Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.30, NO.6, Dec. 2020 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2020.30.6.1031

멜트다운 취약점을 이용한 인공신경망 추출 공격*

정 호 용,[1†] 류 도 현,[2] 허 준 범[3‡]
[1,2,3]고려대학교 (학생, 대학원생, 교수)

Extracting Neural Networks via Meltdown*

Hoyong Jeong,[1†] Dohyun Ryu,[2] Junbeom Hur[3‡]
[1,2,3]Korea University (Student, Graduate Student, Professor)

요 약

클라우드 컴퓨팅 환경에서 기계학습 서비스를 제공하는 Machine-Learning-as-a-Service(MLaaS) 등이 활발히 개발됨에 따라 보다 다양한 분야에서 인공지능 기술을 손쉽고 효과적인 방법으로 활용할 수 있게 되었다. 클라우드 환경에서는 가상화 기술을 통해 각 사용자에게 논리적으로 독립된 컴퓨팅 공간을 제공하는데, 최근 시스템의 취약점을 이용해 클라우드 테넌트(tenant) 사이에 다양한 부채널이 존재할 수 있다는 연구 결과가 발표되고 있다. 본 논문에서는 이러한 멀티-테넌시(multi-tenancy) 환경에서 멜트다운 취약점을 이용하여 딥러닝 모델의 내부 정보를 추출할 수 있는 현실적인 공격 시나리오를 제시한다. 이후 TensorFlow 딥러닝 서비스에 대한 실험을 통해 92.875%의 정확도와 1.325kB/s의 속도로 인공신경망의 모든 정보를 추출할 수 있음을 보인다.

ABSTRACT

Cloud computing technology plays an important role in the deep learning industry as deep learning services are deployed frequently on top of cloud infrastructures. In such cloud environment, virtualization technology provides logically independent and isolated computing space for each tenant. However, recent studies demonstrate that by leveraging vulnerabilities of virtualization techniques and shared processor architectures in the cloud system, various side-channels can be established between cloud tenants. In this paper, we propose a novel attack scenario that can steal internal information of deep learning models by exploiting the Meltdown vulnerability in a multi-tenant system environment. On the basis of our experiment, the proposed attack method could extract internal information of a TensorFlow deep-learning service with 92.875% accuracy and 1.325kB/s extraction speed.

Keywords: Meltdown, neural network stealing, cloud computing, deep learning

I. 서 론

딥러닝 기술의 발달에 따라 인공지능 기술은 실생활의 여러 분야에서 활발히 사용되고 있으며 점점 그 중요성이 증가하고 있다. 특히 기업의 경우 경제적, 서비스적 개선을 위해 앞다투어 인공지능 솔루션을 도입하는 추세이며 이 과정에서 클라우드 서비스를 이용하는 경우가 많다. 인공지능 개발 및 배포를 위해 직접 인프라를 구축하기보다, 빠르고 안정적이며 비교적 저렴하게 인공지능 솔루션을 도입할 수 있는

Received(09. 04. 2020), Modified(11. 06. 2020), Accepted(11. 09. 2020)
* 이 논문은 2019년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No.2019-0-00533, 컴퓨터 프로세서의 구조적 보안 취약점 검증 및 공격 탐지대응), (No.2018-0-00269, 개인정보를 안전하고 편리하게 빅데이터 처리할 수 있는 방법), (No.2019-0-01697, 블록체인 플랫폼 보안취약점 자동분석 기술 개발).
† 주저자, yongari38@korea.ac.kr
‡ 교신저자, jbhur@isslab.korea.ac.kr(Corresponding author)

클라우드 서비스에 대한 관심이 급증하고 있다. 이러한 서비스는 대부분 멀티-테넌시 환경에서 동작한다. 클라우드 서비스는 가상화 기술을 통해 각 사용자에게 논리적으로 독립된 컴퓨팅 공간을 마련하여 멀티-테넌트 환경을 제공한다.

클라우드 의존도의 증가에 따라 새로운 보안 위협과 도전과제들이 생겨났다. 방화벽과 경계 보호가 주를 이루는 로컬 환경과 달리, 클라우드에는 여러 사용자의 데이터가 동일 컴퓨터에 상주할 뿐만 아니라 연산까지 동일 컴퓨터에서 이루어지기도 한다. 이는 클라우드의 멀티-테넌시 구조에서 기인하고, 이러한 구조를 악용하여 다른 사용자의 데이터에 접근이 가능할 수 있다.

클라우드의 딥러닝 모델 내부 구조가 공격에 의해 유출되는 경우, 지적재산권 침해로 인한 경제적 손실뿐 아니라, 유출된 모델을 통해 적대적 공격(Adversarial Attack)[6], 학습 데이터 추출 공격(Inversion Attack)[7] 등 추가적인 피해를 야기할 수 있기 때문에 기계학습 모델을 보호하는 것은 클라우드 환경에서 매우 중요한 문제이다. 이러한 이유로 딥러닝 모델 추출 공격에 대한 연구가 활발하게 이루어지고 있다. 본 연구에서는 기존 연구의 한계점을 극복하고 현실적인 공격이 가능한 수준의 기법을 제안한다. 제안하는 기법의 중요성은 다음과 같다.

1. 멜트다운[1]을 이용하여 멀티-테넌트 환경에서 동작하는 딥러닝 서비스의 내부 정보를 추출하는 공격 시나리오를 제시한다. 기존의 딥러닝 복원 공격들이 인공신경망 구조를 유추하는데 그치는 것에 비해 제안하는 공격은 해당 딥러닝 프로세스의 메모리 영역에 접근해 인공신경망 관련 정보를 전부 추출할 수 있다.
2. 멜트다운과 멜트다운 변종 공격들은 데이터를 선별하여 추출할 수 없고 추출된 데이터를 후처리를 통해 파싱하기도 어렵다는 한계점이 존재한다[2][3]. 본 연구에서는 이를 해결하기 위해 메모리 내에서 목표 데이터가 매핑된 위치를 파악하고 이 위치에 선별적으로 추출 공격을 수행하는 기법을 제시한다.
3. 현존하는 멜트다운 변종 및 부채널 공격 연구는 공격자가 데이터의 위치를 이미 알고 있다는 가정 하에 정해진 위치에 대한 공격을 시연한다. 하지만 TensorFlow나 PyTorch 등 많은 딥러닝 라이브러리는 메모리가 런타임에 동적으로 할당되기 때문에 사전에 목표 데이터 위치를 특정하는 것이 불가능하다. 본 연구에서는 동적으로 할당되는 변수의 위치를 추적하는 방법을 통해 기존 연구에서의 실험환경 제약을 해결한다.

4장에서 공격기법을 제시하며 5장에서 개념증명을 위해 공격자가 TensorFlow 딥러닝 프로세스의 메모리 영역에 접근해 실제 인공신경망 정보를 추출하는 것이 가능함을 실험으로 보인다. 서비스 중인 MNIST[20] 딥러닝 모델에 대해 추출 공격을 선보이며 92.875%의 정확도와 1.325kB/s의 추출 속도로 인공신경망 관련 정보를 모두 추출할 수 있음을 확인했다. 또한 공격 대상 인공지능 서비스에 관련된 어떠한 정보라도 추출이 가능함을 보이기 위해 이미지 인식 딥러닝 서비스의 입력 이미지를 추출하는 실험을 제시한다. 제안하는 공격 기법은 TensorFlow뿐 아니라 PyTorch, Caffe, MxNet 등 다른 라이브러리에 대한 공격으로도 활용될 수 있다.

II. 기존 연구

2장에서는 기존의 인공신경망 추출 연구를 소개하고 이들의 한계점을 명시한다. 본 연구에서는 기존 연구를 다음과 같이 분류하였다.

1. 딥러닝 서비스에 대해 많은 수의 입출력을 발생시키고 이를 분석하는 oracle 기반 공격[10,11,12,13]
2. 딥러닝 서비스가 탑재된 디바이스를 물리적으로 확보한 뒤 전파/전력 부채널 공격을 수행하는 하드웨어 부채널 기반 공격[14,15,16]
3. 클라우드 등 원격 접근이 가능한 딥러닝 서비스에 대해 소프트웨어적으로 부채널 공격을 수행하는 소프트웨어 부채널 기반 공격[17,18,19]

oracle 기반 공격의 경우 공격 대상 딥러닝 서비스에 지속적으로 쿼리를 보낸 후 이에 대한 응답을 분석하는 과정을 거친다. 이들은 주로 원본 모델을 본떠 대체 모델(substitute network)을 새로 구축하는 방식을 사용한다[10,11,12]. 대체 모델은 원본과 비슷한 기능을 수행할 수 있지만 내부 정보를 정확히 알아내는 것은 불가능하다. 또한 추출 가능한 정보의 해상도가 낮아 딥러닝 모델이 복잡해지면 대체 모델을 구성하는 것이 무의미해지기도 한다.

하드웨어 부채널 기반 공격은 딥러닝 서비스가 탑재된 디바이스에 직접 전력 분석이나 전자기파 공격 등을 수행하는 방식이다. 따라서 이들의 공격 범위는 매우 한정적인 반면 본 연구는 클라우드 서비스에 탑재된 딥러닝 모델을 공격하는 기법을 제안한다.

본 연구를 포함한 최근의 연구들은 원격 서버에서 동작하는 딥러닝 서비스에 대한 추출 공격을 제안한다. 이는 원격으로 통제 가능한 소프트웨어 부채널을 찾고 이를 활용하여 데이터를 유출하는 방식이다. 덕분에 공격범위가 크게 확장되지만, 기존 소프트웨어 부채널 기법들은 전부 인공신경망의 대략적인 구조만을 파악하는데 집중한다. 반면, 본 연구의 기법은 원격 부채널을 통해 목표 딥러닝 프로세스 메모리에 직접 접근하여 인공신경망의 모든 데이터를 추출한다.

III. 배경지식

3.1 딥러닝

인공신경망(artificial neural network)은 퍼셉트론(perceptron), 가중치(weight), 편향(bias), 활성함수(activation function) 등으로 이루어진 알고리즘으로써 입력값과 목표 출력값을 통해 가중치가 반복적으로 조정되어 최종적으로 입력과 출력 간의 관계가 학습되는 구조이다. 딥러닝은 인공신경망의 발전된 형태로, 여러 은닉층(hidden layer)의 조합으로 구성된다.

하나의 은닉층은 여러 퍼셉트론으로 구성되어 있으며 각 퍼셉트론에서 내부연산을 거친 출력값은 다음 은닉층의 입력으로 주어진다. 단일 퍼셉트론 내부에서는 입력과 가중치의 가중합에 편향을 더한 후 활성함수를 적용하여 다음 은닉층으로 넘겨준다.

최근 딥러닝 서비스의 신속하고 편리한 운용을 위해 클라우드가 주목을 받고 있다. 인공신경망 학습을 위한 대용량 연산 환경과 서비스 배포에 적합한 유동적인 API 환경을 제공하기에 클라우드는 직접 서버를 구축하는 것보다 저렴하다. Infrastructure-as-a-Service(IaaS), Platform-as-a-Service(PaaS), Machine-Learning-as-a-Service(MLaaS), Software-as-a-Service(SaaS) 등 여러 형태의 클라우드 서비스가 존재하고 이들은 대부분 멀티-테넌시 환경에서 동작한다. 각 기업은 수요에 따라 서비스 형태를 선택하고 그 위에서 사용자에게 딥러닝 서비스를 제공한다.

3.2 멜트다운(Meltdown)

멜트다운은 프로세서의 비순차 실행(out-of-order execution)을 악용하여 접근 권한이 없는 메모리를 읽을 수 있는 하드위어 취약점이다[1]. 2018년 초 공개 당시 대부분의 Intel 프로세서와 ARM Cortex-A75 기반 시스템이 이 취약점에 영향을 받는 것으로 알려졌다. 비순차 실행은 프로세서의 파이프라인을 효과적으로 사용하기 위한 최적화 기술로, 명령어의 순서에 관계없이 실행 가능한 명령어를 우선적으로 실행하는 기술이다. 프로세서는 비순차적으로 실행된 명령어가 커밋(commit)되기 전에 권한 검사를 진행한다. 권한이 없는 메모리 영역에 접근한 것이 확인되면 rollback interrupt를 통해 해당 명령어의 실행을 정지하고, 파이프라인을 초기화한다. 따라서 일반적인 경우 사용자는 해당 명령어의 최종 수행 결과를 알 수 없다. 하지만 해당 명령어가 비순차적으로 실행되는 동안 참조한 메모리 내용이 캐시에 적재되고, 이 데이터는 일반적인 방법으로 접근이 불가능하지만 Flush+Reload[4]와 같은 캐시 부채널 공격을 수행하여 해당 데이터를 추출할 수 있다.

Fig.1은 멜트다운 공격의 핵심인 x86 어셈블리 코드이다. 공격자는 목표 메모리 주소 rcx의 값 X를 알아내려 한다. line 4 명령은 X를 rax 레지스터에 저장한다. 이는 권한이 없는 접근이기에 커밋되기 전에 rollback이 이루어지지만, rollback interrupt가 발생하기 전에 비순차 실행으로 인해 line 5~7 명령이 실행된다. 이 중 line 7 명령은 rbx로부터 X번째 페이지의 내용에 접근하여 해당 페이지를 캐시에 적재한다. 이후 해당 페이지에 다시 접근할 때는 cache hit이 발생하여 다른 페이지에 비해 소요시간이 짧게 관측될 것이다. 이러한 접근 속도 차이로 인해 공격자는 X의 값을 알아 낼 수 있다. 공격자는 자신의 프로세스의 모든 페이지에 하나하나 접근하면서 소요시간을 측정하고, 그 중 접근 속도가 빠른 페이지가 line7에서 접근했던 X번째 페이지임

```
1 ; rcx = kernel address, rbx = probe array
2 xor rax, rax
3 retry:
4 mov al, byte [rcx]
5 shl rax, 0xc
6 jz retry
7 mov rbx, qword [rbx + rax]
```

Fig. 1. core of Meltdown

을 알 수 있다.

멜트다운 이전의 캐시 부채널 공격들은 L1 캐시의 부채널을 활용했기 때문에 공격자와 공격 대상 프로세스(이하 victim)가 같은 코어 내에 할당되어야 하는 강한 조건이 필요하다. 하지만 멜트다운의 경우 Last Level Cache(LLC)의 부채널을 활용하기 때문에 동일 CPU 배치만이 전제된다. 클라우드 환경에서는 여러 사용자가 한정된 숫자의 하드웨어를 공유하기 때문에 동일 CPU에 배치되는 경우가 많다. 이러한 사실을 악용하여 공격자는 클라우드 환경에서 멜트다운 공격을 통해 다른 사용자의 데이터에 접근할 수 있다.

3.3 동적 메모리 할당

컴파일 단계에서 자료형과 크기가 정해지는 정적 할당(static allocation)과 달리 동적 할당(dynamic allocation)은 런타임에서야 그 크기가 결정되고 이에 맞춰 공간 배치가 이루어진다. 따라서 동적 할당은 필연적으로 메모리 영역을 할당하고 해당 영역에 대한 포인터를 저장하는 단계를 거친다.

본 연구는 Python의 동적 메모리 할당에 집중한다. Python은 런타임에 동적 할당된 객체들을 추적하기 위해 딕셔너리 형태의 심볼테이블(symbol table)을 사용하고, 이는 *PyDict_Type* 인스턴스로 저장된다[7].

PyDict_Type 인스턴스에는 심볼테이블에 관한 여러 정보가 저장된다. 딕셔너리의 직접적인 데이터는 외부 위치(*PyDictEntry*)에 저장되고, *PyDict_Type* 인스턴스에는 해당 *PyDictEntry*로의 포인터가 기록되어 있다. *PyDictEntry*의 각 요소는 해시(hash), 키(key) 포인터, 값(value) 포인터로 구성된다. 키와 값 정보는 직접 기록되지 않고 해당 데이터로의 포인터를 갖는다. 키 포인터와 값 포인터는 각각 변수의 이름에 해당하는 문자열 인스턴스와 해당 변수의 실제 데이터에 해당하는 인스턴스를 가리킨다. Fig.2는 *PyDictEntry*에 저장된 데이터의 모습이다.

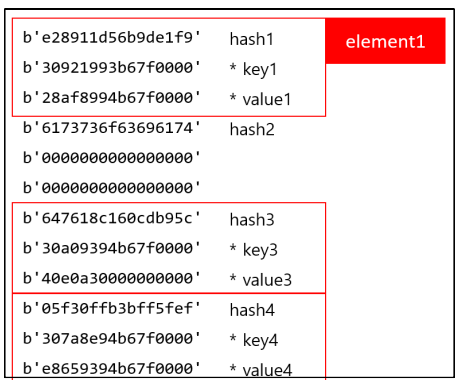

Fig. 2. *PyDictEntry* in memory

IV. 제안 공격 기법

본 장에서는 다른 유저의 딥러닝 프로세스에 접근해 인공신경망을 추출하는 기법을 설명한다. 추출 공격은 총 6단계로 구성되며 이 중 일부를 수정하여 딥러닝 서비스로의 입력 쿼리를 추출하는 공격 또한 가능함을 보인다.

1단계: 사전 작업

사전 작업에는 공격 가능 환경 여부 파악과 victim과 같은 CPU 사용 여부를 파악하기 위한 co-location test 등이 포함된다[8,9]. 공격 가능 환경 여부는 시스템 버전을 확인하는 것 만으로 수행 가능하다. co-location 여부 또한 victim 딥러닝 서비스에 입력 쿼리를 보내고 Flush+Reload 등 캐시 부채널을 이용하여 함수 사용을 모니터링하는 것으로 간단하게 파악할 수 있다.

2단계: victim의 페이지 테이블 추출

리눅스 커널은 프로세스 관리를 위해 모든 프로세스 정보를 연결리스트 형태로 저장한다. 해당 연결리스트의 헤드(head)는 *init_task* 구조체에 저장되어 있고, 이는 커널별로 고정된 위치에 존재한다. 따라서 공개된 리눅스 커널 빌드를 참조하면 프로세스 정보가 저장되어 있는 연결리스트의 헤드 위치를 파악할 수 있다[1]. 이 위치에 대해 멜트다운 공격을 진행한다. 연결리스트의 다음 노드에 대한 포인터를 참조하여 victim의 노드까지 반복적으로 새로운 노드에 대해 멜트다운 공격을 진행한다. 이때 노드 내에 저장되어 있는 프로세스 이름 등을 이용해 해당 노드가 victim의 것인지 판단할 수 있다.

각 노드에는 해당 프로세스의 PID나 이름 등의 정보 뿐만 아니라 페이지 테이블(page table)의 위치도 기록되어 있다. 이를 이용하여 victim의 페이지 테이블 위치를 특정하고 해당 위치에 대해 멜트다운 공격을 진행하여 페이지 테이블을 확보한다.

Table 1. fixed location of Data type classes

data type	address(offset)
PyObject_Type	*0x6367f0*
PyLong_Type	*0xa3c900*
PyBool_Type	*0xa3d160*
PyFloat_Type	*0xa3d300*
PyDict_Type	*0xa3ab20*
PyUnicode_Type	*0xa38440*

3단계: victim 힙 영역 추출

프로세스 내부 데이터의 위치를 식별하기 위해서는 이러한 정보가 저장되어 있는 심볼테이블을 확보해야 한다. 3.3절에서 언급했듯이 Python의 심볼테이블은 *PyDict_Type*라는 딕셔너리 인스턴스로 관리된다. 딕셔너리의 모든 데이터는 힙 영역에 할당되기 때문에 victim의 힙 영역 데이터를 먼저 확보해야 한다.

2단계에서 추출한 페이지 테이블을 이용하여 victim이 매핑된 물리주소(physical address)를 파악하고, 해당 위치에 대해 멜트다운 공격을 진행하여 victim 메모리의 힙 영역을 추출한다.

4단계: victim 인공신경망 인스턴스 추출

목표지향적인 공격을 위해 유의미한 데이터의 위치를 파악해야 한다. 따라서 인공신경망(이하 모델)을 추출하기에 앞서 프로세스 메모리의 어느 부분에 인공신경망이 매핑되어 있는지 파악하는 과정이 선행된다.

모든 Python 인스턴스는 자신의 자료형을 지칭하는 클래스에 대한 포인터를 갖는다. 예를 들어, 모든 유니코드 문자열 데이터는 *PyUnicode_Type* 클래스를 참조한다. 이러한 기본 자료형 클래스의 위치는 Python 빌드별로 고정적이기 때문에 이 위치 값을 검색하여 특정 자료형의 인스턴스를 전부 포착할 수 있다.

Table 1은 Python 3.5.2 빌드에서의 각 자료형 클래스가 존재하는 메모리 위치이다. 이를 참조하여 *PyDict_Type* 클래스는 *0xa3ab20*에 고정적으로 매핑됨을 알 수 있고, 모든 딕셔너리 인스턴스는 이 주소값을 저장하고 있음을 유추할 수 있다. 따라서 3단계에서 추출한 힙 영역 데이터에서 이 주소값을 포함하는 인스턴스를 검색하여 딕셔너리 인스턴스를 모두 포착할 수 있다.

```
{'__name__': '__main__',
…,
'__builtins__': <module 'builtins' (built-in)>,
'tensorflow': <module 'tensorflow' from
'/home/hoyong/.local/lib/python3.6/site-
packages/tensorflow/__init__.py'>,
'model': <tensorflow.python.keras.engine.
sequential.Sequential object at 0x7f4abfcee978>}
```

Fig. 3. global symbol table

Fig.3의 전역 심볼테이블은 포착된 딕셔너리 중 전역 심볼(global symbol)을 포함하고 있는 딕셔너리이다. 전역 심볼테이블만이 포함하는 요소들의 존재 여부로 여러 딕셔너리 중 전역 심볼테이블을 특정해 낼 수 있다. 해당 딕셔너리의 요소 중 모델 인스턴스의 주소에 대해 멜트다운 공격을 수행하여 목표 인스턴스를 추출한다.

5단계: 모델 구조 추출

모델의 인스턴스 안에는 해당 인스턴스 내부에서 사용하는 심볼테이블이 존재한다. 4단계에서 전역 심볼테이블을 선별한 것과 같은 방식으로 인스턴스 심볼테이블을 포착한다.

포착된 모델의 인스턴스는 Fig.4의 형식을 띈다. 이 중 *'network_nodes'*에 인공 신경망 레이어 정보가 배열 형태로 존재하는 것을 확인 할수 있다. 따라서 이 요소를 추출하는 것으로 모델의 구조를 파악할 수 있다.

```
{…,
'_name': 'sequential_6',
…,
'_layers': [<tensorflow.python.keras.engine.input_layer.InputLayer
object at 0x7f5b7e6d5160>, <tensorflow.python.keras.layers.core.Flatten
object at 0x7f5b7e6d1e10>, <tensorflow.python.keras.layers.core.Dense
object at 0x7f5b7e6cf390>,<tensorflow.python.keras.layers.core.Dense
object at 0x7f5b7e6cf400>],
…,
'_network_nodes': {'flatten_input_ib-6', 'flatten_ib-6', 'dense_ib-12',
'dense_1_ib-13'},
…,
'optimizer': <tensorflow.python.keras.optimizer_v2.adam.Adam object at
0x7f5b7e6d3240>,
…,
'loss': 'sparse_categorical_crossentropy',
…}
```

Fig. 4. symbol table of model instance

6단계: 모델 가중치 추출

이제 모델의 가중치를 추출하기 위해 은닉층 정보가 어디에 저장되어 있는지 알아낸다. 모델 인스턴스의 심볼테이블에서 해당하는 요소를 찾고, 이를 분석하여 각 은닉층의 위치를 파악한다. Fig.4의 '_layers' 요소의 값은 각 은닉층 인스턴스의 포인터로 구성된 리스트이므로 이를 활용한다.

각 은닉층 인스턴스에 접근하여 동일한 방식으로

해당 인스턴스의 심볼테이블을 포착하고, 이를 이용하여 '_trainable_weights' 요소의 값에 접근한다.

Fig.5의 *'_trainable_weights'* 요소는 두가지 *tf.Variable*의 배열로 구성되어 있음을 확인할 수 있다. 이는 각각 가중치와 편향 값에 해당하는 행렬이다. *tf.Variable*은 텐서 데이터를 위해 TensorFlow 내부적으로 사용하는 자료형이다.

*tf.Variable*의 구조는 Fig.6과 같다. 행렬을 row-major 형태의 배열로 저장하고 stride와 dimension 등 헤더 정보를 이용하여 배열 데이터를 행렬로 해석한다. Fig.6의 메모리 블록은 3x3 2차원 행렬을 나타낸 것으로 데이터 하나당 4바이트, 한 행은 12바이트임을 알 수 있다.

각 *tf.Variable*에 대한 멜트다운 공격을 통해 배열과 헤더 정보를 추출하고 이를 해석하여 원본 가중치 및 편향 행렬로 복원한다.

```
{…,
'_name': 'dense_12',
…,
'_trainable_weights':[
        <tf.Variable 'dense_12/kernel:0' shape=(784, 10) dtype=float32>,
        <tf.Variable 'dense_12/bias:0' shape=(10,) dtype=float32>
],
'_non_trainable_weights': [],
…}
```

Fig. 5. symbol table of hidden layer instances

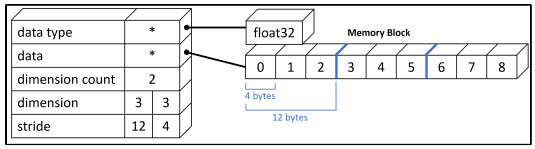

Fig. 6. *tf.Variable* structure

V. 공격 실험

5.1 실험 환경

실험은 로컬 리눅스 서버에서 진행되었으며 커널에서 기본적으로 제공하는 멀티 유저 환경을 사용했다. victim과 공격자는 각기 다른 유저로 해당 시스템에 접근하며 커널에 의해 격리된 독립적인 환경을 부여받는다. victim으로는 TensorFlow MNIST 손글씨 인식 딥러닝 서비스를 사용하였으며 구체적인 환경 설정은 다음과 같다.

- CPU: Intel(R) Core(TM) i5-7500
- kernel: Linux version 4.12.0

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_6 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_12 (Dense)             (None, 10)                7850
_________________________________________________________________
dense_13 (Dense)             (None, 10)                110
=================================================================
Total params: 7,960
Trainable params: 7,960
Non-trainable params: 0
_________________________________________________________________

>> layer2
[<tf.Variable 'dense_13/kernel:0' shape=(10, 10) dtype=float32, numpy=
array([[-1.3233812 ,  0.26736608,  1.0216433 ,  0.59734255, -1.0421642 ,
        -0.3907373 , -0.13165668,  1.0037075 , -1.406748  , -2.09728   ],
       [-0.87404656,  0.26599044,  0.17201532, -0.1804208 ,  0.43607864,
         -2.5771687 , -1.6013101 ,  0.25765058, -0.36030576,  0.3341029 ],
       [ 0.60462254,  0.5616927 , -0.3610924 ,  0.46177837, -1.6261365 ,
```

Fig. 7. internal information of victim model

- OS: Ubuntu 16.04.6 LTS

Fig.7은 victim 딥러닝 모델의 정보이고 공격자의 목표는 이를 추출하는 것이다. 실험의 편의를 위해 관리자 권한으로 공격의 3단계를 수행한 후 권한 없이 나머지 단계를 진행했다.

5.2 실험 결과

Fig.8은 모델 인스턴스 심볼테이블의 *'network_nodes'* 요소에 대한 멜트다운 공격으로 배열의 각 아이템을 추출한 결과이다. 각 은닉층의 이름이 아스키 코드로 저장되어 있는 것을 볼 수 있다. 이를 참고하여 모델 구조를 재현한다.

Fig.9는 victim 인공신경망의 가중치 정보를 성공적으로 추출한 결과이다. 각 가중치 값은 little endian의 부동소수점 형태로 변환되어 저장되기 때문에 Fig.10을 참조하여 첫 번째 가중치를 정확히 추출했음을 확인할 수 있으며 첫 번째 가중치에 이어

```
e4 5c a3 00 00 00 00 00 00 00 00 00 00 00 00 00 | .\.............. |
66 6c 61 74 74 65 6e 5f 36 5f 69 6e 70 75 74 5f | flatten_6_input_ |
69 62 2d 30 00 00 00 00 02 00 00 00 00 00 00 00 | ib-0............ |
…
e4 78 a3 01 00 00 00 00 94 af 00 00 00 00 00 00 | .x.............. |
66 6c 61 74 74 65 6e 5f 36 5f 69 62 2d 30 00 00 | flatten_6_ib-0.. |
00 00 00 00 00 00 00 00 f0 10 ce 74 23 7f d4 8f | ...........t#... |
…
e4 78 a3 00 00 00 00 00 00 00 00 00 00 00 00 00 | .x.............. |
64 65 6e 73 65 5f 31 32 5f 69 62 2d 30 00 00 00 | dense_12_ib-0... |
03 00 00 00 00 00 00 00 40 84 a3 00 00 00 00 00 | ........@....... |
```

Fig. 8. model structure extraction result

```
4260b0780: 8e 64 a9 bf 35 e4 88 3e 35 c5 82 3f 71 eb 18 3f
4260b0790: 01 65 85 bf b8 0e c8 be 02 d1 06 84 7d 79 80 3f
4260b07a0: 52 10 b4 bf d6 39 06 c0 84 c1 5f bf e6 2f 88 3e
4260b07b0: c9 24 30 3e 3b c0 38 be b3 45 df 3e 55 f0 24 c0
4260b07c0: …
```

Fig. 9. weight extraction result

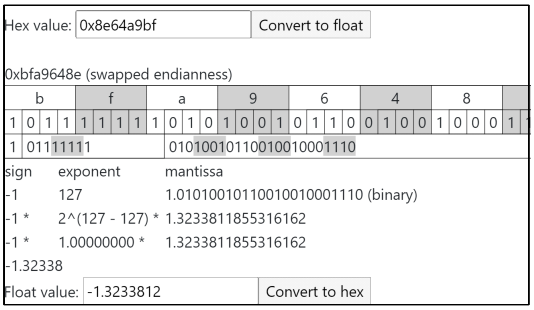

Fig. 10. IEEE 754 single precision decoding

다른 가중치 정보도 연속적으로 저장되어 있음을 볼 수 있다.

또한 딥러닝 서비스로의 쿼리 입력을 목표로 하도록 공격의 4단계를 수정하여 딥러닝 모델의 입력를 추출할 수 있다. 이는 사용자 얼굴 인식이나 음성 인식 등 민감한 데이터가 입력으로 사용되는 경우 큰 위협을 야기한다. Fig.11은 VGG16 이미지 인식 딥러닝 모델[21]에 대해 공격을 수행하여 추출한 입력 데이터이다. 이는 단일 공격으로 추출된 결과이며 반복적인 공격으로 노이즈를 보정하여 추출 정확도를 향상시킬 수 있다.

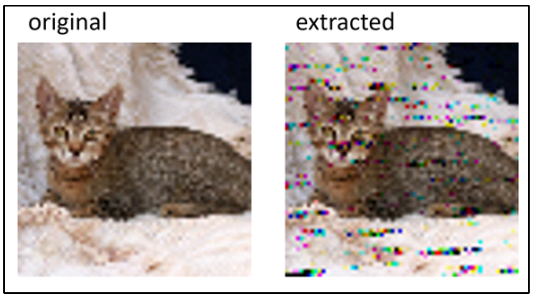

Fig. 11. input image extraction result

VI. 공격 성능 분석

본 연구 이전에도 인공신경망 복원 공격 기법에 대한 다양한 연구가 진행되었다. 하지만 기존의 인공신경망 복원 공격들은 인공신경망의 구조를 복원하는데 그치기에 가중치나 활성함수 종류 등 기타 내부 정보는 복원할 수 없다. 또한 함수사용 모니터링이나 소요시간 만으로 구조를 복원하는 많은 기법들은 인공신경망 구조가 복잡해질수록 정확도가 급격히 감소하는 반면 본 연구에서 제안하는 공격 기법은 인공신경망의 복잡도에 영향을 받지 않는다.

5장에서의 실험은 인공신경망의 구조, 가중치, 입 동일한 방법으로 인공신경망과 관련된 다른 모든 정보를 추출할 수 있다. Table 2는 추출 가능한 정보 측면에서 기존 연구들과의 차이점을 정리한 것이다.

본 논문에서 제시하는 공격의 성능을 평가하기 위해, victim 인공신경망의 가중치 값과 추출해낸 가중치 값을 비교했다. 이를 통해 바이트 수준에서의 추출 정확도와 소요 시간을 측정했다. 해당 실험에서는 멜트다운의 기저로 Flush+Reload를 사용했으며, 5.1절에서 명시한 환경에서의 추출 공격을 분석한 결과 92.875%의 정확도와 1.325kB/s의 추출 속도를 보였다. 이러한 성능은 CPU 및 환경 별로 조금씩 차이를 보이며 다음과 같은 방법으로 최적화 할 수 있다.

최적화 1: 멜트다운 성능 향상

본 연구에서 제안하는 공격의 성능은 멜트다운의 성능에 의존하므로 멜트다운 샘플링 횟수를 증가시키거나 victim과 동일 코어를 사용하도록 설정하는 것으로 전체 공격 파이프라인의 속도와 정확도를 향상시킬 수 있다[1].

최적화 2: Python 고정 해시 활용

심볼테이블 딕셔너리에는 해시와 키 포인터, 값 포인터 정보가 존재한다. 따라서 심볼테이블을 확보하는 것 만으로는 각 요소가 무엇을 뜻하는지 알 수 없고, 멜트다운으로 키 포인터 위치를 추출해야만 해당 요소의 변수명을 파악할 수 있었다. 하지만 Python 3.3 이전 버전의 경우 딕셔너리의 해시 부분에 간단한 해시함수를 사용한다[22]. 따라서 공격자는 키 포인터를 따로 추출할 필요 없이, 자신이 원하는 변수명의 해시를 직접 구한 후 딕셔너리의 해시와 비교해 원하는 변수를 선별할 수 있다. 이 경우 심볼테이블을 해석하는 과정이 단축되어 공격의 성능이 향상된다.

VII. 대응방안

7.1 Kernel Page Table Isolation (KPTI)

KPTI는 멜트다운 보안 취약점을 완화하고 Kernel Address Space Layout Randomization(KASLR)을 우회하려는 시도에 대비하여 커널 강화 기능을 향상시키는 기법이다

Table 2. Table of various neural network stealing techniques and what information they can steal

		structure	learning rate	loss function	weight & bias	model input
oracle	via preduction APIs [10]	△				×
	practical black-box attacks [11]	△				×
	high accuracy and fidelity [12]	△				×
	stealing hyperparam.s in ML [13]	×	×	◐	×	×
hardware side-channel	I know what you see [14]	×	×	×	×	●
	floating-point multiplication [15]	×	×	×	×	●
	CSI NN [16]	●	×	×	◐	×
software side-channel	via timing side-channels [17]	◐	×	×	×	×
	leaky DNN [18]	●	×	×	×	×
	cache telepathy [19]	●	×	×	×	×
	our attack	●	●	●	●	●

● indicates that the technique can successfully steal the corresponding information.
× indicates that the technique cannot extract corresponding information.
◐ indicates that the extraction is only successful with simple neural networks.
△ indicates that the technique does not steal the corresponding information directly, but constructs a substitute network. The contents of such substitute network are distant from the original victim network; hence it cannot be regarded as an extraction.

[23]. 커널 메모리 공간에 대한 페이지 테이블을 별도로 관리하여 사용자 공간에 유효한 커널 메모리 맵핑을 제거하는 것으로 인해 비순차적 실행을 통해 권한이 없는 커널 영역에 접근하려는 시도를 차단한다. 이 기능은 리눅스 커널 버전 4.15부터 적용되었다.

하지만 유저 페이지 테이블에서 커널 영역을 완전히 제거할 수 는 없기에 KPTI는 멜트다운 공격을 완전히 막지 못한다. 더욱이 유저 프로세스는 커널 작업이 필요할 때마다 커널 페이지 테이블을 호출해야 하기 때문에 시스템 콜(system call)이나 IO 인터럽트가 빈번한 경우 최대 30%의 성능 저하가 발생한다[24].

7.2 하드웨어 패치

멜트다운은 하드웨어적 결함에 기인하기에 근본적인 해결책은 하드웨어를 수정하는 것이다. 2018년 초 멜트다운 취약점이 공개된 이후, Intel은 다음 세대의 CPU부터 이에 대응하는 하드웨어 패치를 적용하였다. 패치된 CPU를 사용할 경우, 멜트다운을 이용한 공격 시나리오가 무력화된다. 현재 출시되고 있는 인텔 7세대 CPU 이후 제품들이 멜트다운 공격에 내성이 있고 새롭게 생산하고 있는 전 세대 CPU들도 패치가 되어 나오고 있다. 하지만 2018년도 이전에 생산된 하드웨어로 구성된 서버의 장비를 일괄적으로 교체하는 것은 불가능하기에 아직까지 취약한 하드웨어를 사용하는 경우가 많다[25].

멀티-테넌시 환경의 시스템을 구축하는 경우 해당 공격에 패치가 완료된 하드웨어를 이용해야 한다. 하지만 L1 Terminal Fault(L1TF)[2], TSX Asynchronous Abort(TAA)[3] 등 멜트다운에 이어 기존의 방어기법을 우회하는 변종 기법들이 계속해서 발견되고 있으므로 멜트다운 패치가 적용 되었더라도 다른 기법으로 대체하여 공격 시나리오를 구성할 수 있다. 그러므로 시스템 관리자는 하드웨어 취약점과 관련하여 지속적인 관리가 필요하다.

VIII. 결 론

본 연구는 기존 연구들의 단점을 보완하는 동시에 그들이 취하지 못하는 정보까지 추출이 가능한 기법을 제시한다. 이는 딥러닝 프로세스 메모리에 접근해 내부 정보를 직접 추출해오는 방식이기 때문에 인공신경망의 구조에 한정되지 않고 하이퍼파라미터, 입력 데이터 등 관련한 데이터를 모두 복원할 수 있다. 따라서 기존의 인공신경망 추출 공격들보다 중대한 위협을 야기한다.

또한 멜트다운 공격이 내포하는 한계점을 해결할 수 있는 방법을 제시한다. 기존 기법들은 데이터를 선별할 방식이 없기에 특정 목적을 갖고 실질적인 공격 시나리오를 구성하기에 부족한 반면, 본 연구는 심볼테이블을 포착해 데이터의 위치를 파악하는 방안

을 도입하여 이를 해결했다.

이러한 접근방식은 동적으로 할당되는 변수의 위치를 추적하기에 적합하다. 통제된 실험환경에서 고정된 위치에 대한 추출을 선보이는 기존 연구들과 달리, 본 연구의 실험은 런타임에 동적할당된 데이터를 추적하여 목표 데이터에 도달한다. 이는 Python을 기반으로 한 실험으로 제시되었지만, 심볼을 관리하는 영역을 분석하여 목표 위치를 확보하는 접근법은 다른 언어에서의 동적 매핑에도 적용 가능하다. 따라서 기존의 고정적인 위치에 대한 공격들도 본 연구의 기법을 적용하여 동적인 환경까지 확장될 수 있을 것으로 예상된다.

References

[1] Lipp, M., et al. "Meltdown: Reading kernel memory from user space," 27th USENIX Security Symposium, pp. 973-990, Aug. 2018.

[2] Weisse, O., et al. "Foreshadow-NG: Breaking the virtual memory abstraction with transient out-of-order execution," Aug. 2018.

[3] Schwarz, M., et al. "ZombieLoad: Cross-privilege-boundary data sampling," Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, pp. 753-768, Nov. 2019.

[4] Yarom, Y. and Falkner, K. "FLUSH+RELOAD: a high resolution, low noise, L3 cache side-channel attack," 23rd USENIX Security Symposium, pp. 719-732, 2014.

[5] Moosavi-Dezfooli, S.M., et al. "Deepfool: a simple and accurate method to fool deep neural networks," Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2574-2582, June. 2016.

[6] Fredrikson, M., et al. "Model inversion attacks that exploit confidence information and basic countermeasures," Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 1322-1333, Oct. 2015.

[7] Rossum, G.V. and Drake Jr, F.L. "Python tutorial," Amsterdam: Centrum voor Wiskunde en Informatica, vol. 620, April. 1995.

[8] Zhang, Y., et al. "Cross-tenant side-channel attacks in PaaS clouds," Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pp. 990-1003, Nov. 2014

[9] Inci, M.S., et al. "Co-location detection on the cloud," International Workshop on Constructive Side-Channel Analysis and Secure Design, Springer, Cham, pp. 19-34, April. 2016.

[10] Tramèr, F., et al. "Stealing machine learning models via prediction apis," 25th USENIX Security Symposium, pp. 601-618, Aug. 2016.

[11] Papernot, N., et al. "Practical black-box attacks against machine learning," Proceedings of the 2017 ACM on Asia conference on computer and communications security. pp. 506-519, April. 2017.

[12] Jagielski, M. et al. "High Accuracy and High Fidelity Extraction of Neural Networks," 29th USENIX Security Symposium, Aug. 2020.

[13] Wang, B., and Gong, N.Z. "Stealing hyperparameters in machine learning," 2018 IEEE Symposium on Security and Privacy, pp. 36-52. May 2018.

[14] Wei, L., et al. "I know what you see: Power side-channel attack on convolutional neural network accelerators," Proceedings of the 34th Annual Computer Security Applica-

tions Conference, pp. 393-406, Dec. 2018.
[15] Dong, G., et al. "Floating-Point Multiplication Timing Attack on Deep Neural Network," 2019 IEEE International Conference on Smart Internet of Things, pp. 155-161, Aug. 2019
[16] Batina, L., et al. "CSI NN: Reverse Engineering of Neural Network Architectures Through Electromagnetic Side Channel," 28th USENIX Security Symposium. pp. 515-532. Aug. 2019.
[17] Duddu, V., et al. "Stealing neural networks via timing side channels." arXiv preprint, arXiv:1812.11720, Dec. 2018.
[18] Wei, J., et al. "Leaky DNN: Stealing Deep-learning Model Secret with GPU Context-switching Side-channel," 2020 IEEE/IFIP International Conference on Dependable Systems and Networks. June. 2020.
[19] Yan, M., et al. "Cache Telepathy: Leveraging Shared Resource Attacks to Learn DNN Architectures," 29th USENIX Security Symposium, pp. 2003-2020, Aug. 2020.
[20] Deng, L. "The MNIST database of handwritten digit images for machine learning research [best of the web]," IEEE Signal Processing Magazine, 29.6, pp. 141-142, Nov 2012.
[21] Simonyan, K., and Zisserman, A. "Very deep convolutional networks for large-scale image recognition," arXiv preprint, arXiv:1409.1556. Sep. 2014.
[22] Goodrich, M.T., et al. "Data structures and algorithms in Python," John Wiley & Sons Ltd, 2013.
[23] Gruss, D., et al. "Kaslr is dead: long live kaslr," International Symposium on Engineering Secure Software and Systems, pp. 161-176, Springer, Cham, July. 2017.
[24] Gregg, B. "KPTI/KAISER Meltdown Initial Performance Regressions," http://www.brendangregg.com/blog/2018-02-09/kpti-kaiser-meltdown-performance.html, Feb. 2018.
[25] Zhu, J., et al. "CPU security benchmark," Proceedings of the 1st Workshop on Security-Oriented Designs of Computer Architectures and Processors, pp. 8-14, Jan. 2018.

〈저 자 소 개〉

정 호 용 (Hoyong Jeong) 학생회원
2017년 3월: 고려대학교 컴퓨터학과 입학
2017년 3월~현재: 고려대학교 컴퓨터학과 학부과정
〈관심분야〉 인공지능 보안, 시스템 보안, 부채널 공격

류 도 현 (Dohyun Ryu) 정회원
2020년 2월: 광운대학교 전자공학과 졸업
2020년 9월~현재: 고려대학교 컴퓨터학과 석사과정
〈관심분야〉 정보보호, 부채널, 시스템 취약점, 암호

허 준 범 (Junbeom Hur) 종신회원
2001년 2월: 고려대학교 컴퓨터공학 졸업
2005년 8월: 한국과학기술원 전산학 석사
2009년 8월: 한국과학기술원 전산학 박사
2009년 9월~2011년 8월: University of Illinois at Urbana-Champaign 박사후 연구원
2011년 9월~2015년 2월: 중앙대학교 컴퓨터공학부 조교수
2015년 3월~2016년 8월: 고려대학교 컴퓨터학과 조교수
2016년 9월~현재: 고려대학교 컴퓨터학과 부교수
〈관심분야〉 응용 암호, 네트워크 보안, 클라우드 보안, 시스템 취약점