Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.29, NO.6, Dec. 2019 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2019.29.6.1365

바이너리 패턴 분석을 이용한 멜트다운, 스펙터 악성코드 탐지 방법*

김 문 선,† 이 만 희‡
한남대학교

Detecting Meltdown and Spectre Malware through Binary Pattern Analysis*

Moon-sun Kim,† Man-hee Lee‡
Hannam University

요 약

Meltdown과 Spectre는 프로세서의 비순차 및 추측 실행의 취약점을 이용해 일반 사용자 권한으로 접근할 수 없는 메모리를 읽는 공격이다. 이 공격을 방지하기 위한 대응 패치가 공개되었으나, 적용 가능한 패치가 없는 오래된 시스템 등은 여전히 이 공격에 취약하다고 할 수 있다. 이 공격을 탐지하기 위한 연구가 이루어지고 있지만 대부분 동적 식별 방법을 제안하고 있다. 따라서 본 논문은 Meltdown과 Spectre 악성코드를 실행하지 않고 파일 상태에서 탐지가 가능한 시그니처를 제안한다. 이를 위해 GitHub에 등록된 13종의 악성코드에 대한 바이너리 패턴 분석을 수행하였다. 이를 바탕으로 공격 파일 식별 방법을 제안하였으며, 실험결과 분석한 악성코드와 현재 악성코드 데이터베이스에 등록된 19개의 변종 악성코드를 100% 식별했고, 2,317개의 정상파일 중 0.94%(22건)의 오탐률을 보였다.

ABSTRACT

Meltdown and Spectre are vulnerabilities that exploit out-of-order execution and speculative execution techniques to read memory regions that are not accessible with user privileges. OS patches were released to prevent this attack, but older systems without appropriate patches are still vulnerable. Currently, there are some research to detect Meltdown and Spectre attacks, but most of them proposed dynamic analysis methods. Therefore, this paper proposes a binary signature that can be used to detect Meltdown and Spectre malware without executing them. For this, we collected 13 malicious codes from GitHub and performed binary pattern analysis. Based on this, we proposed a static detection method for Meltdown and Spectre malware. Our results showed that the method identified all the 19 attack files with 0.94% false positive rate when applied to 2,317 normal files.

Keywords: Meltdown, Spectre, Binary Pattern Analysis, Malware Detection

I. 서 론

Meltdown과 Spectre는 각각 프로세서의 파이프라인 최적화 기술인 비순차 실행(Out-of-Order Execution)과 추측 실행(Speculative Execution)의 취약점을 통해 접근 권한이 없는 메모리를 읽는 공격이다[1,2,3]. 일반적으로 사용자 프로세스는 다른 프로세스 또는 커널의 메모리를 접근할 수 없다. 하지만, 해당 취약점을 이용하면 운영체제의 메모리 보호 기능을 우회하여 커널 및 다른 프로세스의 메모리를 읽을 수 있다. 이 취약점은 비순차 실행과 추측 실행 기술을 사용하는 모든 프로세서에 영향을 줄 수 있기 때문에 세계적으로 큰 반향을 일으켰고, 지금도

Received(09. 23. 2019), Accepted(10. 23. 2019)
* 본 논문은 2019년도 한국정보보호학회 하계학술대회에 발표한 우수논문을 개선 및 확장한 것임
† 주저자, kmoonsun95@gmail.com
‡ 교신저자, manheelee@hnu.kr(Corresponding author)

많은 연구자들이 연구를 수행 중이다.

취약점 발표 직전, 각 프로세서 제조사 및 OS는 대응 패치를 배포했지만, 현재 사용되고 있는 모든 버전의 운영체제에 대한 패치가 제공된 것이 아니므로 적용 가능한 패치가 없는 시스템들은 여전히 이 공격에 취약하다[4,5]. 예를 들어, Microsoft 운영체제의 경우, Windows 10을 포함한 많이 사용되는 버전의 대응 패치를 제공하고 있으나 Windows XP, Vista, 8을 위한 패치는 제공하지 않는다. 비록 이 시스템들의 비중은 약 4%로써 점유율 자체는 높지 않지만, Windows 시스템의 사용 대수를 고려하면 이 공격은 여전히 위험한 수준이라고 할 수 있다[6,7]. 또한 일반적으로 보안 패치를 자동으로 적용하지 않는 리눅스 시스템을 고려하면 대응 패치로 이 공격을 모두 방지했다고 말하기 어려운 실정이다.

더욱이 기존의 대응 패치를 우회하는 새로운 공격 방법 및 취약점이 지속적으로 발표되고 있어[8], 이 공격에 대한 탐지 기법 연구가 절실한 실정이다. 하지만, 새로운 취약점을 찾거나 운영체제 및 컴퓨터 구조를 변경하여 이 공격을 방지하는 연구는 많으나, Meltdown과 Spectre 기반 악성코드 탐지 연구는 그리 많지 않다. 그마저도 프로세서가 제공하는 HPC(Hardware Performance Counter)를 이용한 동적 분석 기법 연구가 주를 이루고 있는 실정이다[9,10,11,12].

만약, 백신이 이 공격을 정적으로 탐지할 수 있다면 대응 패치를 지원하지 않는 시스템은 이 공격으로부터 안전하게 보호받을 수 있을 것이다. 하지만 GitHub와 악성코드 데이터베이스 및 분석 사이트인 Hybrid-Analysis 에서 32종의 Meltdown과 Spectre 악성코드를 수집한 뒤, VirusTotal을 이용하여 확인한 결과, 평균 31%의 탐지율을 보였으며, 특히 7개의 변종 코드는 전체 백신의 99% 이상이 전혀 탐지하지 못했다. 즉, 현재 가용한 백신으로부터 Meltdown과 Spectre 공격에 대한 안전을 확보하기 어렵다.

본 논문은 이 안전의 공백을 방지하기 위해 효과적인 정적 탐지 기법을 제안한다. 이를 위해 Meltdown과 Spectre 악성코드에 대해 바이너리 패턴 분석을 수행하였고, 해당 결과를 토대로 공격 파일을 식별할 수 있는 시그니처를 제안했다. 또한, 이 시그니처를 이용하여 자동으로 Meltdown과 Spectre 공격 파일을 탐지하는 도구를 개발하였다. 탐지 실험 결과, 변종을 포함한 악성코드 32종에 대해 100% 탐지율을 보였으며, 2,317개의 정상 실행 파일 중 0.94%(22건)의 오탐률을 확인하였다. 현재까지 파악한 바로는, Meltdown과 Spectre를 탐지하는 정적 분석 기법과 도구가 제안된 적은 없었으므로 그 의미가 있다고 할 수 있다.

본 논문은 다음과 같이 구성된다. 2장에서 Meltdown과 Spectre를 소개하고 3장은 이 공격들을 탐지하기 위해 연구된 방법들을 소개한다. 이후 4장에서 바이너리 분석을 통해 도출한 시그니처와 이를 적용한 Checker를 소개하며, 5장에서 실험 결과를 보이고 6장에서 결론과 향후 연구를 논한다.

II. 배경 지식

2.1 Meltdown

Meltdown은 프로세서의 비순차 실행을 악용하여 커널 메모리를 읽을 수 있는 취약점다. Intel의 모든 프로세서와 ARM Cortex-A75 기반 시스템이 이 취약점에 영향을 받는 것으로 알려졌다[1]. 비순차 실행은 프로세서의 파이프라인을 효과적으로 사용하기 위한 최적화 기술로서, 명령어의 순서에 관계없이 실행 가능한 명령어를 우선적으로 실행한다[3].

프로세서는 비순차적으로 실행된 명령어가 Commit되기 전에 권한 없는 메모리에 접근한 것을 확인하면 rollback interrupt를 통해 해당 명령어의 실행을 정지하고, 파이프라인을 초기화한다. 따라서 사용자는 해당 명령어의 최종 수행 결과를 알 수 없다. 하지만, 이 취약점에 가장 영향을 많이 받는 Intel 프로세서는 rollback interrupt를 Execute 단계 이후에 발생시킨다. 따라서 해당 명령어가 Execute될 때 참조한 메모리 주소가 캐시에 적재된다. 이 데이터는 일반적인 방법으로 읽지 못하지만, Flush+Reload 공격 수행하면 해당 데이터를 읽을 수 있다[13].

Flush+Reload 공격은 캐시 부채널 공격의 일종으로서 희생자 프로세스가 접근한 메모리를 캐시를 통해 간접적으로 읽는 공격이다. 커널 영역은 공유 메모리의 형태로 각 프로세스의 사용자 영역에 함께 매핑되어 있다. 따라서 공격자는 희생자가 접근한 메모리를 L3 캐시를 통해 간접적으로 접근할 수 있다. 여기서 L3 캐시는 Last-Level-Cache로서 프로세서의 모든 코어의 상위 캐시 정보를 담고 있다. Flush+Reload의 자세한 공격 과정은 다음과 같다.

1) 공격자는 clflush 명령어를 통해 희생자 프로세스가 참조하는 위치로 확인된 메모리 주소를 캐시에서 모두 제거한다.
2) 이후 공격자는 희생자 프로세스가 다시 메모리에 접근하는 것을 기다린다.
3) 희생자 프로세스가 다시 메모리에 접근하면 참조한 메모리 페이지 주소가 캐시에 적재된다. 이때 공격자는 각 페이지에 대한 접근 시간을 측정한다. 여기서 가장 빠른 접근 속도를 가지는 페이지 테이블 번호를 비밀 데이터로 확정한다.

캐시는 물리 메모리에 비해 프로세서의 접근 속도가 빠르므로 희생자 프로세스가 참조했던 메모리는 접근 속도가 빠르게 측정된다. 따라서 공격자는 희생자 프로세스가 접근했던 메모리를 간접적으로 유추할 수 있다.

Fig. 1.은 Meltdown 개념 증명 코드에 포함된 Flush+Reload 공격 코드 중 특정 주소에 대한 접근 시간을 측정하는 함수이다. Fig. 1.의 line 11~13은 프로세서의 TSC값을 반환하는 rdtscp 명령어를 호출한다. TSC는 프로세서에 내장된 고성능 레지스터로서, 프로세서의 사이클 수를 계산하는데 이용할 수 있다. line 12, 13은 주소 *addr*에 접근하고, 다시 TSC를 읽는 과정이다. 이후 time2 - time1(line 24)을 통해 프로세서가 *addr*에 접

```
1  cat <<-'EOF'
2      static inline int
3      get_access_time(volatile char *addr)
4      {
5          unsigned long long time1, time2;
6  EOF
7
8  if grep -q rdtscp /proc/cpuinfo; then
9      cat <<-'EOF'
10         unsigned junk;
11         time1 = __rdtscp(&junk);
12         (void)*addr;
13         time2 = __rdtscp(&junk);
14     EOF
15 else
16     cat <<-'EOF'
17         time1 = __rdtsc();
18         (void)*addr;
19         _mm_mfence();
20         time2 = __rdtsc();
21     EOF
22 fi
23 cat <<-'EOF'
24         return time2 - time1;
25     }
26 EOF
```

Fig. 1. Flush+Reload attack in Meltdown Exploit code

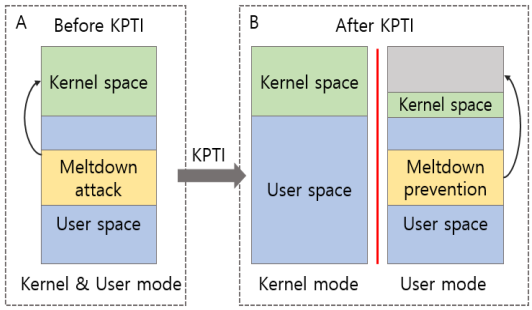


Fig. 2. KPTI(Kernel Page-Table Isolation)

근하는 동안 소요된 사이클을 계산한다.

Meltdown 공격이 커널 메모리를 읽을 수 있는 이유는 모든 프로세스에 할당되는 가상 메모리에 Fig. 2(A).와 같이 커널 영역과 사용자 영역이 함께 매핑된 형태로 페이지 테이블을 공유하기 때문이다. 각 영역은 운영체제의 권한 수준에 의해 격리되지만, Meltdown 공격은 비순차 실행의 취약점을 악용하여 격리를 우회할 수 있다[1].

2018년 1월, Linux는 Meltdown 취약점을 완화하기 위해 KPTI(Kernel Page-table Isolation) 기능을 포함한 패치를 배포하였다[4]. KPTI는 Fig. 2(B).과 같이 사용자 영역과 커널 영역의 페이지 테이블을 분리하여 Meltdown 공격으로부터 커널 메모리를 보호한다. 하지만 유저 모드에서 커널 모드로 진입하기 위해 커널 페이지 테이블을 로드하는 과정이 필요하고, 이 과정에서 최대 30%의 성능 저하를 유발한다[5].

2.2 Spectre variant 1

Spectre variant 1(Bounds check bypass)은 프로세서의 추측 실행을 악용하여, 할당된 배열의 메모리 경계를 벗어나는 영역에 접근하는 공격이다. Intel 프로세서에 국한된 Meltdown과 달리, Spectre는 현대의 모든 프로세서가 영향을 받으며 다양한 형태의 변형 공격이 존재한다[2,8].

추측 실행은 분기 예측기(Branch Predictor)[3]를 통해 프로세서가 필요할 것으로 예상되는 명령어를 추론하여 실행하는 기술이다. 하지만 Spectre 공격을 통해 분기 예측기가 공격자에 의해 제어될 수 있다는 것이 밝혀졌다[2].

이 공격은 분기 예측기를 공격자의 의도대로 학습하는 것으로 시작한다. 이후 공격자는 배열 경계를

```
void victim_function(size_t x)
{
    if (x < array1_size)
        temp &= array2[array[x] * 512];
}
```

Fig. 3. Part of Spectre proof of concept code

벗어난 주소에 접근하는 명령어를 추측 실행 한다. 프로세서는 추측 실행된 명령어가 올바르지 않은 경우 실행된 명령어를 폐기하지만, 명령어가 참조했던 메모리 주소는 캐시에 남게 된다. 따라서 공격자는 2.1의 내용과 같이 Flush+Reload 공격을 통해 커널 및 다른 프로세스 메모리 정보를 읽을 수 있다. Fig. 3.은 Spectre 개념 증명 코드의 핵심 부분 중 일부이며 공격 과정은 다음과 같다.

1) 매개변수 *x*에 *array1_size*보다 작은 값을 5번 전달하여 분기 예측기가 해당 조건 분기를 true로 추측하도록 학습한다.
2) 6번째 횟수에서 매개변수 *x*에 *array1_size*보다 큰 임의의 주소를 전달한다. 분기 예측기는 (1)에서 5번의 true를 통해 경계 검사를 참으로 예측하도록 학습된 상태이다. 따라서 해당 조건 분기를 경계 검사 없이 추측 실행 한다.
3) 프로세서는 추측 실행한 주소에 대해 접근 권한이 없는 것을 확인하고(false) 해당 분기의 실행 결과를 폐기한다. 하지만 추측 실행 도중 참조했던 메모리 정보가 캐시에 남는다.
4) 공격자는 Flush+Reload 공격을 통해 캐시에 저장된 값 *x*를 유출한다.

Intel은 Spectre를 완화하기 위해 lfence 명령어를 사용하도록 권장하고 있다. lfence 명령어는 경계 검사가 완료되기 전까지 다음 명령어가 로드될 수는 있지만 실행되지 않도록 하여 추측 실행을 막는 명령어이다[14]. Linux 커널은 Spectre 공격으로부터 취약한 부분에 lfence 명령어를 삽입하여 이 공격에 대응하였다. 하지만 lfence 명령어는 추측 실행을 제한하기 때문에 성능 저하가 발생한다[15].

III. 관련 연구

HPC는 프로세서에 장착된 특수 목적 레지스터이다. 성능 저하를 최소화한 상태에서 프로세서의 클록 주기, 캐시 Hit/Miss와 같은 이벤트를 측정할 때 사용할 수 있다. Congmiao Li 등은 HPC를 통한 멀웨어 탐지가 가능하다는 점에 착안하여, 일반 시스템과 Spectre 공격을 받는 시스템의 마이크로아키텍처를 분석하고, 머신 러닝을 통한 Spectre 공격 동적 탐지기를 제작하였다[9,10]. 이 방법을 통해 Spectre 공격을 100% 동적 탐지하고, 0.77%의 오탐률을 가진다고 밝혔다.

Jonas Depoix 등은 HPC를 통해 수집한 캐시 Hit/Miss 데이터를 통해 Spectre 공격이 수행하는 Flush+Reload의 패턴을 인공 신경망 모델에 적용했다[11]. 이 모델을 통해 시스템을 모니터링 하여 Spectre 공격을 탐지하는 방법을 제시했다.

Meltdown 공격은 권한이 없는 메모리 영역에 접근을 시도하기 때문에 프로세스 간 통신 메시지로 사용되는 SIGSEGV Signal이 발생한다. Jae-Kyu Lee 등은 Meltdown 공격의 SIGSEGV Signal을 분석하여 커널 메모리 접근 여부, SIGSEGV signal의 주기적 발생 여부 등의 데이터를 수집하였다[12]. 이후 해당 데이터를 바탕으로 공격 여부를 실시간으로 동적 탐지하는 의사 결정 트리 모델을 제안했다.

Meltdown과 Spectre 공격 탐지는 대부분 프로세스를 감시하고 의심되는 동작을 탐지하기 때문에 공격이 실제 실행되어야 한다는 단점이 있다. 완화 방법 또한 아키텍처의 구조적 변경 혹은 성능 저하를 동반하는 패치를 필요로 한다. 하지만, 공격 코드가 실행되기 이전에 탐지할 수 있는 정적 탐지 방법에 관한 연구는 미미한 실정이다.

IV. 시그니처 분석

우리는 이전 연구에서 Meltdown 공격 파일을 정적으로 식별하기 위해 악성코드 정적 분석 방법인 String 검색, Opcode 분석, API 호출 분석을 통한 Meltdown과 Spectre 공격 파일 식별을 시도했다. 하지만 String 검색을 수행한 결과 여러 변종에서 공통적으로 나타난 String은 없었다. 이후 opcode와 API 호출 분석을 수행한 결과 clflush, rdtsc, GetLastError 등 일부 공통적으로 사용되는 opcode와 API 호출을 발견했다.

해당 요소들의 단순 포함 여부를 시그니처 정보로 적용해 본 결과 분석에 사용한 11개의 악성코드 탐지율은 100%였으며, 1,700개의 일반 파일에 대한

오탐률은 0.35%로 나타났다[16]. 하지만, Hybrid-Analysis에서 새롭게 수집한 변종 악성코드 19종에 대한 탐지 비율은 46.15%에 머물렀다. Meltdown과 Spectre 공격 코드는 독립적으로 작동하지 않고 타 악성코드에 일부 기능으로 삽입되는 등 지속적인 변형이 이루어지고 있어 식별이 더 어려워진 것으로 파악된다.

따라서 우리는 기존 연구에서 도출한 시그니처의 변종 탐지 능력을 강화하기 위해 공격 파일 13종에 대해 추가적으로 opcode분석을 수행하였다. 또한 Meltdown과 Spectre 공격 파일과 일반 파일은 clflush와 같은 특정 opcode에 대한 빈도수가 다를 것으로 가정하고, 이를 검증하기 위해 opcode의 호출 횟수와 빈도수 분석을 수행하였다. 이 과정에서 변종에 대한 탐지 능력을 검증하기 위해 새로 수집한 변종 공격 코드는 정적 분석을 수행하지 않았다.

4.1 시그니처 제안

시그니처 분석은 GitHub에 공개된 Meltdown과 Spectre variant 1 공격 파일 13종을 통해 진행했다. Hybrid-Analysis에서 수집 가능한 19개의 변종코드는 시그니처의 작동 여부 검사에 사용하였다. 공격 파일에 대한 정적 분석 결과, 5가지 opcode와 2가지 API 호출을 시그니처로 규정하였다. 각 시그니처를 선정한 이유와 공격 코드 내부에서 수행하는 역할은 다음과 같다.

- rdtsc

rdtsc는 프로세서의 TSC 값을 반환하는 명령어이다[14]. Flush+Reload 공격은 rdtsc를 통해 획득한 프로세서 TSC를 통해 메모리에 대한 접근 시간을 측정한다. 따라서 해당 opcode를 시그니처로 사용하였다.

- rdtscp

rdtscp는 rdtsc와 마찬가지로 프로세서의 TSC를 반환한다[14]. 두 명령어의 차이점은 명령어 직렬화의 여부이다. 직렬화 명령어는 반드시 순차적으로 실행되는 명령어를 의미한다. rdtscp는 직렬화 명령이기 때문에 병렬 실행되지 않지만 일부 구형 시스템이 지원하지 않는 제약점이 있다. 따라서 Meltdown과 Spectre 공격 코드는 시스템 정보를 조회하고 rdtscp의 지원 여부를 확인하는 작업을 선행한다. 이후 해당 시스템이 rdtscp를 지원하지 않는 시스템으로 파악되면 다른 공격 함수에 정의된 rdtsc 공격 코드를 실행한다.

- mfence

mfence는 메모리 로드 및 저장 명령의 병렬수행을 막는 명령어이다[14]. 공격 코드는 시스템이 rdtscp 명령어를 지원하지 않는 경우 rdtsc를 사용한다. 이때 rdtsc 명령어는 직렬화 명령어가 아니기 때문에 TSC에 오차가 발생할 수 있다. 따라서 공격 코드는 rdtsc 명령어 직전에 mfence 명령어를 삽입하여 rdtsc의 병렬 수행을 방지한다. 따라서 rdtsc가 식별되는 경우, mfence를 반드시 포함하는 공격 코드의 특징을 시그니처 정보로 추가하였다.

- clflush

clflush 명령어는 매개 변수로 넘겨진 주소를 포함하는 캐시 라인을 제거하는 명령어이다[14]. 이 명령어는 Flush+Reload 공격에서 희생자 프로세스가 사용한 메모리 주소를 캐시에 제거하는 과정에서 사용된다. Meltdown과 Spectre는 공통적으로 Flush+Reload 공격을 통해 캐시로 부터 데이터를 유출한다. 따라서 clflush는 이 공격 파일을 식별할 수 있는 핵심 시그니처 정보로 사용될 수 있다.

- cpuid

cpuid는 프로세서의 모델, 제조사 등의 정보를 반환하는 명령어이다[14]. Meltdown과 Spectre 공격은 cpuid 호출을 통해 작동하는 시스템 정보를 파악한 뒤, 일부 명령어를 바꾸는 등 서로 다른 공격 패턴을 사용한다. 따라서 해당 opcode는 한 번 이상 호출되는 특징을 보인다. 따라서 cpuid의 포함 여부를 시그니처 정보로 사용하였다.

- SetUnhandledExceptionFilter

SetUnhandledExceptionFilter API는 프로세스에서 예기치 못한 예외가 발생했을 때, 별도의 예외 처리 핸들러가 지정되지 않은 경우 호출된다[17]. Meltdown과 Spectre는 공통적으로 접근할 수 없는 메모리에 접근하는 특징을 가지고 있다. 따라서 반드시 예외가 발생하며, 프로세스는 해당 API를 호출한다. 커널 메모리에 집적 접근하는 Meltdown 공격 파일은 별도의 예외 핸들링 함수를 통해 프로세스가 종료되지 않도록 한다.

Table 1. The number of signatures contained in the Meltdown, Spectre attack file with the highest antivirus engine detection rate.

Instruction and API	Melt.1	Melt.2	Melt.3	Spect.1	Spect.2	Spect.3
rdtsc	3	2	0	6	0	0
rdtscp	0	0	2	2	2	2
mfence	3	3	0	0	0	0
clflush	1	1	2	2	2	2
cpuid	12	5	10	5	4	7
SetUnhandledExceptionFilter	1	1	1	5	1	1
QueryPerformanceCounter	1	1	1	4	1	1

- QueryPerformanceCounter

QueryPerformanceCounter는 프로세서의 TSC를 반환하는 API로써 보통 코드의 동작 시간, 프레임 측정과 같은 고성능 타이머를 요구하는 작업에 사용된다[17]. 이 API는 Windows 환경에서 작동하는 프로그램이면 rdtsc와 rdtscp 명령어를 실행할 때 해당 API를 호출한다.

Table 1.은 Meltdown과 Spectre 악성코드가 시그니처 정보를 얼마나 포함하는지를 보여준다. 가로축은 분석한 Meltdown과 Spectre 악성코드 중 백신의 탐지율이 높은 상위 3종을 선별한 것이다. 세로축은 시그니처들이 각 공격 파일의 바이너리에 몇 번 포함되는지를 나타낸다. 시스템에 따라 다르게 사용되는 rdtsc, rdtscp, mfence를 제외하면 모든 시그니처가 반드시 한 번 이상 포함되는 것을 확인할 수 있다. 따라서 7가지 시그니처를 모두 포함한 파일은 Meltdown과 Spectre 악성코드일 가능성이 있다는 것을 의미한다.

또한, 이 공격 코드는 Flush+Reload 공격을 수행하기 때문에 2번 이상의 TSC 접근이 이루어 질 것으로 가정하고 분석하였다. 그 결과 Table 1.과 같이 rdtsc와 rdtscp의 횟수가 2회 이상인 것으로 나타났다. 따라서 해당 명령어가 2회 이상 포함된다면 공격 파일로 식별하는 시그니처를 추가하였다.

시그니처의 정확도를 확인하기 위해 얼마나 많은 일반 실행파일이 이 시그니처를 포함하는지 살펴보았다. 그 결과 2,317개중 2.37%(55개)의 일반 실행파일이 7가지 시그니처를 모두 가진 것으로 조사되었다. 이 오탐률은 실제 적용되기 어려운 것으로 판단하여 오탐률 최소화하기 위해 clflush 빈도수 분석을 추가적으로 수행하였다.

Meltdown과 Spectre는 공통적으로 Flush+Reload 공격을 수행하여 캐시에 적재된 희생자의 메모리를 읽는다. 따라서 공격 파일은 Flush+Reload 공격에 사용되는 핵심 opcode인 clflush, rdtsc(p), mfence의 비율이 일반 파일에 비해 높을 것으로 가정하고 실험을 진행하였다.

실험 결과, 공격 파일들의 평균 clflush 빈도수는 0.02% 이상인 반면, 일반 파일은 0.001%로 20배 이상 차이가 나는 것으로 나타났다. 따라서 clflush 빈도수를 통해 파일의 악성 여부를 가늠할 수 있을 것으로 판단된다. Table 2.는 분석에 사용한 13종의 공격 파일과 2,317개의 일반 파일(clean)에 대한 opcode 빈도수 분석 결과이다.

Fig. 4.는 분석한 Meltdown과 Spectre 공격

Table 2. Opcode frequency analysis results.

Opcode	Meltdown (%)	Spectre (%)	Clean (%)
clflush	**0.02**	**0.024**	**0.001**
rdtscp	0.008	0.019	0.000
rdtsc	0.015	0.021	0.039
mfence	0.046	0.011	0.000
test	19.261	14.734	20.295
xor	12.908	9.384	13.885
cmp	7.325	5.607	10.591
jz	43.873	46.054	37.352
jnz	16.539	24.140	17.833

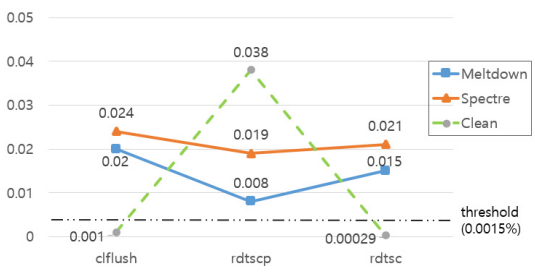


Fig. 4. Threshold of clflush frequency

파일이 포함하는 3가지 opcode의 빈도수 평균을 도식화한 것이다. 공격 파일의 clflush 빈도수는 평균 0.02% 이상으로 나타났지만, 가장 낮은 빈도수를 가지는 개체는 빈도수가 0.0015%로 나타났다. 본 시그니처의 목적은 모든 변종 코드를 탐지하는 것이기 때문에 clflush 빈도수가 0.0015% 미만인 파일은 일반 파일로 분류하는 시그니처를 추가하였다.

4.2 Meltdown과 Spectre 공격 파일 Checker

4.1에 소개한 시그니처를 바탕으로 Windows에서 Meltdown과 Spectre 공격 파일을 실행하지 않고 식별할 수 있는 Checker를 개발하였다. Checker는 7단계로 공격 파일을 식별한다. 각 단계는 시그니처 기반 조건 검사를 통해 일반 파일과 악성 파일을 분류한다. 모든 검사를 거쳐도 악성 여부가 분류되지 않는 경우가 있을 수 있다. 이때 악성 여부가 식별되지 않은 파일을 일반 파일로 분류한다면, 시그니처를 우회하는 변종 공격 파일을 식별하지 못할 가능성이 있다. 따라서 모든 단계의 검사를 거쳤는데도 불구하고 악성 유무가 식별되지 않으면 공격 파일로 확정한다. 공격 파일을 탐지하는 각 단계는 아래와 같으며 Fig. 5.는 이 과정을 순서도로 나타낸 것이다. 순서도 내부의 모든 변수는 해당 변수(opcode)의 개수를 의미한다.

1) 공격 파일에서 rdtscp가 사용된 경우 mfence는 사용되지 않는 특징을 보인다. 반대로 rdtsc가 사용된 경우, 반드시 mfence가 사용된다. 따라서 rdtscp와 mfence의 개수가 같으면 일반 파일로 분류한다.
2) clflush의 수가 1개 이상이면 다음 단계로 이동한다. clflush가 없다면 일반 파일로 분류한다.
3) clflush 빈도수가 0.0015% 미만이면 일반파일로 분류한다. 그 이상이면 다음 단계로 이동한다.
4) rdtsc(p)의 개수가 2개 이상이면 앞선 단계에서 식별한 opcode와의 조합이 Flush+Reload 공격 패턴과 일치한다. 따라서 공격 파일로 확정한다. 2개 미만이면 다음단계로 이동한다.
5) cpuid가 없으면 일반 파일로 분류한다.
6) SetUnhandledExceptionFilter 또는 QueryPerformanceCounter가 없으면 일반 파일로 분류한다.
7) 분류되지 않은 파일은 공격 파일로 확정한다.

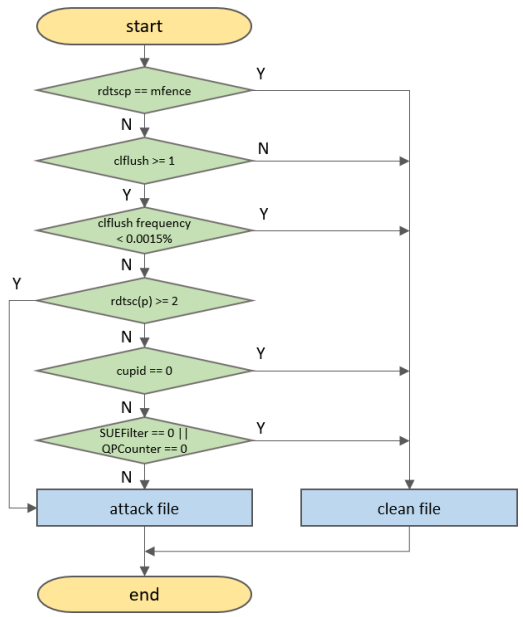


Fig. 5. Attack file identification process

V. 시그니처 성능 실험

실험은 수집한 악성코드와 정상 파일을 Checker로 분석하는 방법으로 진행했다. 실험에 사용한 파일은 Hybrid-Analysis와 GitHub에서 수집한 32종의 Meltdown과 Spectre 악성코드이며, KakaoTalk.exe, notepad++.exe와 같이 일반적인 Windows 환경에서 사용하는 2,317개의 정상 실행 파일이다.

수집한 악성코드를 VirusTotal의 백신 엔진에서 분석할 결과, 평균 탐지율은 31%이며, 이 중 7종은 99%의 백신이 탐지하지 못했다. 이에 반해, 본 연구에서 제안한 시그니처를 이용하는 Checker는 Meltdown과 Spectre 악성코드 32종을 100% 탐지했으며 정상 파일에 대한 오탐률은 0.94%로 나타났다(Table 3.)

Table 3. Checker performance valuation result

False Alarm	Detection / Set	Rate(%)
False Positive	22 / 2317	0.94
False Negative	0 / 32	0
True Positive	32 / 32	100

VI. 결 론

본 논문은 Meltdown과 Spectre 공격 파일을 정적으로 탐지하기 위한 바이너리 패턴을 제안하고, 이를 바탕으로 구현한 자동화된 분석 도구인 Checker를 소개하였다. Checker의 성능실험 32종의 Meltdown과 Spectre 악성코드 100% 식별했으며, 오탐률은 0.94%였다. 현재 해당 악성코드들은 평균 31%의 백신만 탐지하기 때문에 완화 패치를 적용하기 어려운 시스템에서 기존의 백신과 제안한 Checker를 함께 운용하면 Meltdown과 Spectre 공격을 더욱 효과적으로 예방할 수 있을 것으로 기대한다. 향후 연구로 docker와 VMware 같은 가상화 이미지 파일에 포함된 Meltdown과 Spectre 공격 파일을 정적 탐지할 수 있는 Checker를 개발 중에 있다.

References

[1] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Hass, S. Mangard, P. Kocher, D. Genkin, Y. Yarom and M. Hamburg, "Meltdown: Reading Kernel Memory from User Space," the 27th USENIX Security Symposium, pp. 973-990, Aug. 2018.

[2] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," 2019 IEEE Symposium on Security and Privacy, pp. 1-19, May. 2019.

[3] A. Fog, "The Microarchitecture of Intel, AMD and VIA CPUs: An optimization guide for assembly programmers and compiler makers" https://www.agner.org/optimize/microarchitecture.pdf, Sep. 2018.

[4] M. Löw, "Overview of Meltdown and Spectre patches and their impacts," WAMOS 2018, pp. 53-61, Jul. 2018.

[5] J. Corbet, "Kaiser: hiding the kernel from user space" LWN.net, https://lwn.net/Articles/738975/, Nov. 2017.

[6] Microsoft, "Protect your Windows devices against speculative execution side-channel attacks" https://support.microsoft.com/en-us/help/4073757/protect-windows-devices-from-speculative-execution-side-channel-attack/, Sep. 2019.

[7] NetMarketShare, "market share report" https://netmarketshare.com/, Aug. 2019.

[8] C. Canella, J. Van Bulck, M. Schwarz, M. Lipp, B. von Berg, P. Ortner, F. Piessens, D. Evtyushkin and D. Gruss, "A systematic evaluation of transient execution attacks and defenses," the 28th USENIX Security Symposium, pp. 249-266, Aug. 2019.

[9] C. LI and JL. Gaudiot, "Online Detection of Spectre Attacks Using Microarchitectural Traces from Performance Counters," 30th IEEE International Symposium on Computer Architecture and High Performance Computing, pp. 25-28, Sep. 2018.

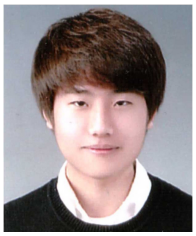
[10] J. Demme, M. Maycock, J. Schmitz, A. Tang, A. Waksman, S. Sethumadhavan and S. Stolfo, "On the feasibility of online malware detection with performance counters," ACM SIGARCH Computer Architecture News, 41(3), pp. 559-570, Jun. 2013.

[11] J. Depoix and P. Altmeyer, "Detecting Spectre Attacks by identifying Cache Side-Channel Attacks using Machine Learning," WAMOS 2018, pp. 75-85, Jul. 2018.

[12] Lee Jaekyu and Lee Hyungwoo, "Meltdown Threat Dynamic Detection Mechanism using Decision-Tree based Machine Learning Method," Journal of

Convergence for Information Technology, pp. 209-215, Dec. 2018.
〔13〕 Y. Yarom and K. Falkner, "FLUSH+RELOAD: a high resolution, low noise, L3 cache side-channel attack," 23rd USENIX Security Symposium, pp. 719-732, Aug. 2014.
〔14〕 Intel, "Intel® 64 and IA-32 Architectures Software Developer's Manual" https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-instruction-set-reference-manual-325383.pdf, Sep. 2016.
〔15〕 J. Corbet, "Meltdown/Spectre mitigation for 4.15 and beyond" LWN.net https://lwn.net/Articles/744287/, Jan. 2018.
〔16〕 Kim Moonsun and Lee Manhee, "Meltdown Attack Identification Using Binary Pattern Analysis," CISC-W'19, pp. 374-377, Jun. 2019.
〔17〕 Microsoft Docs, "Programming reference for Windows API" https://docs.microsoft.com/en-us/windows/win32/api/, Sep. 2019.

〈저자소개〉



김 문 선 (Moon-sun Kim) 학생회원
2014년 3월~현재: 한남대학교 컴퓨터통신무인기술학과 학부과정
〈관심분야〉 시스템 보안, 역공학, 취약점 분석, 악성코드 탐지

이 만 희 (Man-hee Lee) 종신회원
1995년 2월 경북대학교 컴퓨터공학과 공학사
1997년 2월 경북대학교 공학석사
2008년 8월 Texas A&M 대학교 컴퓨터공학과 공학박사
1997년~2003년 한국과학기술정보연구원 연구원
2008년~2009년 Cisco Systems, San Jose
2010년~2012년 국가보안기술연구소 선임연구원
2012년~현재 한남대학교 부교수
〈관심분야〉 네트워크/시스템/스마트폰 보안, 고성능 시스템, 컴퓨터교육