 여러 캐시 부채널 공격들은 이와 같은 경우에 발생하는 시간차를 분석하여 공격에 활용할 수 있고, 대표적으로 플러시+리로드(Flush+Reload)와 프라임+프로브(Prime+Probe) 유형으로 분류될 수 있다. 최근의 캐시 부채널 공격들은 CPU 아키텍처의 다양한 하드웨어 이벤트들을 활용하여 암호화 비밀키, 인터넷 브라우저 사용 내역, 방화벽 규칙, 휴대폰 잠금 패턴 등의 다양한 비밀 정보를 공격 목표로 한다.

[0003] Flush+Reload 공격은 캐시 부채널 공격의 한 종류로, 공격자가 Intel x86 아키텍처의 clflush 명령을 이용하여 감시하고자 하는 위치의 캐시 라인(cache line)의 데이터를 비우는 플러시(Flush) 과정과 공격자가 다시 감시하고자 하는 데이터를 액세스하고 소요 시간을 측정하는 리로드(Reload) 과정으로 이루어 진다. 두 과정 사이에 감시 당하고 있는 위치에 희생자가 접근하여 데이터를 적재하였을 때, 공격자는 해당 위치에 캐시 미스 없이 접근하게 되어 희생자의 접근이 없었을 때와 비교하여 짧은 시간을 소모하게 된다. 공격자는 희생자의 접근여부에 따라 발생하는 리로드(Reload) 과정에 소요되는 시간 차이를 분석하여 희생자의 실행 흐름을 파악할 수 있다.

[0004] Prime+Probe 공격은 캐시 부채널 공격의 한 종류로, 프라임(Prime) 과정과 프로브(Probe) 과정으로 이루어져 있다. 공격자는 프라임(Prime) 과정에서 공격자가 감시하고자 하는 캐시 셋(cache set)에 데이터를 채우고, 프로브(Probe) 과정에서 다시 해당 데이터를 로드하고 시간을 측정한다. 두 과정 사이에 희생자가 캐시에 자신의 데이터를 적재하기 위해 공격자가 채워 놓은 데이터를 비웠을 경우, 공격자가 프로브(Probe) 과정을 수행하는 동안 캐시 미스가 발생하여 더 긴 시간이 소요된다. 이러한 시간차를 분석하여 공격자가 해당 캐시 셋에 대한 희생자의 접근여부를 판단하고 실행 흐름을 파악할 수 있다.

[0005] 프라임+어보트 공격(Prime+Abort attack)은 캐시 부채널 공격의 한 종류로, 프라임(Prime) 과정과 어보트(Abort) 과정으로 이루어져 있다. 프라임(Prime) 과정에서 공격자가 감시하고자 하는 캐시 셋에 Intel Transactional Synchronization Extensions 기술을 활용하여 하드웨어 트랜잭셔널 메모리(transactional memory)에 데이터를 채우고, 어보트(Abort) 과정에서 희생자가 캐시에 데이터를 적재하기 위해 공격자가 채워놓은 데이터를 비우게 될 때 발생하는 transactional abort hardware callback을 이용하여 공격자가 희생자의 실행 흐름을 파악할 수 있다.

[0006] 하드웨어 성능 카운터(Hardware performance counter)는 CPU에 포함되어 있는 특수목적 레지스터로, 실행되고 있는 다양한 프로그램들이 유발하는 하드웨어 이벤트들을 분석하기 위해 사용할 수 있다. 예를 들어, Intel사의 하드웨어 성능 카운터는 고정 기능 카운터(Fixed-function counter)와 프로그래머블 카운터(Programmable counter)로 분류 할 수 있고, 고정 기능 카운터는 미리 정해진 하드웨어 이벤트만을 모니터링하고 프로그래머블 카운터는 사용자가 이벤트 번호(event number) 등을 이용해 모니터링 하고자 하는 하드웨어 이벤트를 설정할 수 있다. 제조사와 모델에 따라 모니터링 가능한 다양한 종류의 하드웨어 이벤트가 존재하나, 탑재된 레지스터의 개수가 한정되어 있기 때문에, 동시에 모니터링 할 수 있는 서로 다른 종류의 이벤트 가짓수가 하드웨어 설계상 제한되어 있다.

[0007] Intel Performance Counter Monitor는 Intel의 Xeon과 CORE 계열 CPU의 활동을 분석하는데 사용 할 수 있는 소프트웨어이다.

[0008] 본 발명에서는, 다른 종류의 하드웨어 이벤트를 활용하는 다종의 캐시 부채널 공격을 단일 탐지기로 구체적인

종류를 구별하여 탐지하고자 한다. 캐시 부채널 공격은 탐지하고자 하는 공격이 발생시키는 하드웨어 이벤트들의 특징을 분석하고, 이를 하드웨어 모니터링 자원을 활용해 탐지하는 방법으로 공격 여부를 판단할 수 있다. 이때, 동시에 모니터링할 수 있는 이벤트의 수가 하드웨어 설계상 제한되어 있어, 각각의 공격 유형 탐지에 유용한 여러 이벤트들을 찾더라도 허용된 수 이상의 다른 종류의 하드웨어 이벤트를 동시에 모니터링 할 수는 없다. 알려진 탐지 기법들은 활용 가능한 모든 자원을 활용해 비슷한 종류의 하드웨어 이벤트로 탐지할 수 있는 잘 알려진 다종의 공격을 탐지하였다. 새롭게 알려진 서로 다른 종류의 하드웨어 이벤트를 활용하는 여러 캐시 부채널 공격 탐지에 관해 분석하고 탐지에 유용한 하드웨어 이벤트를 획득할 수 있더라도, 기존의 탐지 방법들에서 하드웨어 이벤트를 추가적으로 모니터링 할 수 있는 가용 하드웨어 자원이 없기 때문에, 실질적으로 다른 종류의 하드웨어 이벤트를 활용하는 캐시 부채널 공격을 추가적으로 탐지하는 것은 매우 어렵다. 따라서, 알려진 탐지 방법들이 제공할 수 있는 탐지 범위를 희생하지 않고 이를 확장하여 새롭게 등장한 캐시 부채널 공격을 효과적으로 탐지 하는 것은 매우 어렵다. 본 발명에서는 고정 기능 카운터로 모니터링 할 수 있는 하드웨어 이벤트들을 프로그래머블 카운터로 모니터링 할 수 있는 하드웨어 이벤트들과 새로운 관점에서 비교분석하여 다종의 캐시 부채널 공격 탐지에 유용한 정보를 획득함으로써, 기존의 탐지 기법들이 보인 탐지범위 확대에 가용할 수 있는 하드웨어 자원 부족에 관한 한계점을 극복하는 전략을 수립하였고 실험을 통해 효용성을 보인다.

[0009] 캐시 부채널 공격은 다양한 프로그램들이 실행되고 있는 환경에서 이루어질 수 있다. 실제 환경에서 실행 중인 프로그램들은 각 시점에서 예측하기 어려운 수의 하드웨어 이벤트를 유발하며, 캐시 부채널 공격이 유발하는 하드웨어 이벤트와 유사한 패턴의 하드웨어 이벤트가 다른 프로그램들에서 빈번하게 발생할 수 있다. 따라서, 여러 프로그램들이 동시에 실행되는 환경을 충분히 고려하지 않는 탐지 방법들로는 실제 환경에서 캐시 부채널 공격을 효과적으로 탐지하기 어렵다.

선행기술문헌

비특허문헌

[0010] (비특허문헌 0001) Akram, Ayaz, et al. "Meet the Sherlock Holmes' of side channel leakage: A survey of cache SCA detection techniques." IEEE Access 8 (2020): 70836-70860.

발명의 내용

해결하려는 과제

[0011] 본 발명이 이루고자 하는 기술적인 과제는 실시간으로 다종 캐시 부채널 공격을 탐지할 수 있는 장치 및 방법을 제공하는 것이다.

과제의 해결 수단

[0012] 본 발명의 일 실시예에 따른 캐시 부채널 공격(Cache side-channel attack) 실시간 탐지 방법은 적어도 프로세서를 포함하는 컴퓨팅 장치에 의해 수행되고, 타겟 하드웨어 이벤트 정보가 획득되는 단계, 미리 학습된 딥러닝 기반의 탐지 모델의 입력 데이터가 생성되는 단계, 및 상기 탐지 모델의 출력에 기초하여, 캐시 부채널 공격 여부를 탐지하는 단계를 포함하고, 상기 타겟 하드웨어 이벤트 정보는, 미리 정해진 시간 구간 동안 미리 정해진 시간 간격으로 측정된 카운트 정보이되, L3 캐시 미스, unhalted cycle, INST_RETIRED.ANY, RTM_RETIRED.START, 및 RTM_RETIRED.ABORTED의 카운트 정보인 것을 특징으로 한다.

발명의 효과

[0013] 본 발명의 실시예에 따른 다종 캐시 부채널 공격 실시간 탐지를 위한 장치 및 방법에 의할 경우, 다종의 캐시 부채널 공격을 하나의 탐지 모델을 이용하여 실시간으로 탐지할 수 있는 효과가 있다.

도면의 간단한 설명

[0014] 본 발명의 상세한 설명에서 인용되는 도면을 보다 충분히 이해하기 위하여 각 도면의 상세한 설명이 제공된다.

도 1은 전통적인 캐시 부채널 공격과 정상 프로그램에 의해 유발되는 캐시 이벤트의 분포를 도시한다.

도 2는 PRIME+ABORT 공격과 정상 프로그램에 의해 유발되는 하드웨어 이벤트의 분포를 도시한다.

도 3은 상이한 프로그램 실행 환경에서 하드웨어 이벤트의 변화를 도시한다.

도 4는 PRIME+PROBE 공격과 AES 암호화 하에서 하드웨어 이벤트의 변화를 도시한다.

도 5는 제안하는 탐지기의 개요도이다.

도 6은 높은 워크로드 조건에서 FLUSH+RELOAD, PRIME+PROBE, 및 정상 프로그램의, 선택된 하드웨어 이벤트의 분포를 도시한다.

도 7은 본 발명의 일 실시예에 의한 캐시 부채널 공격 실시간 탐지 방법을 설명하기 위한 흐름도이다.

발명을 실시하기 위한 구체적인 내용

[0015] 본 명세서에 개시되어 있는 본 발명의 개념에 따른 실시예들에 대해서 특정한 구조적 또는 기능적 설명들은 단지 본 발명의 개념에 따른 실시예들을 설명하기 위한 목적으로 예시된 것으로서, 본 발명의 개념에 따른 실시예들은 다양한 형태들로 실시될 수 있으며 본 명세서에 설명된 실시예들에 한정되지 않는다.

[0016] 본 발명의 개념에 따른 실시예들은 다양한 변경들을 가할 수 있고 여러 가지 형태들을 가질 수 있으므로 실시예들을 도면에 예시하고 본 명세서에서 상세하게 설명하고자 한다. 그러나, 이는 본 발명의 개념에 따른 실시예들을 특정한 개시 형태들에 대해 한정하려는 것이 아니며, 본 발명의 사상 및 기술 범위에 포함되는 모든 변경, 균등물, 또는 대체물을 포함한다.

[0017] 제1 또는 제2 등의 용어는 다양한 구성 요소들을 설명하는데 사용될 수 있지만, 상기 구성 요소들은 상기 용어들에 의해 한정되어서는 안 된다. 상기 용어들은 하나의 구성 요소를 다른 구성 요소로부터 구별하는 목적으로만, 예컨대 본 발명의 개념에 따른 권리 범위로부터 벗어나지 않은 채, 제1 구성 요소는 제2 구성 요소로 명명될 수 있고 유사하게 제2 구성 요소는 제1 구성 요소로도 명명될 수 있다.

[0018] 어떤 구성 요소가 다른 구성 요소에 "연결되어" 있다거나 "접속되어" 있다고 언급된 때에는, 그 다른 구성 요소에 직접적으로 연결되어 있거나 또는 접속되어 있을 수도 있지만, 중간에 다른 구성 요소가 존재할 수도 있다고 이해되어야 할 것이다. 반면에, 어떤 구성 요소가 다른 구성 요소에 "직접 연결되어" 있다거나 "직접 접속되어" 있다고 언급된 때에는 중간에 다른 구성 요소가 존재하지 않는 것으로 이해되어야 할 것이다. 구성 요소들 간의 관계를 설명하는 다른 표현들, 즉 "~사이에"와 "바로 ~사이에" 또는 "~에 이웃하는"과 "~에 직접 이웃하는" 등도 마찬가지로 해석되어야 한다.

[0019] 본 명세서에서 사용한 용어는 단지 특정한 실시예를 설명하기 위해 사용된 것으로서, 본 발명을 한정하려는 의도가 아니다. 단수의 표현은 문맥상 명백하게 다르게 뜻하지 않는 한, 복수의 표현을 포함한다. 본 명세서에서, "포함하다" 또는 "가지다" 등의 용어는 본 명세서에 기재된 특징, 숫자, 단계, 동작, 구성 요소, 부분품 또는 이들을 조합한 것이 존재함을 지정하려는 것이지, 하나 또는 그 이상의 다른 특징들이나 숫자, 단계, 동작, 구성 요소, 부분품 또는 이들을 조합한 것들의 존재 또는 부가 가능성을 미리 배제하지 않는 것으로 이해되어야 한다.

[0020] 다르게 정의되지 않는 한, 기술적이거나 과학적인 용어를 포함해서 여기서 사용되는 모든 용어들은 본 발명이 속하는 기술 분야에서 통상의 지식을 가진 자에 의해 일반적으로 이해되는 것과 동일한 의미를 가진다. 일반적으로 사용되는 사전에 정의되어 있는 것과 같은 용어들은 관련 기술의 문맥상 가지는 의미와 일치하는 의미를 갖는 것으로 해석되어야 하며, 본 명세서에서 명백하게 정의하지 않는 한, 이상적이거나 과도하게 형식적인 의미로 해석되지 않는다.

[0021] 이하, 본 명세서에 첨부된 도면들을 참조하여 본 발명의 실시예들을 상세히 설명한다. 그러나, 특허출원의 범위가 이러한 실시예들에 의해 제한되거나 한정되는 것은 아니다. 각 도면에 제시된 동일한 참조 부호는 동일한 부재를 나타낸다.

[0022] **캐시 부채널 공격 탐지(CACHE SIDE-CHANNEL ATTACK DETECTION)**

[0023] 이하에서는, 캐시 이벤트 외에도 공격과 관련된 숨겨진 하드웨어 이벤트를 식별하기 위해, FLUSH+RELOAD, PRIME+PROBE, 및 PRIME+ABORT 캐시 부채널 공격에 대한 심층 분석을 수행한다. 분석을 바탕으로, 공격을 효과적으로 탐지하기 위한, 다양한 유형의 이벤트에 대한 최적의 조합을 도출한다. 마지막으로, FLUSH+RELOAD, PRIME+PROBE, 및 PRIME+ABORT 공격을 동시에 탐지할 수 있는 새로운 딥러닝 기반 캐시 부채널 공격 탐지 기법

(FRIME라 명명됨)을 제안한다.

[0024] A. 공격 탐지 범위(Detection Coverage of Attacks)

[0025] 캐시 부채널 공격은 타겟 캐시 레벨(target cache level), 캐시 포함 정책(cache inclusion policy), 프로세서 아키텍처 등과 같은 타겟 시스템 환경에 따라 상이하게 설계될 수 있다. 본 발명에서는, 탐지 메커니즘의 주요 대상으로 Intel CPU 아키텍처의 인클루시브 캐시(inclusive cache) 내 L3 캐시에 대한 FLUSH+RELOAD, PRIME+PROBE, 및 PRIME+ABORT의 세 가지 접근 기반(access-driven) 캐시 부채널 공격에 초점을 맞춘다. 그러나, 본 발명의 권리범위가 CPU 아키텍처의 종류나 캐시 부채널 공격의 유형에 제한되는 것은 아니다.

[0026] FLUSH+RELOAD와 PRIME+PROBE는 종래 캐시 이벤트 기반 탐지 방법의 대부분에서 탐지의 주요 타겟이었다. 따라서, 본 발명에서도 기존의 캐시 부채널 공격에 대한 탐지기 적용 가능성을 보여주면서 탐지 범위를 확장하기 위해 이러한 공격들을 주요 타겟으로 고려한다. 반면, PRIME+ABORT의 동작은 아직 많이 연구되지 않았기 때문에 효과적인 탐지를 위해 캐시 이벤트보다 더 밀접하게 연결된 하드웨어 이벤트를 도출할 필요가 있다. 따라서, PRIME+ABORT를 또 다른 탐지 타겟으로 고려하여 현재 캐시 이벤트 기반 탐지 방법의 한계를 극복하고 캐시 부채널 공격의 탐지 범위를 확장하고자 한다.

[0027] B. 캐시 부채널 공격 분석(Analysis of Cache Side-Channel Attacks)

[0028] 대부분의 기존 캐시 부채널 공격이 어느 정도 규칙적인 캐시 활동을 수반하기 때문에, 캐시 이벤트의 집합은 캐시 부채널 공격에 대한 강력한 지표로 사용될 수 있다. 따라서, 캐시 동작과 캐시 부채널 공격 간의 상관관계를 탐구하기 위해, 먼저 하드웨어 성능 카운터(HPC)를 사용하여 다양한 워크로드(workload) 조건에서 FLUSH+RELOAD, PRIME+PROBE, PRIME+ABORT 공격 하의 글로벌 시스템 이벤트를 수집한다. 구체적으로, 다음의 여섯 가지 다른 조건에서 GnuPG 1.4.13의 RSA 복호화 및 Openssl의 T-table AES 암호화 동안 발생하는 하드웨어 이벤트와 그 행동 패턴을 조사한다:

[0029] 조건 1: RSA 복호화를 수행하면서 FLUSH+RELOAD 공격 실행

[0030] 조건 2: RSA 복호화만 수행

[0031] 조건 3: Openssl 1.01g의 AES 암호화를 수행하면서 PRIME+PROBE 공격 실행

[0032] 조건 4: Openssl 1.01g의 AES 암호화만 수행

[0033] 조건 5: Openssl 1.01e의 AES 암호화를 수행하면서 PRIME+ABORT 공격 실행

[0034] 조건 6: Openssl 1.01e의 AES 암호화만 수행

[0035] 수집한 전체 데이터 셋 중에서, 공격과 관련된 하드웨어 이벤트를 다음 두 가지 기준에 따라 찾아낸다: (1) 카운트 빈도가 독특한 분포(distinctive distributions)를 보여야 하고, (2) 정상(또는 선량한) 프로그램 실행(benign program executions)과 구별될 수 있는 시간적 행동(temporal behavior)을 가져야 한다.

[0036] **실험 환경.** 16GB RAM과 인텔 i7-6700 스카이레이크(Skylake) 프로세서가 장착된 시스템을 사용하였다. 이 프로세서는 하이퍼스레딩(hyperthreading)이 있는 3.40GHz의 쿼드 코어(quad cores)와 8MB의 LLC를 가지고 있다. CPU는 세 개의 고정 기능 카운터 그리고 논리 코어당 PMU에서 최대 48 비트의 이벤트를 카운트할 수 있는 세 개의 프로그래머블 카운터를 지원한다. 2019년 11월 12일에 발표된 Intel PCM을 기반으로 데이터 수집기를 구현하였으며, 이는 Ubuntu 16.04 LTS에서 PMU의 하드웨어 이벤트 값을 수집하는 데 사용된다.

[0037] **하드웨어 이벤트의 분포.** 프로그래머블 카운터는 인텔 스카이레이크 아키텍처에서 캐시와 관련된 40가지 이상의 상이한 유형의 하드웨어 이벤트를 지원한다. 실제 환경에서 타겟 공격이 캐시 이벤트에 미치는 실제 영향을 조사하기 위해, 6가지 조건(조건 1 내지 6) 하에서 캐시 미스와 관련된 9개의 후보 하드웨어 이벤트 유형을 모니터링하였다. 조사를 바탕으로, 이들 중 세 가지가 가장 구별되는 분포를 보인다는 것을 확인하였다. 도 1은 85,000개의 캐시 관련 하드웨어 이벤트 샘플을 보여주며, 여기에는 MEM_LOAD_ UOPS_RETIRED.L1_MISS (L1 miss), MEM_LOAD_UOPS_RETIRED.L2_MISS (L2 miss), MEM_LOAD_UOPS_RETIRED.L3_MISS (L3 miss)가 포함된다. 모든 6가지 다른 조건에서 각각의 이벤트에 대한 전역 카운트를 매 10ms마다 측정함으로서 샘플을 수집하였다. 도 1(a), (c), 및 (e)는 조건 1과 2 (RSA 복호화 중 FLUSH+RELOAD 공격과 단독 RSA 복호화) 하의 결과를 나타내고, 도 1(b), (d), 및 (f)는 조건 3과 4 (AES 암호화 중 PRIME+PROBE 공격과 단독 AES 암호화) 하의 하드웨어 이벤트 분포를 제시하며, 그리고 도 2(a), (c), 및 (e)는 조건 5와 6 (AES 암호화 중 PRIME+ABORT 공격과 단독 AES 암호화) 하의 결과를 보여준다. 도 1과 도 2의 가로축과 세로축은 각각 발생한 하드웨어 이벤트의

카운트와 그 빈도(frequency)를 나타낸다. 조건 1과 2에서, FLUSH+RELOAD 공격이 모든 캐시 레벨에서, 단독 RSA 복호화에 비해, 평균적으로 25% 이상의 캐시 미스를 유발하는 것을 관찰하였다. 또한, 캐시 레벨이 증가함에 따라 분포가 눈에 띄게 분리되는 것을 확인하였다. 따라서, 캐시 미스의 현저한 증가는 FLUSH+RELOAD 공격의 주요 지표가 될 수 있다. 조건 3과 4에서, 모든 캐시 레벨에서 캐시 미스의 분포가 중복(overlap) 없이 완전히 분리되며, 이는 이러한 구분되는 분포를 활용하여 PRIME+PROBE 공격을 AES 암호화와 구별하기가 간단함을 보여준다. 그러나, 조건 5와 6에서, 캐시 이벤트의 분포가 대부분 서로 겹치며, 이는 PRIME+ABORT 공격이, 정상적인 AES 암호화와 비교할 때, 캐시 동작에 크게 영향을 미치지 않음을 암시한다. 그러므로, FLUSH+RELOAD나 PRIME+PROBE와는 달리, 캐시 미스 이벤트만을 관찰하여 PRIME+ABORT 공격을 탐지하는 것은 매우 어렵다.

[0038] 따라서, 효과적인 PRIME+ABORT 탐지에 적합한 하드웨어 이벤트를 찾기 위해, PRIME+ABORT가 주로 활용하는 것으로 알려진 인텔 TSX 하드웨어에서 발생할 수 있는 여러 인텔 TSX 하드웨어 이벤트를 잠재적인 후보로 조사하였다. PRIME+ABORT의 공격 방식이, 인텔 TSX의 사용을 제외하면, 기존의 접근 기반 공격과 유사한 절차로 구성된다는 사실을 바탕으로, FLUSH+RELOAD와 PRIME+PROBE에서의 캐시 이벤트처럼 PRIME+ABORT에 의해 반복적으로 발생하는 하드웨어 이벤트를 식별하고 그들의 시스템 동작을 이해하고자 한다. 프로그래머블 카운터는, 아키텍처에 따라 약간씩 다를 수 있지만, 약 20-30개의 상이한 인텔 TSX 하드웨어 이벤트를 지원한다. 조건 5와 6 하에서 트랜잭션 영역(transactional region)과 관련된 11개의 이벤트를 모니터링하였다. 도 2(b), (d), 및 (f)는 85,000 개 샘플의 분포를 보여주며, 각각 10ms마다 측정한 RTM_RETIRED.START, RTM_RETIRED.ABORTED, TX_MEM.ABORT_CAPACITY_WRITE이다. 도면에서 가로축과 세로축은 각각 이벤트의 카운트와 빈도(frequency)를 나타낸다. 파란색 영역은 AES 암호화를, 빨간색 영역은 PRIME+ABORT 공격을 나타낸다. 도면에서 보이듯이, PRIME+ABORT는, 상대적으로 매우 적은 수의 이벤트를 발생시키는 AES와 비교하여, 현저히 많은 수의 TSX 이벤트를 유발한다.

[0039] 실험 결과는, 세 가지 TSX 이벤트가 PRIME+ABORT를 정상적인 AES 암호화와 구별하기 위한 실현 가능한 지표로 사용될 수 있음을 보여준다. 보다 정확하게는, 정상 프로그램은, 트랜잭셔널 메모리의 이점을 최대한 활용하기 위해, TSX 이벤트를 유발하는 트랜잭셔널 중단(transactional aborts)을 거의 유발하지 않는다. 반면에, PRIME+ABORT는 공격의 성공을 위해 많은 트랜잭션 중단을 반드시 유발한다.

[0040] **하드웨어 이벤트의 시간적 행동(Temporal behavior).** 다음으로, 다양한 공격과 정상 프로그램 실행 하에서 모니터링된 하드웨어 이벤트의 시간적 행동을 분석하였다. 도 3은 160ms 동안 매 1ms 마다 측정한 각 하드웨어 이벤트의 카운트 값을 보여준다. 각 이벤트의 시간적 행동은 공격을 식별하는 데 의미 있는 정보를 제공한다. 관련 공격의 고유한 절차를 자연스럽게 반영할 수 있기 때문이다. 예를 들어, 공격 프로세스의 반복은 특정 하드웨어 이벤트에서 현저히 규칙적인 패턴이나 급격한 변화를 유발할 가능성이 있다. 따라서, 앞서 논의한 바와 같이, FLUSH+RELOAD 공격과 RSA 복호화, PRIME+ABORT 공격과 AES 암호화 하에서 각 하드웨어 이벤트의 시간적 동작을 조사하여 공격과 밀접하게 연결된 하드웨어 이벤트를 식별하고자 한다.

[0041] 구체적으로, RSA 복호화 중 FLUSH+RELOAD 공격을 수행하는 경우(조건 1)와 RSA 복호화(조건 2)를 수행하는 동안, 세 가지 캐시 이벤트, HALT 상태가 아닌 논리 코어에 의해 사용된 클록 사이클을 제공하는 CPU_CLK_UNHALTED.THREAD (unhalted cycle), 그리고 논리 코어의 완료된 명령(retired instructions) 개수를 제공하는 INST_RETIRED.ANY의 시간적 패턴을 분석하였다. unhalted cycle과 INST_RETIRED.ANY를 모니터링하기 위해, 고정 기능 카운터를 사용하였다. 또한, AES 암호화 중 PRIME+ABORT 공격을 수행하는 경우(조건 5)와 AES 암호화(조건 6)를 수행하는 동안, 세 가지 TSX 이벤트, unhalted cycles, 그리고 INST RETIRED.ANY의 시간적 패턴도 분석하였다.

[0042] FLUSH+RELOAD의 경우, 도 3(c)와 (e)에서 볼 수 있듯이, L3 miss, unhalted cycle, 그리고 INST_RETIRED.ANY의 패턴이 거의 완벽하게 동기화되어 있음을 관찰하였다. 도 3(a)에서 볼 수 있듯이, unhalted cycle이 피크에 달할 때, FLUSH+RELOAD의 경우에서, 도 3(a)에 도시된 바와 같이 RSA 복호화와 경우와는 반대로, L3 miss 또한 현저하게 증가한다. 마찬가지로, 도 3(g)의 경우와 동일하게, INST_RETIRED.ANY가 피크에 달할 때, L3 미스 또한 현저하게 증가한다. 또한, 인간이 쉽게 인식하기에는 너무 미묘한 신호일 수 있으나 공격은 반복적인 절차를 가지기 때문에, 유도된 하드웨어 이벤트는 주기적인 패턴을 가질 수 있다. 따라서, 주기성과 샘플 간의 장기적인 시간 의존성(long-range temporal dependence)을 고려하는 RNN(Recurrent Neural Networks)과 LSTM(Long Short-Term Memory)을 사용하는 것이 더 적합할 수 있다. RNN과 LSTM에 대한 더 자세한 논의는 후술한다.

[0043] 이 실험 결과는, CPU가 clflush 명령을 반복적으로 실행하도록 강제하여 엄청난 수의 주기적인 캐시 미스와 unhalted cycle 및 INST_RETIRED.ANY의 카운트 값을 증가시키는 FLUSH+RELOAD 공격의 절차를 나타낸다.

반면에, RSA는 이러한 절차가 없다. unhalted cycle의 카운트가 증가할 때 L1 미스 및 L2 미스도 때때로 함께 증가하지만, L3 미스와는 달리 완벽하게 동기화되지는 않는다. 따라서, L3 미스, unhalted cycle, 그리고 INST_RETIRED.ANY는 FLUSH+RELOAD 공격의 시간적 행동을 식별하는 데 적합한 하드웨어 이벤트이다.

[0044] PRIME+PROBE의 경우, 도 4(b)에서 볼 수 있듯이, 세 가지 캐시 이벤트가 50ms 마다 주기적으로 낮은 지점으로 감소하는 것을 관찰하였다. 도면에서는, L3 미스는 다른 캐시 이벤트에 비해 카운트 규모가 작아서, 정기적으로 감소하는 패턴을 더 인식하기 쉽게 하기 위해 실제 값의 10배로 확대하였다. 그러나, 모든 캐시 미스 이벤트에서 많은 수의 카운트를 보이는 이상치(outlier) 때문에 패턴을 식별하기 어려웠다. 반면에, AES 암호화는, 도 4(a)에서 보이듯이, 캐시 이벤트에서 뚜렷한 규칙적인 패턴을 보이지 않았다. 도 4(c)와 (d)에서, unhalted cycle과 INST_RETIRED.ANY는, PRIME+PROBE와 AES 암호화의 두 경우 모두에서, 50ms 마다 반복적으로 감소하는 것을 보여준다. 캐시 이벤트에서 관찰된 이상치처럼, unhalted cycle과 INST RETIRED.ANY의 카운트 값도 거의 같은 순간에 증가하였다. unhalted cycle과 INST_RETIRED.ANY가 PRIME+PROBE 및 AES 암호화 두 경우 모두 거의 같은 값과 주기로 규칙적인 감소 패턴을 관찰하였지만, AES 암호화 하에서는 하드웨어 이벤트의 수에서 큰 변화가 관찰되지 않았다. 따라서, 캐시 이벤트, unhalted cycle, 그리고 INST_RETIRED.ANY는 PRIME+PROBE의 시간적 행동을 구별하는 데 유용한 정보를 제공할 수 있다고 결론지었다.

[0045] AES 암호화와 PRIME+ABORT의 경우, 전반적으로 매우 주기적인 패턴을 관찰하였다. 구체적으로, unhalted cycle 카운트의 변화와는 상관없이, AES 암호화는 RTM_RETIRED.START 카운트를 25ms 마다 피크에 도달하게 하지만, RTM_RETIRED.ABORTED에는 이러한 영향을 미치지 않는다. 이는 도 3(b)와 (f)에서 확인할 수 있다. PRIME+ABORT의 경우, 세 가지 TSX 이벤트 모두 25ms 마다 피크에 도달하고 50ms 마다 낮은 지점으로 떨어진다. 그러나, RTM_RETIRED.START와 RTM_RETIRED.ABORTED의 카운트 값은 거의 동일한 반면, TX_MEM.ABORT_CAPACITY_WRITE는 현저하게 다른 카운트 값을 보인다. 이는 도 3(d)에서 확인할 수 있다. PRIME+ABORT의 unhalted cycle 또한 이전의 세 가지 TSX 이벤트와 동일한 패턴을 보이지만, 다른 변화량을 보여준다. 이는 도 3(f)에서 확인할 수 있다. PRIME+ABORT의 이러한 동작(behaviors) 및 카운트 패턴의 주요 원인은, 트랜잭셔널 영역을 설계하기 위해 인텔 TSX 하드웨어의 인터페이스인 제한된 트랜잭셔널 메모리(RTM, Restricted Transactional Memory)를 사용하기 때문이다. 구체적으로, RTM_RETIRED.START와 RTM_RETIRED.ABORTED는 각각 RTM 실행과 종료 횟수를 카운트한다. PRIME+ABORT의 공격 프로세스는 트랜잭션을 시작하고 중단 신호(abort signals)를 수신하기 위해 CPU를 지속적으로 사용해야 하므로, RTM_RETIRED.START와 RTM_RETIRED.ABORTED의 카운트는 unhalted cycle의 변화와 밀접하게 관련될 수밖에 없다. 따라서, RTM_RETIRED.START, RTM_RETIRED.ABORTED, 그리고 unhalted cycle의 시간적 변화를 모니터링하면 PRIME+ABORT 공격을 탐지하는 데 유용한 정보를 제공할 수 있다.

[0046] C. FRIME 탐지기 구축

[0047] 실험 결과, L3 캐시 미스, unhalted cycle, INST_RETIRED.ANY, 그리고 두 가지 Intel TSX 이벤트(RTM_RETIRED.START 및 RTM_RETIRED.ABORTED)가 FLUSH+RELOAD, PRIME+PROBE, 그리고 PRIME+ABORT를 정상 프로그램과 구별하는 데 가장 효과적인 측정 기준임을 발견하였다. 따라서, FRIME 탐지기는 FLUSH+RELOAD, PRIME+PROBE, 및 PRIME+ABORT 공격을 탐지하기 위해 다섯 가지 하드웨어 이벤트를 기반으로 설계될 수 있다.

[0048] 그러나, 실제 시스템에서는 다양한 정상 프로그램의 동적 워크로드가, 고정된 또는 정적인 측정 기준(예: 각 하드웨어 이벤트에 대한 사전 정의된 임계값)에 기반하여 공격을 탐지한다면 공격 탐지 방법의 정확성을 심각하게 저하시킬 수 있는 복잡한 하드웨어 이벤트를 유발할 수 있다. 이러한 실질적인 문제를 해결하기 위해, FRIME이라 명명되는 새로운 캐시 부채널 탐지 도구를 디자인하기 위해 딥러닝 기술을 도입한다. 구체적으로, 선택된 하드웨어 이벤트를 수집하고 수집된 데이터를 학습 세트로 사용하여 딥러닝 모델을 훈련시켜 공격의 존재를 추론한다.

[0049] 도 5에 제시된 바와 같이, FRIME는 다음 세 가지 과정으로 구성된다: (1) 하드웨어 이벤트의 카운트를 수집되고 처리되는 데이터 수집, (2) 수집된 카운트를 학습 세트로 사용하여 딥러닝 모델을 학습시키는 학습, 그리고 (3) 캐시 부채널 공격을 탐지하기 위해 학습된 모델이 사용되는 실시간 탐지. 자세한 공격 탐지 과정은 다음과 같다.

[0050] 1) 탐지는 하드웨어 이벤트를 카운트하기 위해 PMU를 프로그래밍하는 것으로 시작된다. 구체적으로, 고정 기능 카운터는 제어 레지스터(control registers)의 'enable field' 비트를 설정하여 활성화될 수 있고, 프로그래머블 카운터는 관련된 이벤트 번호와 umask를 사용하여 모니터할 타겟 이벤트를 지정할 수 있다. 그런 다음, 프로그래밍된 PMU는 특정 시간 구간 동안 발생한 하드웨어 이벤트의 수를 카운트한다. 카운트된 값은 하드웨어 이벤트 모니터링을 위해 PMU를 설정하기 위한 다양한 옵션을 제공하는 Intel PCM을 사용하여 구현한 PCM 컴포넌트에

의해 수집된다. 이는 아키텍처마다 다르기 때문에, 타겟 아키텍처에 적절한 구성이 필요하다.

[0051] 2) 다음으로, 데이터 처리 컴포넌트는 PMU의 카운트 값을 형식화된 데이터의 세트로 정리한다. 도 5(a)에서 설명한 것처럼, 이벤트의 카운트를 시간 순서(chronological order)대로 기록하기 위해 CSV 형식(즉, CSV 파일)이 사용될 수 있다. 그런 다음, 공격과 정상 프로그램을 구별하기 위해 딥러닝 모델에 입력되어 학습에 사용된다.

[0052] 3) 마지막으로, 학습이 완료된 후 FRIME는 처리된 CSV 파일을 사용하여 학습된 모델을 통해 공격을 탐지할 수 있다(도 5(b) 참조). 모델이 학습되고 사용을 위해 배포되면, FRIME의 탐지 절차는 실행 중 더 이상 학습을 위한 시스템 자원을 필요로 하지 않기 때문에 탐지 효율성이 매우 높다.

[0053] **딥러닝 모델.** 정적인 임계값을 활용하여 모니터링된 하드웨어 이벤트의 분포에서 뚜렷한 지점을 이용해 캐시 부채널 공격을 탐지하는 것은 실행 가능한 접근법일 수 있다. 그러나, 이러한 접근법은 주로 정적인 임계값 때문에 실용적인 한계가 있다. 예를 들어, 특정 시스템 환경에서 임계값의 효과가 입증되었더라도, 심지어 동일한 시스템의 다른 환경에서 그 임계값이 효과적일 것이라는 보장은 없다. 실용적인 시스템은 일반적으로 하드웨어 이벤트의 패턴을 크게 달라지게 하는 동적으로 변화하는 프로그램을 실행할 수 있기 때문에, 실용적인 환경에서 공격 패턴을 적응적으로 결정하기 위해 FRIME에 딥러닝 모델을 채택하여 이 한계를 극복하려고 한다.

[0054] 분석의 결과를 최대한 활용하기 위해, FRIME 탐지기를 디자인하기 위해 세 가지 DNN(Deep Neural Network) 아키텍처를 선택하였다. 구체적으로, FLUSH+RELOAD, PRIME+PROBE, 및 PRIME+ABORT 공격을 탐지하기 위해, 다층 퍼셉트론(Multi-Layer Perceptrons, MLP), 순환 신경망(Recurrent Neural Networks, RNN), 및 장단기 메모리(Long Short-Term Memory, LSTM) 모델을 채택하였다. 이 세 가지 모델은 모두 다양한 시계열 데이터를 분석하기 위해 연구되어 왔다. 비록 이 세 가지 DNN 아키텍처가 시계열 데이터를 효과적으로 학습하지만, 각 모델은 시간 정보를 포착하는 능력에 차이가 있다. 예를 들어, MLP는 샘플 간의 시간 종속성(temporal dependency)을 포착하지 않는다. MLP는 단순하게 각 샘플을 독립적으로 처리한다. RNN은, 이전 샘플로부터의 정보를 히든 상태(hidden states)를 통해 전달함으로써, 샘플 간의 시간 종속성을 고려한다. 하지만, 이는 단기적인 시간 종속성에 제한된다. 이 한계를 극복하기 위해, LSTM이 제안되었다. LSTM 유닛은 셀(cell), 입력 게이트(input gate), 출력 게이트(ourput gate), 및 망각 게이트(forget gate)로 구성된다. 이 게이트들은 임의의 시간 간격 동안 히든 상태를 유지하고 더 유연하게 히든 상태를 업데이트할 수 있게 해준다. 따라서, 탐지 결과는 사용하는 메모리의 길이에 따라 다를 수 있다. 최적의 메모리 길이는 시스템 환경에 따라 다를 수 있기 때문에, 세 가지 DNN 아키텍처를 모두 사용하여 FRIME을 구현하고 각 구현의 성능을 비교 평가한다.

[0055] 각 모델에 대해, 두 가지 활성화 함수, 즉 sigmoid와 ReLU를 사용한다. sigmoid는 최종 레이어에서 출력 점수(output scores)를 생성하는 데 사용되며, ReLU는 비선형 변환을 위해 나머지 모든 레이어에 적용된다. LSTM과 RNN의 경우, 정규화를 위해 드랍아웃(dropout)을 사용한다. 모델을 학습하기 위해, 크로스 엔트로피 손실 함수와 Adam 옵티마이저를 사용한다.

[0056] 모니터링된 하드웨어 카운트는 0에서 2.0e+08 이상의 큰 값을 가지므로, 데이터의 수치적 안정성과 성능 향상을 위해 강건한 스케일링(robust scaling)을 사용하여 전처리할 수 있다. 원본 특징(original feature) x가 주어졌을 때, 전처리된 값 x_{pre}는 다음 식을 통해 얻어진다.

[0057] [수학식 1]

[0058]
$$x_{pre} = \frac{x - Q_2}{Q_3 - Q_1}$$

[0059] 여기서, Q_1, Q_2, Q_3는 특징 값의 첫 번째, 두 번째, 세 번째 사분위수(quartiles)이다. 사분위수는 소수의 이상치에 대해 강건하기 때문에, 수학식 1의 전처리는 최대값과 최소값에 의존하는 min-max 정규화, 즉 $x_{pre} = (x - x_{min})/(x_{max} - x_{min})$보다 더 강건한 것으로 알려져 있다.

[0060] 이러한 점을 고려하여 본 발명에서는 256 개의 뉴런, 256 개의 은닉상태, 그리고 1024 개의 은닉상태를 각각 MLP, RNN, 그리고 LSTM 모델에 적용하여 각 모델이 서로 다른 정도의 시간적 특성을 학습하도록 구현하였다. RNN구조에서는 과적합 문제를 방지하기 위해 256 개의 뉴런을 이용한 완전연결층을 은닉상태 이후에 적용하였다. 모든 모델의 출력층으로는 256개의 뉴런을 이용한 완전연결층을 활용해 네 가지 경우를 분류하는 네 개의 값을 출력하도록 하였다. 네 가지의 경우는 각각 (1) 일반 프로그램, (2) FLUSH+RELOAD 공격, (3) PRIME+PROBE 공격, 그리고 (4) PRIME+ABORT 공격이다.

[0061] **평가**

[0062] 이하에서는, FRIME의 성능을 평가한다. 실제 환경에서 다양한 워크로드 조건에서 FRIME의 효능을 입증하기 위해, Intel i7-6700 프로세서에서 CPU 사용량을 세 가지 다른 수준으로 변경하여 성능을 평가하였다: 적음 (CPU 사용량 35% 미만), 중간 (35% 이상 75% 이하), 많음 (75% 초과). Ubuntu 16.04 LTS 운영 체제에서 실행되는 CPU에서 강렬한 캐시 및 TSX 활동을 생성하기 위해, GnuPG의 RSA 복호화, Openssl의 T-테이블 AES 암호화, Sysbench 벤치마크, STAMP 벤치마크를 정상 프로그램으로 사용하였다. 딥러닝 모델을 학습하고 테스트하기 위해, 정상 프로그램과 타겟 공격에 대한 총 2,520,000 개의 하드웨어 이벤트 샘플을 사용하였다. 이 중 37%를 학습 세트로, 63%를 테스트 세트로 사용하였다.

[0063] A. 정상 프로그램의 워크로드 생성

[0064] 다양한 응용 프로그램의 다양한 워크로드를 신뢰할 수 있는 방식으로 모방하기 위해, 이전 연구 문헌에 사용된 벤치마크 프레임워크를 채택하였다. 이러한 목적으로, 벤치마크 내 여러 응용 프로그램을 동시에 실행하여 다양한 워크로드에서 복잡한 정상 하드웨어 동작을 생성하였다. Sysbench는 TSX 활동이 포함되지 않은 복잡한 CPU 사용량을 위한 워크로드를 생성하는 데 사용되었다. CPU와 Intel TSX 하드웨어 모두에 대해 무거운 워크로드를 실험하기 위해, STAMP를 활용하였다. STAMP는 트랜잭셔널 메모리를 활용하는 광범위한 도메인을 커버하는 여러 벤치마크 응용 프로그램을 제공한다. 그 중, 'intruder'와 'vacation'이라는 응용 프로그램을 선택하여 사용하였다. 'intruder'는 네트워크 패킷을 병렬로 처리하는 네트워크 침입 탐지 시스템을 모방한다. 예를 들어, FRIME이 네트워크 서버에 배포되면, 다른 보안 메커니즘과 함께 서버에서 실행될 수 있다. 이러한 조건은 매우 신중하게 다루어야 하는데, 이는 시스템 전체의 보안이 이러한 공존하는 다양한 보안 메커니즘의 충돌로 인해 영향을 받을 수 있기 때문이다(예: 한 프로그램의 정상 동작이 다른 프로그램에 의해 악의적인 동작으로 탐지될 수 있음). 실험에서, intruder의 하드웨어 행동은 이러한 충돌을 고려하여 정상 워크로드에 포함된다. 정상 응용 프로그램의 일반적인 워크로드를 시뮬레이션하기 위해, vacation 응용 프로그램을 선택하였다. 이 응용 프로그램은 회사 내 여행 항목(travel items)에 대한 고객과 그들의 예약을 추적하는 시스템을 모방한다. 실행 중, 여러 클라이언트 스레드(client threads)가 여행 추적 시스템(travel tracking system)의 데이터베이스에 접근한다. 실행 흐름은 네트워크 서버에서 실행되는 실제 응용 프로그램과 유사한 하드웨어 이벤트 동작을 생성할 수 있다.

[0065] 현실 세계의 시스템은 일반적으로 서로 다른 CPU 사용량을 가진 여러 프로세스를 동시에 실행하며, 이는 동적으로 변화하는 하드웨어 이벤트를 유발할 수 있다. 상이한 수준의 워크로드를 갖는 다양한 하드웨어 이벤트 패턴을 보이는 이러한 실질적인 환경을 반영하기 위해, 상이한 수준의 워크로드 생성을 위한 여러 애플리케이션의 조합을 사용하였다. 구체적으로, Sysbench, STAMP의 intruder와 vacation, RSA 복호화, 그리고 AES 암호화의 가능한 조합을 나열하고, 각 조합을 3분 동안 실행하면서 CPU 사용량을 측정하였다. 측정 결과를 바탕으로, 워크로드 조합을 세 가지 수준으로 분류하였다: 적음(35% 미만의 CPU 사용량), 중간(35% 이상 75% 이하의 CPU 사용량), 많음(75% 초과의 CPU 사용량). 표 1은 세 가지 워크로드 그룹에서 실행될 수 있는 정상 프로그램의 상세한 조합을 요약한다.

[0066] [표 1]

Low (Less than 35% of CPU usage)

•Sysbench	•Sysbench + RSA decryption	•Sysbench + AES encryption
•Intruder A†	•Indruder A + AES encryption	•Vacation
•Intruder B‡	•Indruder B + AES encryption	•Vacation + AES encryption

Moderate (Between 35% and 75% of CPU usage)

•Two of Sysbench	•Three of Sysbench
•Two of Sysbench + RSA decryption	•Three of Sysbench + RSA encryption
•Two of Sysbench + AES decryption	•Three of Sysbench + AES encryption
•Intruder A + Intruder B	•Intruder A + Intruder B + AES encryption

High (More than 75% of CPU usage)

• Four of Sysbench + RSA decryption	•Intruder A + Intruder B + Three of Sysbench
• Four of Sysbench + AES decryption	•Intruder A + Intruder B + Four of Sysbench
• Four of Sysbench	• Intruder A + Intruder B + Three of vacation + AES encryption
	• Intruder A + Intruder B + Four of vacation + AES encryption

† ‡ Intruder A and B are configured with different parameter settings. Type A induces around 30% more number of TSX events and a similar number of cache events compared to type B.

[0067]

[0068] B. 다중 공격 탐지 성능

[0069] 도 6 (a), (b), 및 (c)는 높은 워크로드 조건에서 1ms 간격으로 85,000회 측정된 L3 미스, RTM_RETIRED.START, 및 RTM_RETIRED.ABORTED 이벤트 카운트의 분포를 각각 보여준다. 도면에서 가로축과 세로축은 각각 카운트된 하드웨어 이벤트와 카운트 값의 빈도(frequency)를 나타낸다. 많은 프로그램이 높은 워크로드 조건에서 실행되므로, 프로그램 실행 흐름의 복잡성과 그에 따른 하드웨어 이벤트는 자연스럽게 증가한다. 워크로드가 적음에서 많음으로 증가함에 따라 캐시 이벤트의 두 분포(하나는 각 공격 실행 도중 획득된 것이고 다른 하나는 정상 프로그램 실행 도중 획득된 것)가 더 유사해지고 겹치는 부분도 증가함을 관찰하였고, 이는 공격 탐지를 상당히 복잡하게 만든다. 특히, 이러한 겹치는 분포는, 하드웨어 카운트 값만을 기반으로 공격의 존재 여부를 나타내는 경계 값을 결정하는 것을 어렵게 만든다. 따라서, 고정 임계값 기반의 탐지는, 워크로드가 매우 낮은 단순한 환경에서는 효과적일 수 있지만, 복잡한 워크로드 조건에서는 정상 프로그램과 악의적인 공격을 구분하는 데 더 이상 효과적이지 않을 수 있다. 제안된 방식은 심층 신경망 모델을 이용하여 이러한 문제를 해결한다. 표 2는 서로 다른 심층 신경망 모델을 사용하여 400,000회 공격 탐지를 수행하여 세 가지 워크로드 조건에서 FRIME의 성능을 요약한다.

[0070] [표 2]

Attack	Workload	Accuracy (%)	Precision (%)	Recall (%)	False Positive (%)	False Negative (%)
MLP						
FLUSH + RELOAD	Low	79.62	99.98	75.00	0	24.99
	Moderate	67.41	99.98	59.62	0	40.38
	High	82.45	99.95	48.08	0	51.91
PRIME + PROBE	Low	50.15	51.30	7	0	92.99
	Moderate	75.17	52.99	9.57	0	90.42
	High	75.00	39.05	9.48	0	90.51
PRIME + ABORT	Low	86.56	88.89	73.09	0	26.91
	Moderate	80.13	80.93	73.19	0	26.80
	High	75.21	98.85	26.86	0	73.13
RNN						
FLUSH + RELOAD	Low	92.27	99.81	91.65	0.09	8.34
	Moderate	94.45	97.90	99.53	1.18	0.47
	High	97.17	98.91	98.63	0.27	1.36
PRIME + PROBE	Low	51.34	55.35	76.79	0.11	23.20
	Moderate	85.16	66.89	96.21	2.30	3.78
	High	73.27	48.15	92.29	3.40	7.70
PRIME + ABORT	Low	81.64	70.86	67.07	1.81	32.93
	Moderate	74.01	64.36	52.44	2.06	47.55
	High	78.20	64.17	40.24	1.23	59.75
LSTM						
FLUSH + RELOAD	Low	98.69	99.98	99.13	0	0.87
	Moderate	97.86	98.83	99.92	1.12	0.07
	High	99.89	99.79	99.81	0.06	0.18
PRIME + PROBE	Low	81.53	99.29	99.32	0.01	0.67
	Moderate	96.48	89.30	99.98	0.94	0.01
	High	77.99	54.18	99.77	1.08	0.22
PRIME + ABORT	Low	99.80	99.89	99.47	0.03	0.52
	Moderate	99.42	99.99	99.23	0	0.76
	High	98.86	99.99	99.63	0	2.36

[0071]

[0072] 다중 유형의 공격이 연속적으로 수행되는 환경에서 FRIME의 효과를 입증하기 위해, 다중 클래스 분류 방법을 사용한다. 구체적으로, FLUSH+RELOAD, PRIME+PROBE, PRIME+ABORT, 그리고 정상 프로그램 실행의 4가지 경우로 모니터링된 하드웨어 이벤트를 구분하기 위해, 각각 MLP, RNN, LSTM으로 구현된 세 가지 버전의 FRIME을 구현하였다. 단일 훈련된 모델로 FLUSH+RELOAD, PRIME+PROBE, 및 PRIME+ABORT 공격을 동시에 탐지하는 FRIME의 정확도(accuracy), 정밀도(precision), 및 재현율(recall)에 대한 성능을 평가한다. 정밀도와 재현율은 각각 모든 긍정 답변의 진양성율(true positive rate)과 전체 실제 긍정 사례의 진양성율을 나타냅니다. 예를 들어, 높은 정밀도와 재현율은 각각 공격의 존재를 탐지할 때 모델이 낮은 위양성율(false positive rate) 및 위음성율(false negative rate)을 보인다는 것을 의미한다. 정확도는 성공적인 공격 탐지 비율을 의미한다.

[0073] 표 2에 제시된 실험 결과에 따르면, FRIME은, 평균적으로 (1) FLUSH+RELOAD 공격을 MLP로 76.49%, RNN으로 94.63%, LSTM으로 98.81%의 정확도로, (2) PRIME+PROBE 공격을 MLP로 66.77%, RNN으로 69.92%, LSTM으로 85.33%의 정확도로, 그리고 (3) PRIME+ABORT 공격을 MLP로 80.63%, RNN으로 77.95%, LSTM으로 99.36%의 정확도로 동시에 탐지할 수 있다. 이 경우에서, LSTM 기반 방법이 최고의 성능을 보였다. 앞서 논의한 바와 같이, LSTM은 샘플 간의 단기 및 장기 의존성을 유연하게 포착한다. 따라서, 샘플을 독립적으로 처리하고 시간 정보를 무시하는 MLP에 비해 우수한 성능을 달성한다.

[0074] 다음으로, FRIME의 공격 탐지 시간을 측정하여 실용성을 평가한다. 실제로, 실시간 탐지는 실용적인 배포를 위한 또 다른 중요한 요구 사항이다. 이는 시스템이 공격받을 때 복구를 위한 피해자의 후속 조치에 영향을 미칠 수 있기 때문이다. 7,200 번의 탐지 시도에서 FRIME에 의해 소요된 평균 시간을 측정하였다. 실험에 따르면, FRIME은 탐지 결과를 보고하는 데 MLP는 평균 4ms, RNN은 4.5ms, LSTM은 4.7ms가 요구되었다. 반면 공격은 AES 암호화 및 RSA 복호화를 타겟 프로그램으로 하여 키 정보(key information)를 탈취하는 데 단일 실행을 관찰하는 데 2.67ms에서 15.51ms가 요구된다. 이는, FRIME이 RSA 복호화의 첫 번째 관찰 시도 전에 FLUSH+RELOAD를

탐지할 수 있음을 나타낸다. 단일 RSA 복호화에는 15.51ms가 필요하기 때문이다. 비록 FRIME의 다중 클래스 버전이 AES 암호화에서 PRIME+PROBE 및 PRIME+ABORT 공격자가 단일 AES 암호화를 관찰하도록 허용할 수 있지만(2.67ms 필요), 공격은 두 번째 시도 전에 탐지될 수 있다. PRIME+PROBE 및 PRIME+ABORT 공격은 보통 최소 수천 번의 암호화가 필요하다는 점을 고려할 때, 실제로 민감한 데이터가 유출되기 훨씬 전에 진행 중인 공격을 효과적으로 탐지할 수 있다.

[0075] 제안하는 방법은 MLP를 사용했을 때 평균적으로 0.00%의 위양성율과 57.56%의 위음성율을 보였고, RNN을 사용했을 때는 평균적으로 1.38%의 위양성율과 20.56%의 위음성율을 보였으며, LSTM을 사용했을 때는 평균적으로 0.36%의 위양성율과 0.63%의 위음성율을 보였다. 오탐을 줄이기 위해, 훈련된 모델의 분류 임계값을 조정하거나, 연속적인 탐지 결과 후에 탐지 결과를 보고하는 방식으로 FRIME을 변경하는 방법을 고려할 수 있다(예: 동일한 결과가 연속으로 다섯 번 탐지된 후 보고). RSA 복호화에 대한 FLUSH+RELOAD 공격은 최소한 하나의 완전한 복호화 동안 캐시 접근 시간을 반복적으로 측정해야 하므로, 연속적으로 모니터링된 하드웨어 이벤트는 연속적인 공격 동작을 포함할 가능성이 높다.

[0076] 비교를 위해, 각각의 공격을 탐지하기 위해 개별적으로 훈련된 모델을 사용하여 FRIME을 구현하였다. 제안 기법의 개별 공격 탐지 버전은 (1) FLUSH+RELOAD 공격은 MLP로 96.63%, RNN으로 96.63%, LSTM으로 99.92%의 정확도, (2) PRIME+PROBE 공격은 MLP로 85.49%, RNN으로 84.81%, LSTM으로 98.53%의 정확도, (3) PRIME+ABORT 공격은 MLP로 92.57%, RNN으로 97.50%, LSTM으로 99.71%의 정확도로 공격을 탐지할 수 있었다. 공격 탐지 시간에 관해서는, MLP는 평균 0.0923ms, RNN은 0.1306ms, LSTM은 0.1817ms가 소요되었다. 각 공격에 대하여 개별적으로 훈련된 모델을 사용하는 구현과 비교했을 때, 멀티클래스 버전의 FRIME은, 공격을 동시에 탐지하기 위해 단일 딥러닝 모델만 유지하고 관리하기 때문에, 하드웨어 오버헤드가 낮다. 그러나, 그 대가로, 멀티클래스 버전의 FRIME은 감소된 정확도를 보인다(MLP에서 평균 16.93%, RNN에서 12.15%, LSTM에서 4.89% 감소). 그럼에도 불구하고, LSTM 기반 구현은 FLUSH+RELOAD, PRIME+PROBE, PRIME+ABORT에 대해 각각 평균 98.81%, 85.33%, 99.36%의 높은 탐지 정확도를 유지하였다. 따라서, 멀티클래스 버전의 FRIME은 약간 감소한 탐지 정확도에도 불구하고 여전히 실용적으로 수용 가능한 탐지 정확도를 제공하며, 각 공격에 대한 개별 딥러닝 모델 관리 오버헤드를 줄이는 데 효율적이다.

[0077] 기존 방법들 대부분이 상이한 하드웨어 이벤트를 모니터링하고 입력으로 활용한다는 점을 고려할 때, 동일한 데이터셋으로 FRIME의 성능을 모든 기존 방법들과 공정하게 평가할 수 있는 일관된 환경을 구성하는 것은 어렵다. 그러나, 여러 후보 중 적절한 방법 선택이라는 측면에서 제안하는 방법을 다른 방법들과 비교한다. 표 3은 여러 탐지 스킴을 탐지 범위, 하드웨어 이벤트 선택의 기준, 워크로드 고려 사항, 그리고 시연된 환경설정(demonstrated configurations)의 측면에서 비교한다. 표 3에서, 'o'는 고려되었거나 자세히 명세되었음을 의미하고, 'x'는 고려되지 않았거나 자세히 명세되지 않았음을 의미한다. 대부분의 탐지 방법들이 각자가 상정한 환경과 탐지대상에 대하여 상당히 정확한 탐지 결과를 보였기 때문에, 비교를 위해 적절한 기준으로 각 탐지기가 활용하는 정보 유형과 관련한 내용을을 선택하였다. 성능 향상을 위해 인텔 TSX 하드웨어가 많이 사용되는 환경에서는, FRIME이 TSX 이벤트를 모니터링하기 위해 성능 카운터를 사용할 수 없는 다른 방법들보다 우수할 수 있다. 다른 방법들은 시연 중에 시스템 자원 사용량 및 워크로드 분포를 거의 분석하지 않았다. 이러한 세부 사항은 실제 워크로드 조건에서 탐지 방법의 실제 성능에 강한 영향을 미칠 수 있다. FRIME은 다른 방법들이 제공하지 않는 세부 정보를 제공하여 어느 수준으로 복잡한 워크로드에 대응할 수 있는지 추정할 수 있다. 또한, 제안 방법이 동작할 수 있는 가장 짧은 모니터링 간격을 평가한다. 이는 배포 환경의 보안상의 민감도를 고려하기 위한 유용한 지표가 될 수 있다. FRIME의 특징 중 하나로, 공격의 존재 여부를 탐지할 뿐만 아니라 탐지된 공격의 정확한 유형을 구별할 수 있는 기능이 있다. 이러한 기능은 여러 종류의 공격들에 대해 일반적인 대응책을 선택하여 시스템의 전체적인 성능에 광범위한 영향이 있을 수 있는 점을 감수하는 대신에, 시스템 성능에 더 적은 영향을 줄 수 있도록 각 공격에 특화된 적합한 대응책을 선택하고자 할 때 유용하게 이용될 수 있다.

[0078] [표 3]

Considerations	FRIME	Tong et al. [26]	Mushtaq et al. [31]	Chiappetta et al. [21]	Zhang et al. (CloudRader) [24]	Payer [41] (HexPADs)	Allaf et al. [29]	Kulah et al. [22] (SpyDetector)	Briongos et al. (CacheShield) [25]	Mushtaq et al. [30] (NIGHTs-WATCH)	Mushtaq et al. (WHISPER) [23]
Detection coverage											
FLUSH+RELOAD	o	o	o	o	o	o	o	o	o	o	o
PRIME+PROBE	o	o	x	x	o	o	o	o	o	o	o
PRIME+ABORT	o	x	x	x	x	x	x	x	x	x	x
Hardware events selection											
Different distributions of induced hardware events between attack and benign programs	o	x	o	x	x	x	x	o	o	o	o
Periodical temporal patterns related to attacks behavior	o	x	x	x	x	x	x	x	x	x	x
Others	x	Random Forest feature selection	x	Threshold, signature	Threshold, signature	Threshold, signature	x	Threshold	x	x	x
Workload											
Number of considered workload condition	3	3	3	1	1	1	2	4	1	3	3
Details of workload compositions	o	o	o	o	o	o	o	o	o	o	o
Scaled intensity of each workload level (e.g., CPU usage)	o	x	x	x	x	x	x	o	x	x	x
Analyzed how workloads can affect detection result	o	x	x	x	x	x	x	o	o	o	x
Demonstration											
Real-time detection	o	o	o	o	o	o	o	o	o	o	o
Tested shortest time intervals to monitor hardware events	1ms	100ms	x	10μs	100μs	x	x	x	10μs	x	100ms
Detecting attacks with individually trained model or selected algorithm	o	o	o	o	o	o	o	o	o	o	o
Detecting presence of multiple attack types as an integrated detector	o	o	x	x	o	x	x	x	o	x	x
Distinguishing exact type of ongoing attack as an integrated detector	o	x	x	x	x	x	x	x	x	x	x
Utilized algorithms or machine learning models for detection	MLP, RNN, and LSTM	SVN	LDA, LR, SVM, QDA, Naive-Bayes, Decision Tree, KNN, and Random Forest	Anomaly detection	Anomaly detection	x	PCANN, Decision Tree C4.5, and, KNN	Anomaly detection	Cumulative Sum, Expectation Maximization, and Self Organizing Maps	MLP, RNN, and LSTM	RF, DT, SVM, and Ensemble

* o: considered or described, * x: not considered or not described

[0079]

[0080] 위 표 3에서, FRIME는 본 발명에서 제안하는 기법을 나타내고, 기존 기법을 기술한 논문은 다음과 같다.

[0081] [26] Z. Tong, Z. Zhu, Z. Wang, L. Wang, Y. Zhang, and Y. Liu, “Cache side-channel attacks detection based on machine learning,” in Proc. IEEE 19th Int. Conf. Trust, Secur. Privacy Comput. Commun. (TrustCom), Dec. 2020, pp. 919-926.

[0082] [31] M. Mushtaq et al., “Machine learning for security: The case of side-channel attack detection at run-time,” in Proc. 25th IEEE Int. Conf. Electron., Circuits Syst. (ICECS), Dec. 2018, pp. 485-488.

[0083] [21] M. Chiappetta, E. Savas, and C. Yilmaz, “Real time detection of cache-based side-channel attacks using hardware performance counters,” Appl. Soft Comput., vol. 49, pp. 1162-1174, Dec. 2016.

[0084] [24] T. Zhang, Y. Zhang, and R. B. Lee, “CloudRadar: A real-time side-channel attack detection system in clouds,” in Proc. Int. Symp. Res. Attacks, Intrusions, Defenses. Paris, France: Springer, 2016, pp. 118-140.

[0085] [41] M. Payer, “HexPADS: A platform to detect ‘stealth’ attacks,” in Proc. Int. Symp. Eng. Secure Softw. Syst. London, U.K.: Springer, 2016, pp. 138-154

[0086] [29] Z. Allaf, M. Adda, and A. Gegov, “A comparison study on flush+reload and prime+probe attacks on AES using machine learning approaches,” in Proc. UK Workshop Comput. Intell. Cardiff, U.K.: Springer, 2017, pp. 203-213.

[0087] [22] Y. Kulah, B. Dincer, C. Yilmaz, and E. Savas, “SpyDetector: An approach for detecting side-channel attacks at runtime,” Int. J. Inf. Secur., vol. 18, no. 4, pp. 393-422, Aug. 2019.

[0088] [25] S. Briongos, G. Irazoqui, P. Malagon, and T. Eisenbarth, “CacheShield: Detecting cache attacks through self-observation,” in Proc. 8th ACM Conf. Data Appl. Secur. Privacy, Mar. 2018, pp. 224-235.

[0089] [30] M. Mushtaq, A. Akram, M. K. Bhatti, M. Chaudhry, V. Lapotre, and G. Gogniat, “NIGHTs-WATCH: A cache-based side-channel intrusion detector using hardware performance counters,” in Proc. 7th Int. Workshop Hardw. Architectural Support Secur. Privacy, Jun. 2018, p. 1.

[0090] [23] M. Mushtaq et al., “WHISPER: A tool for run-time detection of side-channel attacks,” IEEE

Access, vol. 8, pp. 83871-83900, 2020.

[0091] 도 7 본 발명의 일 실시예에 따른, 다종 캐시 부채널 공격 실시간 탐지 방법을 설명하기 위한 흐름도이다.

[0092] 도 7을 참조하면, 다종 캐시 부채널 공격 실시간 탐지 모델 생성 방법 등으로도 명명될 수 있는 탐지 방법은 적어도 프로세서(Processor) 및/또는 메모리(Memory)를 포함하는 컴퓨팅 장치에 의해 수행될 수 있다. 다시 말하면, 탐지 방법을 이루는 단계들 중 적어도 일부는 컴퓨팅 장치에 포함되는 프로세서의 동작으로 이해될 수도 있다. 컴퓨팅 장치는 PC(Personal Computer), 서버(Server), 랩탑 컴퓨터, 태블릿 PC 등을 포함할 수 있다. 이하에서, 탐지 방법을 설명함에 있어, 앞선 기재와 중복되는 내용에 관하여는 그 구체적인 기술을 생략하기로 한다.

[0093] 우선, 하드웨어 이벤트 정보가 획득된다(S110). 수집되는 하드웨어 이벤트 정보는 미리 정해진 시간 구간 동안 미리 정해진 간격으로 측정된 하드웨어 이벤트에 대한 카운트 정보일 수 있다. 실시예에 따르면, 수집되는 하드웨어 이벤트 정보는 L3 캐시 미스, unhalted cycle, INST_RETIRED.ANY, RTM_RETIRED.START, 및 RTM_RETIRED.ABORTED) 중 적어도 하나에 대한 카운트 정보를 포함할 수 있다. 하드웨어 이벤트 정보는 하드웨어 성능 카운터를 활용하여 수집될 수 있다. 보다 구체적으로, L3 캐시 미스, unhalted cycle, 및 INST_RETIRED.ANY에 관한 이벤트 정보는 고정 기능 카운터를 통해, RTM_RETIRED.START와 RTM_RETIRED.ABORTED에 관한 이벤트 정보는 프로그래머블 카운터를 통해 수집될 수 있다. 하드웨어 이벤트는 어느 하나의 캐시 부채널 공격이 수행되는 동안, 캐시 부채널 공격이 주행되지 않는 동안(즉, 적어도 하나의 정상 프로그램만이 동작하는 동안), 어느 하나의 캐시 부채널 공격과 정상 프로그램이 동시에 수행되는 동안 중 적어도 하나의 환경 하에서 수집될 수 있다. 따라서, 하드웨어 이벤트 정보는 레이블 정보를 포함할 수 있으며, 레이블은 해당 하드웨어 이벤트 정보에 대응하는 캐시 부채널 공격(즉, FLUSH+RELOAD, PRIME+PROBE, 및 PRIME+ABORT 중 어느 하나) 및/또는 정상 프로그램 실행일 수 있다. 즉, 하드웨어 이벤트 정보는 캐시 부채널 공격의 유무 및/또는 구체적인 캐시 부채널 공격에 대한 정보를 포함할 수 있다. 또한, 실시예에 따라, 하드웨어 이벤트 정보를 획득하는 단계는 하드웨어 이벤트 정보의 획득을 위한 프로그래머블 카운터의 설정 동작 등을 포함할 수 있다.

[0094] 물론, 하드웨어 이벤트 정보를 생성하는 컴퓨팅 장치와 탐지 방법을 수행하는 컴퓨팅 장치는 동일하거나 상이할 수 있다. 두 컴퓨팅 장치가 상이한 경우, 탐지 방법을 수행하는 컴퓨팅 장치는 하드웨어 이벤트 정보를 생성한 컴퓨팅 장치로부터, 유무선 통신망을 통하거나 USB 메모리 장치와 같은 저장 장치를 통해, 수신할 수 있다. 실시예에 따르면, 하드웨어 이벤트 정보는 CSV 파일의 형태로 저장될 수 있다.

[0095] 학습 데이터가 생성된다(S120). 하드웨어 이벤트 정보를 전처리함으로서, 학습 데이터를 생성할 수 있다. 전처리 동작은, 하드웨어 이벤트 정보의 시간 구간을 미리 정해진 시간 구간으로 분할하는 동작, 수치적 안정성 등을 위한 스케일링 동작(예컨대, 강건한 스케일링 동작) 중 적어도 하나를 포함할 수 있다.

[0096] 탐지 모델이 생성된다(S130). 탐지 모델은 학습 데이터로 소정의 인공 신경망(Artificial Neural Network, ANN) 모델을 학습시킴으로써 생성될 수 있다. 탐지 모델은 입력되는 하드웨어 이벤트 정보를 미리 정해진 분류들 중 어느 하나로 분류(또는 유추)하는 모델로써, 예시적인 인공신경망 모델은 MLP, RNN, 및 LSTM 중 어느 하나일 수 있다.

[0097] 이하에서는, 탐지 모델이 생성된 후 캐시 부채널 공격을 탐지하는 과정을 설명한다. 이하의 과정은 탐지 모델을 생성한 컴퓨팅 장치와 동일한 컴퓨팅 장치이거나 상이한 컴퓨팅 장치일 수 있다. 두 컴퓨팅 장치가 상이할 경우, 탐지 동작을 수행하는 컴퓨팅 장치에는 생성된 탐지 모델이 미리 저장되어 있을 수 있다.

[0098] 타겟 하드웨어 이벤트 정보가 획득된다(S140). 타겟 하드웨어 이벤트 정보는 S110 단계에서 수집된 하드웨어 이벤트 정보와 동일할 수 있다. 즉, 타겟 하드웨어 이벤트 정보는 미리 정해진 하드웨어 이벤트에 대한 카운트 정보로써, 미리 정해진 시간 구간 동안 미리 정해진 시간 간격으로 측정된 카운트 정보를 의미할 수 있다.

[0099] 탐지 모델의 입력 데이터가 생성된다(S150). 입력 데이터는 타겟 하드웨어 이벤트 정보를 전처리함으로써 생성될 수 있다. 입력 데이터를 생성하는 과정은 학습 데이터를 생성하는 과정과 동일할 수 있다.

[0100] 캐시 부채널 공격이 탐지된다(S160). 생성된 입력 데이터를 탐지 모델에 입력함으로써 캐시 부채널 공격이 탐지될 수 있다. 탐지 모델은 입력 데이터에 대응하는 분류 결과를 출력하고, 분류 결과는 FLUSH+RELOAD 공격, PRIME+PROBE 공격, PRIME+ABORT 공격, 및 정상 프로그램 동작 중 어느 하나일 수 있다. 즉, 탐지 모델의 출력에 따라, 캐시 부채널 공격을 탐지할 수 있다.

[0101] 이상에서 설명된 장치는 하드웨어 구성 요소, 소프트웨어 구성 요소, 및/또는 하드웨어 구성 요소 및 소프트웨

어 구성 요소의 집합으로 구현될 수 있다. 예를 들어, 실시예들에서 설명된 장치 및 구성 요소는, 예를 들어, 프로세서, 콘트롤러, ALU(Arithmetic Logic Unit), 디지털 신호 프로세서(Digital Signal Processor), 마이크로컴퓨터, FPA(Field Programmable array), PLU(Programmable Logic Unit), 마이크로프로세서, 또는 명령(instruction)을 실행하고 응답할 수 있는 다른 어떠한 장치와 같이, 하나 이상의 범용 컴퓨터 또는 특수 목적 컴퓨터를 이용하여 구현될 수 있다. 처리 장치는 운영 체제(Operating System, OS) 및 상기 운영 체제 상에서 수행되는 하나 이상의 소프트웨어 애플리케이션을 수행할 수 있다. 또한, 처리 장치는 소프트웨어의 실행에 응답하여, 데이터를 접근, 저장, 조작, 처리 및 생성할 수도 있다. 이해의 편의를 위하여, 처리 장치는 하나가 사용되는 것으로 설명된 경우도 있지만, 해당 기술 분야에서 통상의 지식을 가진 자는, 처리 장치가 복수 개의 처리 요소(Processing Element) 및/또는 복수 유형의 처리 요소를 포함할 수 있음을 알 수 있다. 예를 들어, 처리 장치는 복수 개의 프로세서 또는 하나의 프로세서 및 하나의 콘트롤러를 포함할 수 있다. 또한, 병렬 프로세서(Parallel Processor)와 같은, 다른 처리 구성(Processing Configuration)도 가능하다.

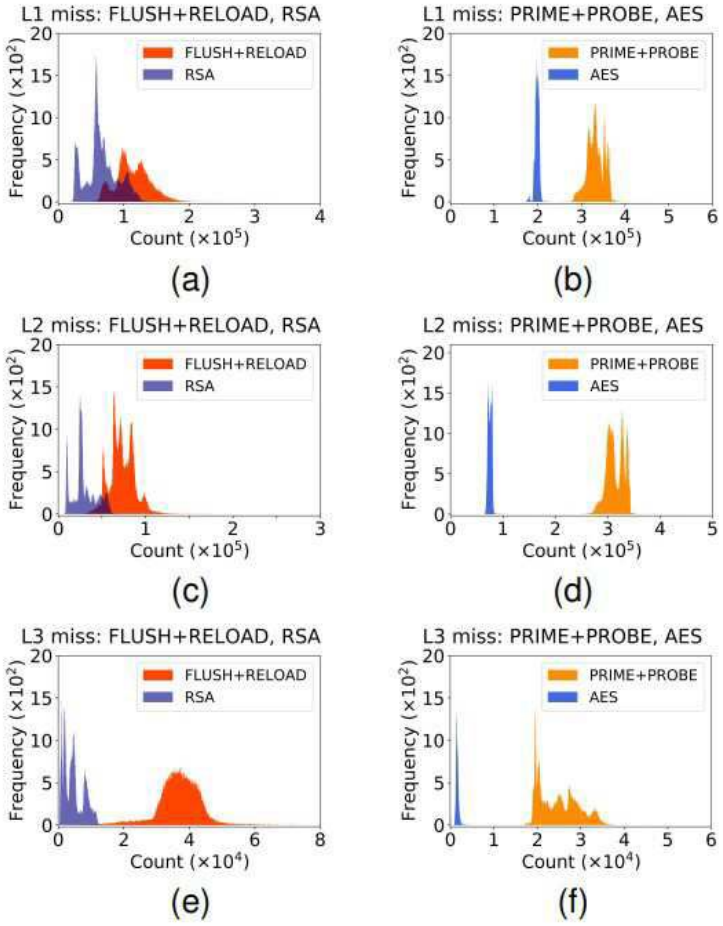
[0102] 소프트웨어는 컴퓨터 프로그램(Computer Program), 코드(Code), 명령(Instruction), 또는 이들 중 하나 이상의 조합을 포함할 수 있으며, 원하는 대로 동작하도록 처리 장치를 구성하거나 독립적으로 또는 결합적으로(Collectively) 처리 장치를 명령할 수 있다. 소프트웨어 및/또는 데이터는, 처리 장치에 의하여 해석되거나 처리 장치에 명령 또는 데이터를 제공하기 위하여, 어떤 유형의 기계, 구성 요소(Component), 물리적 장치, 가상 장치(Virtual Equipment), 컴퓨터 저장 매체 또는 장치, 또는 전송되는 신호 파(Signal Wave)에 영구적으로, 또는 일시적으로 구체화(Embody)될 수 있다. 소프트웨어는 네트워크로 연결된 컴퓨터 시스템 상에 분산되어서, 분산된 방법으로 저장되거나 실행될 수도 있다. 소프트웨어 및 데이터는 하나 이상의 컴퓨터 판독 가능 기록 매체에 저장될 수 있다.

[0103] 실시예에 따른 방법은 다양한 컴퓨터 수단을 통하여 수행될 수 있는 프로그램 명령 형태로 구현되어 컴퓨터 판독 가능 매체에 기록될 수 있다. 상기 컴퓨터 판독 가능 매체는 프로그램 명령, 데이터 파일, 데이터 구조 등을 단독으로 또는 조합하여 포함할 수 있다. 상기 매체에 기록되는 프로그램 명령은 실시예를 위하여 특별히 설계되고 구성된 것들이거나 컴퓨터 소프트웨어 당업자에게 공지되어 사용 가능한 것일 수도 있다. 컴퓨터 판독 가능 기록 매체의 예에는 하드 디스크, 플로피 디스크 및 자기 테이프와 같은 자기 매체(Magnetic Media), CD-ROM, DVD와 같은 광기록 매체(Optical Media), 플롭티컬 디스크(Floptical Disk)와 같은 자기-광 매체(Magneto-optical Media), 롬(ROM), 램(RAM), 플래시 메모리 등과 같은 프로그램 명령을 저장하고 수행하도록 특별히 구성된 하드웨어 장치가 포함된다. 프로그램 명령의 예에는 컴파일러에 의해 만들어지는 것과 같은 기계어 코드뿐만 아니라 인터프리터 등을 사용해서 컴퓨터에 의해서 실행될 수 있는 고급 언어 코드를 포함한다. 상기된 하드웨어 장치는 실시예의 동작을 수행하기 위해 하나 이상의 소프트웨어 모듈로서 작동하도록 구성될 수 있으며, 그 역도 마찬가지이다.

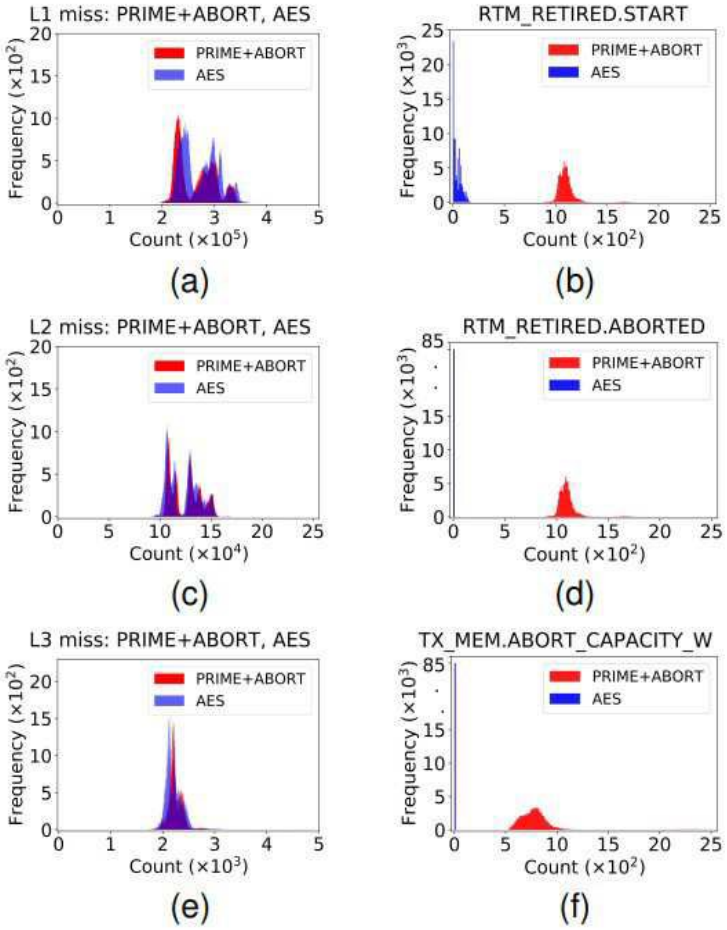
[0104] 본 발명은 도면에 도시된 실시예를 참고로 설명되었으나 이는 예시적인 것에 불과하며, 본 기술 분야의 통상의 지식을 가진 자라면 이로부터 다양한 변형 및 균등한 타 실시예가 가능하다는 점을 이해할 것이다. 예를 들어, 설명된 기술들이 설명된 방법과 다른 순서로 수행되거나, 및/또는 설명된 시스템, 구조, 장치, 회로 등의 구성 요소들이 설명된 방법과 다른 형태로 결합 또는 조합되거나, 다른 구성 요소 또는 균등물에 의하여 대치되거나 치환되더라도 적절한 결과가 달성될 수 있다. 따라서, 본 발명의 진정한 기술적 보호 범위는 첨부된 등록청구범위의 기술적 사상에 의해 정해져야 할 것이다.

도면

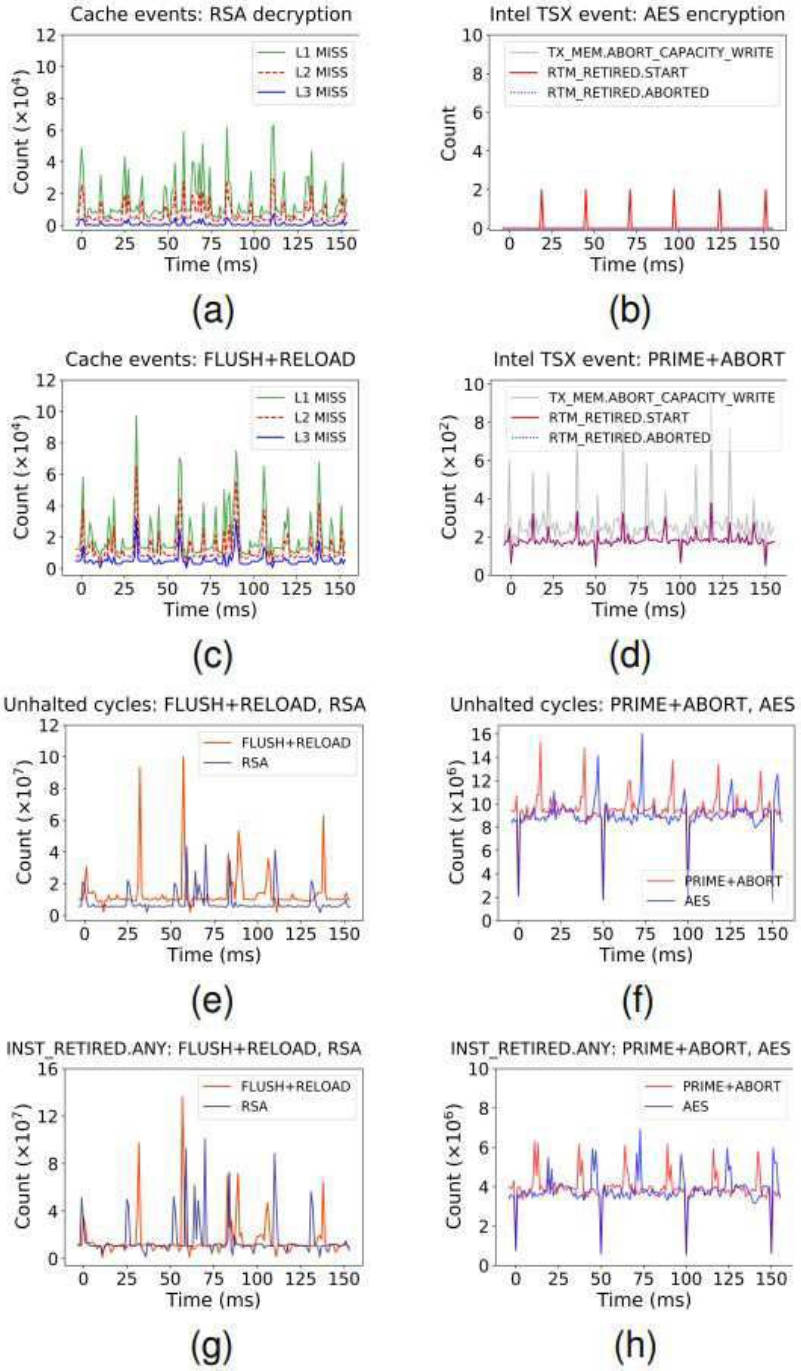
도면1


L1 miss: FLUSH+RELOAD, RSA
FLUSH+RELOAD
RSA
Frequency (×10²)
Count (×10⁵)
(a)
L1 miss: PRIME+PROBE, AES
PRIME+PROBE
AES
Frequency (×10²)
Count (×10⁵)
(b)
L2 miss: FLUSH+RELOAD, RSA
FLUSH+RELOAD
RSA
Frequency (×10²)
Count (×10⁵)
(c)
L2 miss: PRIME+PROBE, AES
PRIME+PROBE
AES
Frequency (×10²)
Count (×10⁵)
(d)
L3 miss: FLUSH+RELOAD, RSA
FLUSH+RELOAD
RSA
Frequency (×10²)
Count (×10⁴)
(e)
L3 miss: PRIME+PROBE, AES
PRIME+PROBE
AES
Frequency (×10²)
Count (×10⁴)
(f)

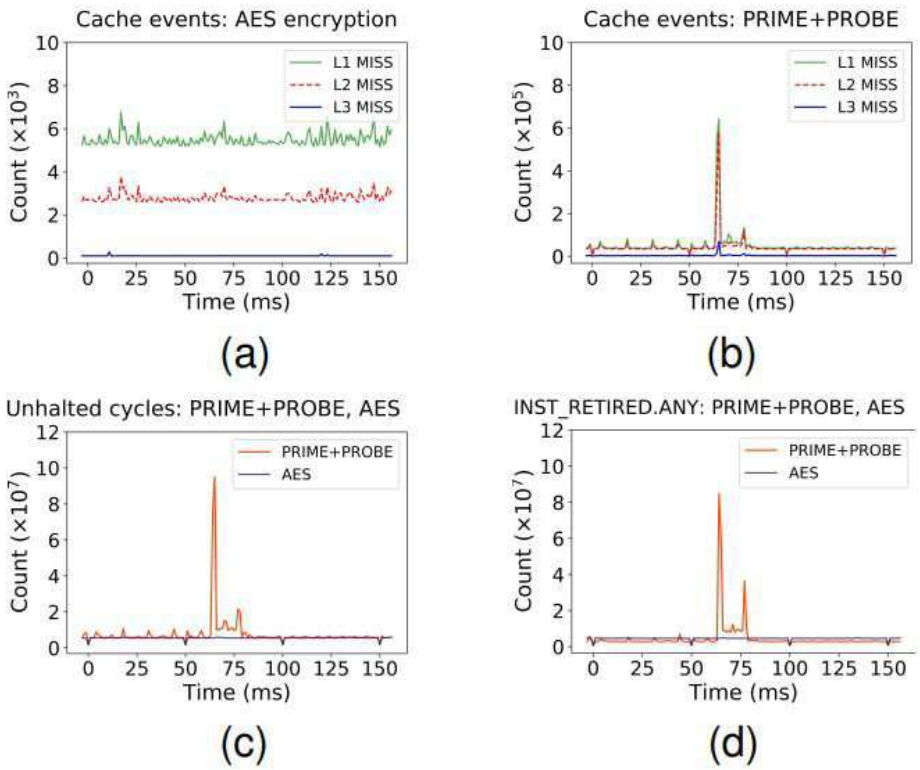
도면2


L1 miss: PRIME+ABORT, AES
PRIME+ABORT
AES
Frequency (×10²)
Count (×10⁵)
(a)
RTM_RETIRED.START
PRIME+ABORT
AES
Frequency (×10³)
Count (×10²)
(b)
L2 miss: PRIME+ABORT, AES
PRIME+ABORT
AES
Frequency (×10²)
Count (×10⁴)
(c)
RTM_RETIRED.ABORTED
PRIME+ABORT
AES
Frequency (×10³)
Count (×10²)
(d)
L3 miss: PRIME+ABORT, AES
PRIME+ABORT
AES
Frequency (×10²)
Count (×10³)
(e)
TX_MEM.ABORT_CAPACITY_W
PRIME+ABORT
AES
Frequency (×10³)
Count (×10²)
(f)

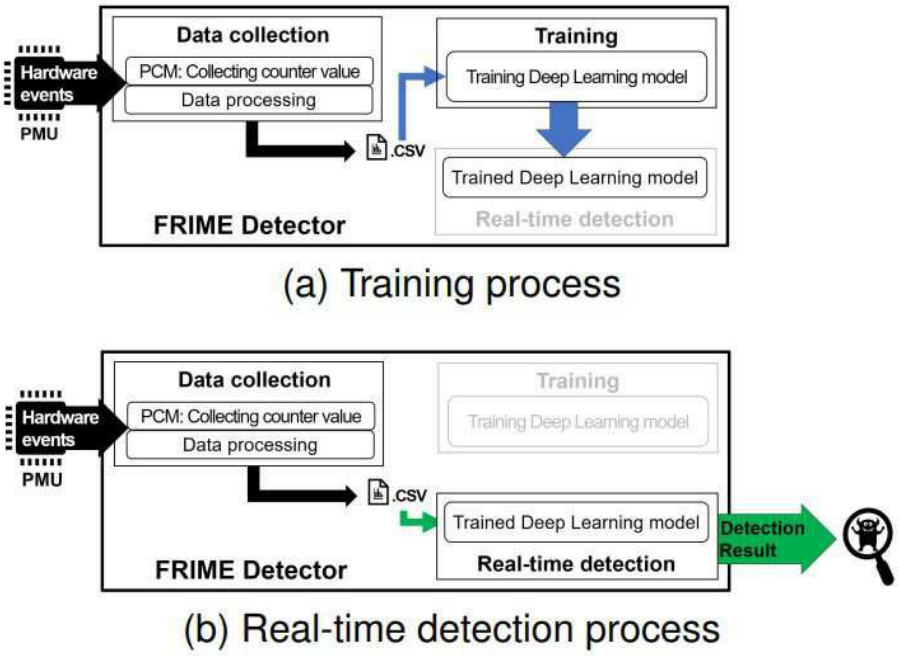
도면3


Cache events: RSA decryption
L1 MISS
L2 MISS
L3 MISS
Count (×10⁴)
Time (ms)
(a)
Intel TSX event: AES encryption
TX_MEM.ABORT_CAPACITY_WRITE
RTM_RETIRED.START
RTM_RETIRED.ABORTED
Count
Time (ms)
(b)
Cache events: FLUSH+RELOAD
L1 MISS
L2 MISS
L3 MISS
Count (×10⁴)
Time (ms)
(c)
Intel TSX event: PRIME+ABORT
TX_MEM.ABORT_CAPACITY_WRITE
RTM_RETIRED.START
RTM_RETIRED.ABORTED
Count (×10²)
Time (ms)
(d)
Unhalted cycles: FLUSH+RELOAD, RSA
FLUSH+RELOAD
RSA
Count (×10⁷)
Time (ms)
(e)
Unhalted cycles: PRIME+ABORT, AES
PRIME+ABORT
AES
Count (×10⁶)
Time (ms)
(f)
INST_RETIRED.ANY: FLUSH+RELOAD, RSA
FLUSH+RELOAD
RSA
Count (×10⁷)
Time (ms)
(g)
INST_RETIRED.ANY: PRIME+ABORT, AES
PRIME+ABORT
AES
Count (×10⁶)
Time (ms)
(h)

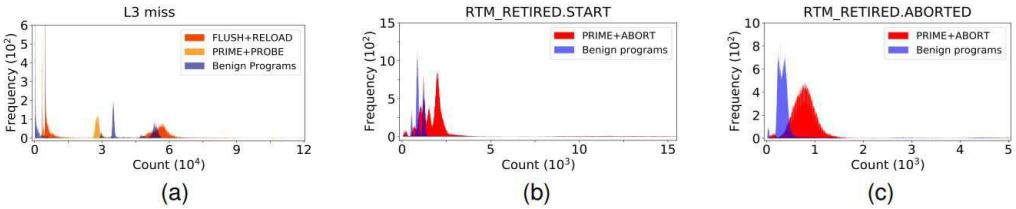
도면4


Cache events: AES encryption
Count (×10³)
L1 MISS
L2 MISS
L3 MISS
Time (ms)
(a)
Cache events: PRIME+PROBE
Count (×10⁵)
L1 MISS
L2 MISS
L3 MISS
Time (ms)
(b)
Unhalted cycles: PRIME+PROBE, AES
Count (×10⁷)
PRIME+PROBE
AES
Time (ms)
(c)
INST_RETIRED.ANY: PRIME+PROBE, AES
Count (×10⁷)
PRIME+PROBE
AES
Time (ms)
(d)

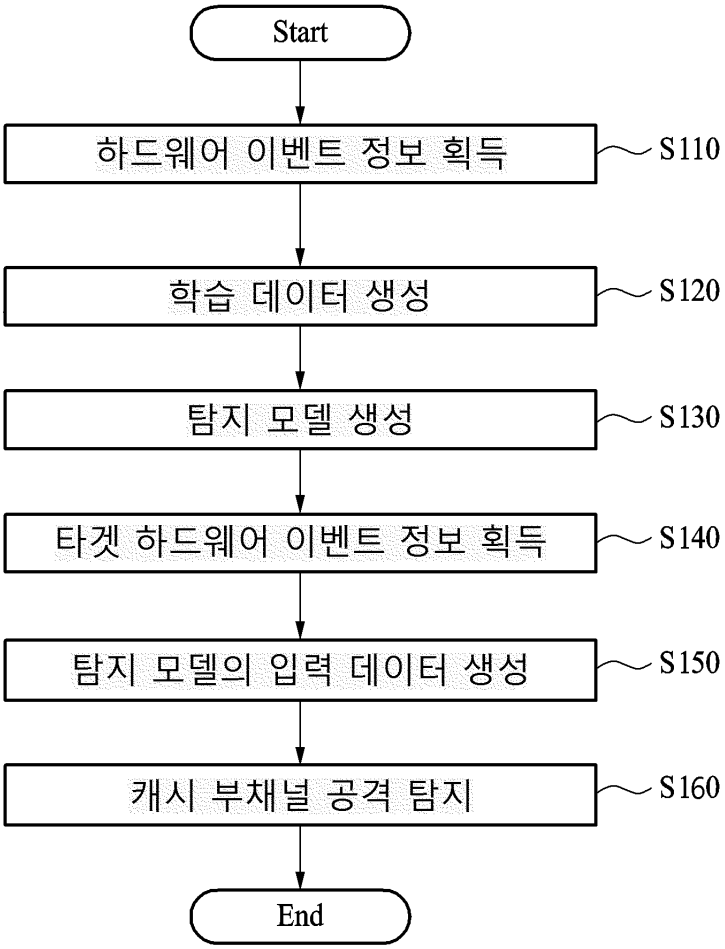
도면5


Data collection
Hardware events
PMU
PCM: Collecting counter value
Data processing
.CSV
Training
Training Deep Learning model
Trained Deep Learning model
Real-time detection
FRIME Detector
(a) Training process
Data collection
Hardware events
PMU
PCM: Collecting counter value
Data processing
.CSV
Training
Training Deep Learning model
Trained Deep Learning model
Real-time detection
Detection Result
FRIME Detector
(b) Real-time detection process

도면6


L3 miss
FLUSH+RELOAD
PRIME+PROBE
Benign Programs
Frequency (10^2)
Count (10^4)
(a)
RTM_RETIRED.START
PRIME+ABORT
Benign programs
Frequency (10^2)
Count (10^3)
(b)
RTM_RETIRED.ABORTED
PRIME+ABORT
Benign programs
Frequency (10^2)
Count (10^3)
(c)

도면7


Start
하드웨어 이벤트 정보 획득
S110
학습 데이터 생성
S120
탐지 모델 생성
S130
타겟 하드웨어 이벤트 정보 획득
S140
탐지 모델의 입력 데이터 생성
S150
캐시 부채널 공격 탐지
S160
End