ISSN 2586-1816
Journal of Convergence for Information Technology e-ISSN 2586-4440
Vol. 8. No. 6, pp. 209-215, 2018 DOI : https://doi.org/10.22156/CS4SMB.2018.8.6.209

의사결정트리 기반 머신러닝 기법을 적용한 멜트다운 취약점 동적 탐지 메커니즘

이재규[1], 이형우[2*]
[1]한신대학교 컴퓨터공학부 연구원, [2]한신대학교 컴퓨터공학부 교수

Meltdown Threat Dynamic Detection Mechanism using Decision-Tree based Machine Learning Method

Jae-Kyu Lee[1], Hyung-Woo Lee[2*]
[1]Researcher, Division of Computer Engineering, Hanshin University
[2]Professor, Division of Computer Engineering, Hanshin University

요 약 본 논문은 동적 샌드박스 도구를 이용하여 최근 급증하고 있는 멜트다운(Meltdown) 악성코드를 사전에 검출 및 차단하는 방법을 제시하였다. 멜트다운 공격 취약점에 대한 패치가 일부 제공되고 있으나 여전히 해당 시스템의 성능 저하 등의 이유로 의도적으로 패치를 적용하지 않는 경우가 많다. 이와 같이 적극적인 패치가 적용되지 않은 인프라를 위해 머신러닝 기법을 이용하여 기존의 시그니처 탐지 방식의 한계를 극복하는 방법을 제시하였다. 우선 멜트다운의 원리를 이해하기 위해 가상 메모리, 메모리 권한 체크, 파이프 라이닝과 추측 실행, CPU 캐시 등 4가지의 운영체제 구동 방식을 분석하고 이를 토대로 멜트다운 악성코드에 리눅스 strace 도구를 활용하여 데이터를 추출하는 메커니즘을 제공하였으며 이를 기반으로 의사 결정 트리 기법을 적용하여 멜트다운 악성코드를 판별하는 메커니즘을 구현하였다.

주제어 : 멜트다운 공격, 운영체제, OS 취약점, 의사결정트리, 동적 탐지

Abstract In this paper, we propose a method to detect and block Meltdown malicious code which is increasing rapidly using dynamic sandbox tool. Although some patches are available for the vulnerability of Meltdown attack, patches are not applied intentionally due to the performance degradation of the system. Therefore, we propose a method to overcome the limitation of existing signature detection method by using machine learning method for infrastructures without active patches. First, to understand the principle of meltdown, we analyze operating system driving methods such as virtual memory, memory privilege check, pipelining and guessing execution, and CPU cache. And then, we extracted data by using Linux strace tool for detecting Meltdown malware. Finally, we implemented a decision tree based dynamic detection mechanism to identify the meltdown malicious code efficiently.

Key Words : Meltdown Attack, Operating System, OS Vulnerabilities, Decision Tree, Dynamic Detection.

1. 서론

멜트다운(Meltdown, "붕괴") 방식은 대부분의 인텔 CPU와 일부 ARM CPU를 대상으로 시스템 보안 취약점을 악용한 최신 공격 방식이다. 멜트다운 버그는 마이크로프로세서가 컴퓨터의 메모리의 전체를 볼 수 있도록 프로그램의 접속을 허용한 것을 토대로 컴퓨터 내부 파일 전체에 접근할 수 있다는 문제점에 근거하고 있다. 즉,

*This work was supported by Hanshin University Research Grant.
*Corresponding Author : Hyung-Woo Lee(hwlee@hs.ac.kr)
Received October 18, 2018 Revised November 12, 2018
Accepted December 20, 2018 Published December 31, 2018

악의적인 응용 프로그램을 통해서 운영체제의 시스템 메모리에 접근할 수 있다는 것을 의미하기 때문에 결과적으로 커널 단에서 이루어지는 모든 작업이 외부로 유출되거나 노출된다는 것을 뜻한다. 멜트다운 공격은 발견 초기에 CVE-2017-5754로 등재되었으며, 2018년 1월 3일 구글에서 최초로 공식 발표하였다. 멜트다운 공격은 1995년경부터 제공된 CPU의 비순차적 명령어 처리(Out-of-order Execution) 기술과 추측 실행을 악용한 보안 취약점에 해당한다. 일반 pc의 메모리는 각각의 프로세스들에게 일정한 범위의 영역을 할당하게 되고, 각 커널과 프로세스들은 각각의 page table을 가지고 있다. 일반적인 user process들에서만 데이터를 처리하면 별다른 문제는 없겠으나, syscall 등의 요청이 필요한 경우 커널 내부로 들어가야 한다. 따라서 이를 위해 일반 user process인 경우 역시 kernel 영역의 가상/물리 메모리 테이블을 가지고 있다. 하지만 보호 비트로 접근제어 되기 때문에 일반적으로 kernel 영역의 메모리 주소를 요청할 수 없다[1].

하지만, 멜트다운 공격은 CPU내에서 코드 실행시 발생하는 예외 처리 핸들러(exception handler)를 처리하는 과정에서 캐시에 삭제되지 않고 남아 있는 데이터 정보를 이용하여 구동되는 보안 취약점을 의미한다. 시스템 성능 향상을 위해 커널이 사용하는 가상 메모리 영역 매핑 과정에서의 취약점을 악용하는 것으로 이를 해결하기 위해서는 결국 커널 공간을 가상 메모리에 매핑하는 것을 가급적 최소화해야 한다.

따라서 이 경우 파일시스템 I/O 쪽에서 영향을 크게 받을 것으로 예상되며, 리눅스 OS 패치 전후의 테스트 결과에 따르면 성능이 절반 이하로 크게 감소한 경우도 보고되었으며, SQL DB와 컴파일러, 데이터베이스 서버 등에서 성능 저하가 크게 나타나는 것으로 알려졌다.

이에 본 논문에서는 멜트다운 악성코드를 동적으로 탐지하기 위해 일차적으로 샌드박스에서 사전에 차단하는 방법을 제시하였다. 또한 검출 성능을 향상시키기 위해서 의사결정트리 기반 머신러닝 기법을 적용하였다. 취약점 패치 버전이 등장하였으나 성능 저하의 이유로 내부 시스템 혹은 성능에 영향이 민감한 IoT 장비에 대해서는 의도적으로 패치를 실행하지 않는 경우가 많다. 따라서 패치가 되지 않은 인프라를 위해 의사 결정 트리 기반 머신러닝 기법을 적용하여 멜트다운 취약점을 동적으로 탐지하며 기존 시그니처 탐지 방식의 한계를 극복하는 방법을 제시하고자 한다.

2. 멜트다운 공격

2.1 정의 및 위험도

CVE-2017-5754를 통해 발표된 멜트다운 취약점은 CPU 결함을 이용한 시스템 취약점으로 다음 Fig. 1과 같이 CVSS 기본 점수(Base Score)가 5.6으로 높지는 않다. 하지만 데이터 무결성(Integrity), 가용성(Availability)에 전혀 영향이 없는 것에 비해 높은 기본 점수가 배정된 이유는 기밀성(Confidentiality)에 심각한 위험을 초래할 수 있기 때문에 결과적으로 치명적인 취약점이 될 수 있다는 것을 의미한다[2].

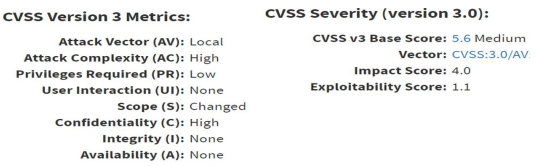

Fig. 1. CVE-2017-5754 Detail

멜트다운은 지역 악성코드(local exploit) 형태의 공격이기 때문에 호스트에서 악성 프로세스가 실행된다는 것을 전제로 한다. 일단 악성 프로세스가 실행되기 시작하면 감염된 호스트의 모든 정보를 추출하는 것이 가능하다. 이는 프로세스 간의 정보 독립성을 무시하며 심지어 유저의 권한으로 커널 정보까지 빼내 올 수 있기 때문에 가능한 방식이다. 따라서 멜트다운 취약점에 대한 자동 탐지 및 대응 메커니즘이 개발되어야 한다.

2.2 취약점 발생 원인

멜트다운 취약점은 Intel CPU 기반 시스템에서 주로 발생하고 AMD CPU에서 발생하지 않는다. 그 이유는 두 기업의 비순차적 명령어 처리(Out-of-order Execution) 기술의 차이에서 발생한다. 두 기업 모두 명령어의 처리율을 높이기 위해 각각의 명령어를 병렬로 구동시켜 처리 흐름을 유지하는 파이프 라이닝(Pipe lining) 기술을 사용한다. 하지만 이 과정에서 더욱더 높은 처리율을 제공하기 위해 비순차적 명령어의 방법론 중 일부인 추측 실행 기법을 사용한다. 추측 실행은 분기 문에서 결과를 예측하여 CPU에서 명령어를 미리 처리하는 기술이다.

여기서 추측 실패시 돌아가는 시점(rollback)이 Intel과 AMD가 다르므로 멜트다운 공격이 Intel CPU에서만 발생하는 것이다. Fig. 2를 보면 AMD는 추측 실패 인터럽트 시점이 excute 이전이고 Intel은 commit 과정 이전이다. 여기서 롤백 시점의 writeback 단계 포함 유무에 따라서 멜트다운 공격 가능성이 구별 된다. writeback 단계는 CPU 캐시에 데이터를 기록하는 단계이다. Intel CPU는 추측 실패로 인한 롤백 시에 캐시에 그대로 데이터가 남게 되기 때문에 멜트다운 취약점이 발생하는 것이다.

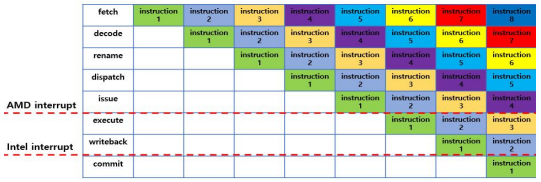

Fig. 2. Intel, AMD CPU Interrupt rollback point

3. 멜트다운 공격 방식 상세 분석

3.1 멜트다운 공격 원리

GitHub의 멜트다운 공격코드를 통해 Fig. 3과 같은 순서도를 작성할 수 있다[3]. 여기서 캐시에 있는 데이터를 어떻게 빼내오는지 확인할 필요가 있다. 부 채널 사이드 공격의 일종인 캐시 타이밍 어택을 이용해서 CPU 캐시에 있는 데이터를 가져온다.

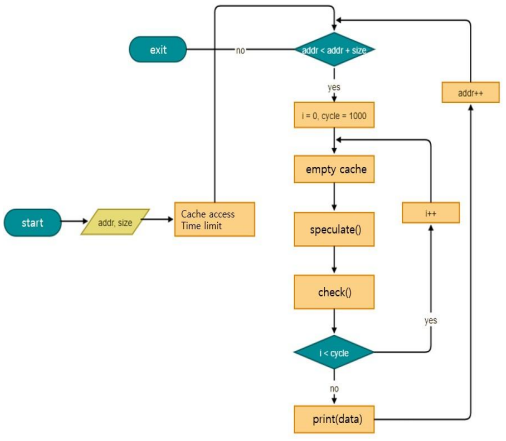

Fig. 3. Meltdown Attack Flowchart

멜트다운의 공격코드는 Fig. 4에 정의된 4행의 어셈블리어가 핵심이다. 공격코드는 4행으로 구성된 어셈블리어의 반복 수행을 통해 공격의 성공 가능성을 높인다. 환경 요인(CPU 사용량 등)에 따라 멜트다운 공격이 실패할 수도 있기 때문이다. 먼저 target 사용자 변수를 선언해 놓고 Fig. 4의 첫 번째 라인 movzx명령어를 통해 커널 주소(kernel_addr)에 접근한다. 그리고 eax 레지스터에 담긴 커널 데이터의 값에 target 주소를 더한 후 movzx 명령어를 실행 한다. 그러면 target의 주소+커널의 데이터 값이 CPU 캐시에 담기게 된다. 그러나 CPU 캐시에 있는 데이터는 일반적인 방법으로 추출 할 수 없기 때문에 캐시 데이터 추출을 위해서 저장장치 접근 속도의 차이를 이용한 캐시 타이밍 공격(Cache Timing Attack)을 사용해야 한다[4].

```
speculate()
movzx (%[kernel_addr]), %%eax
shl $12, %%eax
jz meltdown
movzx (%[target], %%eax, 1), %%ebx
```

Fig. 4. Meltdown attack Assembly language(x86)

Fig. 4는 어셈블리어 첫 줄의 movzx 명령어가 메모리 접근권한이 없기 때문에 인터럽트가 발생되는 과정을 제시하고 있다. 하지만 파이프 라이닝, 추측 실행 등으로 인해 4번째 줄의 어셈블리어 명령어까지 실행이 되는 것이다. 공격코드는 Fig. 5의 그림 위에 제시된 것처럼 target+0*4096, target+1*4096 ... target+n*4096 순으로 접근 속도의 차이를 계산하기 시작한다. 이중 가장 빠른 접근속도를 나타내는 target+x*4096의 x값이 커널의 데이터 값이 된다. 중간에 시프트 명령어(shl)를 통해 2의 12승 곱하기 연산(4096)을 실행하는 이유는 캐시 쓰기 정책 중 하나인 지역 참조성 때문이다. 지역 참조성은 주기억장치에서 캐시로 쓰기 작업을 할 때 해당 데이터가 존재하는 위치의 일정 영역을 모두 캐시로 가져오는 정책이다. 여기서는 2의 12승(4096)으로 설정했으며 각 CPU 아키텍처에 따라 값은 변경 될 수 있다.

만약 shl 연산이 없다면 Fig. 5의 아래 그림 부분처럼 캐시의 지역참조 정책에 의해 target+0, ... ,target+n 모든 데이터의 접근속도가 유사하게 나와 캐시타이밍 어택을 사용할 수 없게 된다.

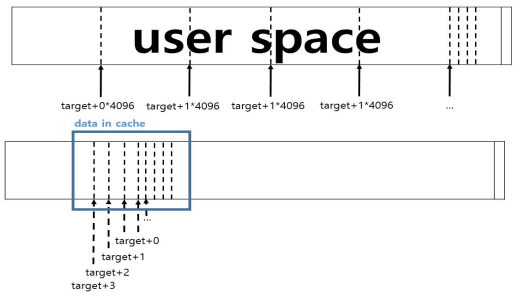

Fig. 5. cache Locality of reference

3.2 멜트다운 공격 방식

멜트다운 취약점을 사용한 악성 프로세스가 전체 프로세스의 정보를 볼 수 있는 이유는 커널 영역을 볼 수 있기에 가능한 일이다. Fig. 6처럼 대부분의 운영체제는 유저 프로세스에서 문제가 발생하더라도 전체 시스템에 영향이 없도록 하기 위해 커널과 유저의 메모리 영역을 나누고 커널 영역을 공유하는 형태를 취한다. (x86_64 기준) 커널 영역은 다시 Fig. 7처럼 역할에 따라 메모리 영역이 나누어진다[5]. 여기서 direct mapping space(64TB)는 물리 메모리 영역 전체를 매핑하는 곳이다. 따라서 이 영역을 볼 수 있다는 것은 결국 공격자는 멜트다운 공격을 통해서 운영체제 실행시 커널내 모든 프로세스의 정보를 엿볼 수 있다는 것을 의미한다.

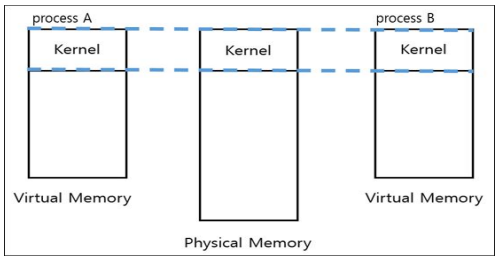

Fig. 6. Physical Memory and Virtual Memory

따라서 멜트다운을 이용한 공격 방식은 OpenSSL Heartbleed 공격[6]과 유사한 형태를 취하게 된다. 왜냐하면 direct mapping space를 통해 물리 메모리를 덤프하는 공격 방식이 Heartbleed의 물리 메모리 덤프 공격과 동일하기 때문이다 그렇기 때문에 공격자는 중요한 정보를 C&C로부터 가져오기 위해서는 일정 시간 동안 일정 영역을 지속해서 공격할 필요가 있다.

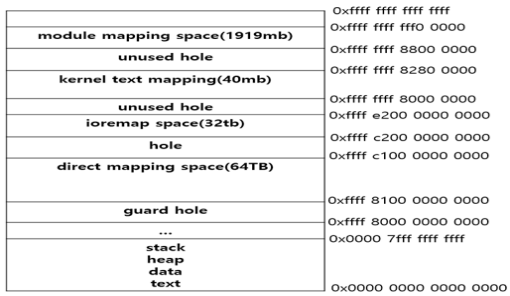

Fig. 7. x86_64 linux kernel map

4. 멜트다운 공격 탐지

4.1 SIGSEGV 시그널 발생 모니터링

멜트다운 공격은 권한이 없는 영역에 접근을 시도하기 때문에 시그널이 발생할 수밖에 없다. 시그널이란 프로세스 또는 프로세스 그룹간 통신을 위해 사용하는 통신 기법중 하나이다. Fig. 8을 보면 멜트다운 취약점은 필연적으로 SIGSEGV 시그널이 발생하게 된다. 이는 유저 모드에서 권한이 없는 커널 메모리 영역에 접근하려고 시도하기 때문이다.

SIGSEGV는 si_addr속성을 통해 어떤 주소에 접근을 시도하다가 실패했는지를 알리게 된다. 2.4에서 멜트다운 공격이 성립하기 위해서는 일정 시간 동안 일정 영역을 지속해서 공격해야 한다고 설명했다. 즉 si_addr 값과 시그널 발생 횟수를 판단하며 단순 소스코드 오류에 의한 시그널 발생인지 멜트다운 공격인지 판별할 수 있다.

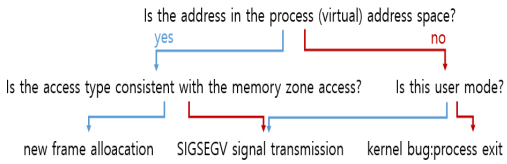

Fig. 8. page fault handler

4.2 멜트다운 공격 탐지 메커니즘 제시

Fig. 9와 같이 휴리스틱 기반 방식 동적 탐지를 사용한다면 멜트다운 공격인지 여부를 확인할 수 있다. 먼저 si_addr이 커널 영역인지 확인한다. 그리고 주기적으로 발생하는지를 확인한다. 마지막으로 si_addr에 나타나는 주소들이 인접해 있는지 확인한다.

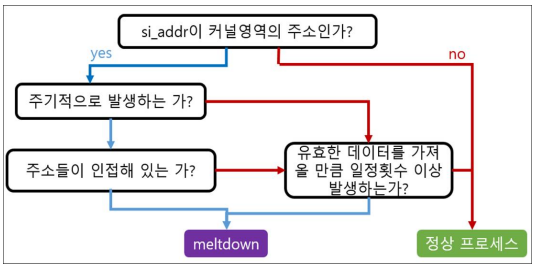

Fig. 9. Page Fault Handler

strace, ps 명령어를 사용하여 탐지 도구를 제작 하였다. process_list.py는 현재 실행 중인 프로세스들의 pid를 가져 오도록 구현하였고, open_process.py는 프로세스를 strace로 추적하여 SEGV_MAPERR가 발생 하는지 확인하며, filter_addr.py는 SEGV_MAPERR가 발생한 주소가 커널 영역인지 그리고 주기적으로 발생하는지 또한 인접한 주소들에서 발생하는지 확인하도록 구현하였다. Fig. 10, 12는 Fig. 9의 휴리스틱 탐지를 python 스크립트로 작성한 것이다. 1) open_process.py를 통해 타겟 프로세스에 strace로 attach를 한다. 2) Fig. 11과 같이 주소, 접근 시도회수의 쌍으로 python dict type이 생성된다. 3) 생성된 dict를 가지고 filter_addr에서 검증을 시작한다. 4) 필터링 결과 만약 멜트다운 악성 프로세스라면 "meltdown"을 출력한다.

```python
from subprocess import *

def strace_pid(pid):
    print("hi")
    p = Popen("sudo strace -p 35051", shell=True, stdout=PIPE, stderr=PIPE)

    stdout_=b"0x64"
    stderr_=b"0x64"

    try:
        call(['kill','-9','35051'],shell=True)
        stdout_, stderr_ = p.communicate(timeout=5)
    except TimeoutExpired:
        print("time out")
    print(stdout_)
    print(stderr_)

    listStraceOutput = stderr_.decode("ascii").split("\n")
    dictAddrCount = {}
    listSigegv = []
    for lineStrace in listStraceOutput:
        if(lineStrace.find("SEGV_MAPERR") != -1):
            if(lineStrace.find("SIGSEGV") != -1):
                index_si_addr = lineStrace.find("si_addr")
                listSigegv.append(lineStrace[index_si_addr+10:index_si_addr+26])
                try: dictAddrCount[lineStrace[index_si_addr+10:index_si_addr+26]] += 1
                except: dictAddrCount[lineStrace[index_si_addr+10:index_si_addr+26]] = 1
    print(dictAddrCount)

strace_pid(1)
```

Fig. 10. open_process.py

Fig. 11은 공격자에 의해서 0xffff 8800 0000 0000 ~ 0xffff 8800 0000 000f 주소내 데이터를 추출하는 멜트다운 공격 프로세스의 시그널 정보 추출 결과이다. 추출한 결과를 가지고 filter_addr.py를 실행할 때 멜트다운 여부를 탐지하는 접근 시도 회수의 임계치를 결정해야하는데 서버의 부하 상태, 하드웨어 스펙 등에 따라 달라질 수 있다. 의사결정트리를 사용한다면 각 서버에 맞는 멜트다운 탐지를 위한 임계치를 찾을 수 있을 것이다.

```
penguin@ubuntu:~/sol_melt$ sudo python3 open_process.py
{'ffff880000000002': 1000, 'ffff880000000006': 1000, 'ffff880000000009': 1000,
 'ffff880000000005': 1000, 'ffff88000000000c': 1000, 'ffff88000000000a': 1000,
 'ffff880000000000': 1000, 'ffff880000000001': 1000, 'ffff88000000000f': 1000,
 'ffff880000000003': 1000, 'ffff88000000000d': 1000, 'ffff88000000000b': 1000,
 'ffff880000000004': 1000, 'ffff880000000007': 1000, 'ffff88000000000e': 1000,
 'ffff880000000008': 1000}
```

Fig. 11. Meltdown Attack Detection Result

```python
def filter_addr(setMemoryAccess):
    kernelAddr = 0x00007fffffffffff
    listMemoryAddr = list(setMemoryAccess.keys())

    checkKernelAddr = False
    KernelAddr_ = ''
    checkCycle = False
    checkAdjacent = False
    checkExeCount = False

    for addr in listMemoryAddr:
        if(int(addr,16)>=kernelAddr):
            print("1:kernel address!!")
            checkKernelAddr=True
            KernelAddr_ = addr
            break

    beforeAddr = listMemoryAddr[0]
    per = 4

    if(checkKernelAddr):
        for addr in listMemoryAddr[1:]:
            if(int(addr,16) - int(beforeAddr,16)) < per:
                print("2:adjacent addr!!")
                checkAdjacent = True
                break
            beforeAddr = addr
    else:
        print("not meltdown")

    if(checkAdjacent):
        print("meltdown")
    else:
        if(setMemoryAccess[KernelAddr]>2):
            checkExCount = True
            print("2:count over")
        print("not meltdown")
```

Fig. 12. filter_addr.py

4.3 임계치 측정을 위한 의사결정트리 적용

의사결정트리(decision tree learning) 기반 머신러닝 [7]은 관측 값과 목표 값을 연결해주는 예측 모델로 데이터 분류에 사용할 수 있다. 몇몇 입력 변수를 바탕으로 목표 변수의 값을 예측하는 모델을 생성하는 것이 결정트리의 목표이다. 결정트리를 구성하는 각 노드에서 최적의 부분값을 찾아내기 위해 휴리스틱 기법을 사용한다.

Fig. 9에서 제시된 메커니즘의 분기문을 각 노드로 두는 의사결정트리를 사용한다면 각 호스트에 맞는 적절한 멜트다운 탐지 여부를 결정하는 임계치를 결정할 수 있다. 다만 의사결정트리 모델을 결정하는 임계치는 멜트다운 취약점이 발생하는 하드웨어 스펙, 부하 상황 등에 따라 크게 달라 질 수 있다.

Fig. 10을 이용하여 측정한 값을 좌표로 입력하면 Fig. 13과 유사하게 나올 것이다. 그리고 X축의 커널 주소, Y축의 접근 시도회수의 임계치를 찾는다. 임계치가 측정되면 Fig 12에 설정된 임계치를 수정하여 모델을 완성한다.

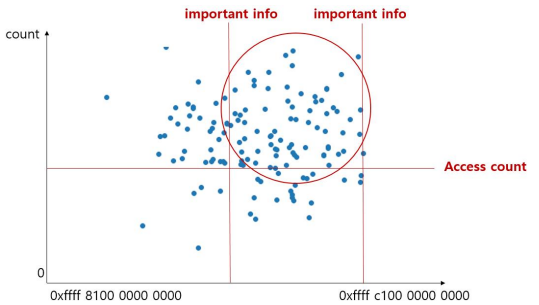

Fig. 13. Meltdown attack detection with decision tree

4.4 CPU/메모리 부하에 따른 임계치 측정

기존 네트워크 취약점을 이용한 시니핑 공격[10]과 데이터에 대한 ATP 공격[11] 형태 및 악성코드를 이용하여 운영체제의 취약점을 공격[12] 형태와 달리 멜트다운 공격은 CPU 결함에서 발생하는 취약점을 악용한 방식이기 때문에 호스트의 환경에 따라 의사 결정트리에서 측정되는 적정 임계치가 실행 환경에 따라 다를 수 있다. 서버 부하가 많다는 것은 프로세스 자원 사용량이 많다는 것을 의미한다. 서버 부하라도 I/O, CPU 두 가지로 나눌 수 있는데 직접적인 영향을 끼치는 것은 후자이다. 그리고 CPU 코어 자체 부하보다 프로세스 스위치가 많은 상황일수록 cache 값 변동이 자주 일어나기 때문에 멜트다운 취약점에 끼치는 영향이 더 크다. Table 1은 멜트다운 공격 중 3번 연속 동일한 데이터를 추출하는데 성공했을 때 측정한 커널 내 특정 영역에 대한 접근 시도 회수를 기록한 것이다. 아래 표를 확인하면 코어(CPU) 사용량 보다는 메모리 사용량이 클수록 접근 회수의 임계치가 커지는 것을 확인 할 수 있다.

Table 1. Threshold value variation with load

Test	Stress for 1 CPU	Stress for 2 CPU	Stress for 3 CPU	Stress for 4 CPU
Stress for 1 MEM	7	8	7	4
Stress for 2 MEM	24	18	19	17
Stress for 3 MEM	87	86	78	70
Stress for 4 MEM	99	94	97	76

4.5 탐지 방식 비교

Table 2에는 시그니처 및 휴리스틱 기반 탐지 방법과 본 연구에서 제시한 의사 결정트리 탐지 방식에 대한 비교 결과를 제시하였다. 시그니처 방식을 이용할 경우 특정 바이너리 부분에 대한 수정 과정을 통해 hash 값을 변경할 수 있으며 에 의한 우회가 가능하다. 하지만 휴리스틱, 결정트리 두 기법은 동적탐지를 기반으로 하기 때문에 단순 값 변경에 의한 우회가 불가능하다.

휴리스틱 탐지는 임계치를 엔지니어 임의로 설정하기에 접근 시도 회수를 변경하면 우회가 가능하다. 하지만 결정트리는 서버에 부하, 스펙에 맞는 모델을 제시하기에 커널 주소 접근 시도 회수 임계치 변경에 따른 우회를 막을 수 있다.

Table 2. Signature vs Heuristic vs Decision Tree

Compare	CPR[8]	ENDGAME[9]	This Work
Used Mechanism	Signature DB	Heuristic	Decision Tree
Detect method	Specific Value(melt, speculate, etc.)	Branch check	Machine learning
Hash bypass detection	x	o	o
Access count bypass detection	x	x	o

5. 결론

본 논문에서는 의사 결정 트리 기반 머신러닝 기법을 적용하여 기존 시그니처 탐지 방식의 한계를 극복하는 방법을 제시하였다. 리눅스 strace 도구를 활용하여 멜트다운 기반 악성코드로부터 추출되는 데이터를 탐지한 다음, python 기반 휴리스틱 기법을 사용한 탐지 스크립트를 개발하였다. 그리고 악성코드에 대해서 X축 변수에 커널 참조 값, y축 변수에 참조 회수를 설정 한 다음 의사 결정 트리를 이용하여 멜트다운 여부를 결정하는 임계치를 측정하는 더욱더 개선된 메커니즘을 제시하였다. 마지막으로 부하 테스트를 통해 프로세스 스위치가 많은 상황에서 임계치가 올라가는 것을 확인하였고 시그니처 기반 탐지[8] 및 휴리스틱 기반 탐지 방법[9]과 본 연구에서 제시한 결정트리 기반 탐지 방식을 각각 비교하였다. 비교 결과 본 연구에서 제시한 기법은 기존 검출 방법과 달리 의사 결정 트리 방식 기반으로 구동하여 Hash bypass 및 Access count bypass 문제점을 해결할 수 있

다는 것을 확인할 수 있었고, 따라서 기존 기법보다 향상된 검출 성능을 제공한다.

실시간 meltdown 탐지를 위해 (1) 현재 실행중인 프로세스에 strace로 attach하고, (2) attach 후 일정시간 동안 발생한 시그널 데이터를 추출하며, (3) 의사결정트리 기반 머신러닝 기법을 적용하여 나온 임계치를 filter_addr.py에 적용하였으며 최종적으로 멜트다운 공격을 판별할 수 있었다. 향후 서버 부하 또는 스펙에 따른 모델을 각각 만들어 샌드박스에 적용한다면 더욱더 개선된 탐지가 가능할 것이다.

REFERENCES

[1] M. Lipp. et al. (2018). *"Meltdown: Reading Kernel Memory from User Space.* https://meltdownattack.com/meltdown.pdf.

[2] CVE-2017-5754 Detail, NIST (2017). https://nvd.nist.gov/vuln/detail/CVE-2017-5754.

[3] paboldin. (2018). *meltdown-exploit.* github. https://github.com/paboldin/meltdown-exploit.

[4] *Timing attack,* WIKIPEDIA. (2018). https://en.wikipedia.org/wiki/Timing_attack.

[5] S. J. Paek & J. M. Choi. (2015). *Linux Kernel Internal*, ArtStudio Book.

[6] *Recommendation for countermeasures against OpenSSL vulnerability (HeartBleed).* KrCert, https://www.krcert.or.kr/data/secNoticeView.do?bulletin_writing_sequence=20884.

[7] Decision tree learning, WIKIPEDIA. (2018).https://en.wikipedia.org/wiki/Decision_tree_learning.

[8] I. Erez, M. Daniel, A. Yoav, G. Aviv & O. Ben. (2018). *Detection of the Meltdown and Spectre Vulnerability.* Check Point Research. https://research.checkpoint.com/detection-meltdown-spectre-vulnerabilities-using-checkpoint-cpu-level-technology/

[9] Code Pierce. (2018). *Detecting Spectre and Meltdown Using Hardware Performance Counters.* ENDGAME Online Website (Our Blog). https://www.endgame.com/blog/technical-blog/detecting-spectre-and-meltdown-using-hardware-performance-counters

[10] S. Hong & Y. J. Seo. (2016). Countermeasure of Sniffing Attack: Survey. *Journal of Convergence Society for SMB, 6(2),* 31-36. DOI : 10.22156/CS4SMB.2016.6.2.031

[11] H. J. Mun, S. H. Choi & Y. C. Hwang. (2016). Effective Countermeasure to APT Attacks using Big Data. *Journal of Convergence Society for SMB, 6(1),* 17-23. DOI : 10.22156/CS4SMB.2016.6.1.017

[12] M. S. Gul & Y. Z. Li. (2015). A Study of Countermeasures for Advanced Persistent Threats attacks by malicious code. *Journal of Convergence Society for SMB, 5(4),* 37-42.

이 재 규(Lee, Jae Kyu) [정회원]

- 2018년 2월 : 한신대학교 컴퓨터공학부(학사)
- 2018년 9월 ~ 현재 : 한신대학교 산학협력단 소속 컴퓨터공학부 연구원
- 관심분야 : 보안, 머신러닝
- E-Mail : jk5309@naver.com

이 형 우(Lee, Hyung Woo) [정회원]

- 1994년 2월 : 고려대학교 컴퓨터학과(학사)
- 1997년 2월 : 고려대학교 컴퓨터학과(석사)
- 1999년 2월 : 고려대학교 컴퓨터학과(박사)
- 1999년 2월 ~ 2003년 2월 : 백석대학교 정보통신학부 조교수
- 2003년 3월 ~ 현재 : 한신대학교 컴퓨터학공학부 부교수, 정교수
- 관심분야 : 모바일/네트워크 보안, 디지털포렌식, 정보보호
- E-Mail : hwlee@hs.ac.kr