정 보 보 호 학 회 지
제 30 권 제1호, 2020. 2

# 캐시 부채널 공격을 이용한 부채널 공격 동향

윤 한 재*, 이 만 희*

요 약

2018년 1월, Meltdown, Spectre와 같은 마이크로아키텍처의 취약점을 이용하는 부채널 공격이 등장하면서 전 세계적으로 부채널 공격에 관한 관심이 증가하였다. 또한, 소모 전력 분석, 전파 분석 등 전통적 부채널 공격과는 달리 캐시의 상태 변화를 이용하는 공격인 캐시 부채널 공격이 Meltdown, Spectre에 이용되면서 이에 관한 다양한 연구가 진행되고 있다.

이러한 유형의 공격은 완벽하게 방어할 수 있는 대응 패치가 존재하지 않고 일부 공격에 대응할 수 있는 대응 패치도 모든 시스템에 적용할 수 없은 경우가 많으므로 완벽한 대응이 매우 힘든 실정이다. 특히 캐시 부채널 공격을 이용하여 SGX와 같은 TEE(Trusted Execution Environment)를 공격할 수 있다는 것이 드러나면서 TEE를 공격하기 위한 다양한 공격 도구로 이용되고 있다. 본 논문에서는 Meltdown과 Spectre 및 다양한 캐시 부채널 공격에 대한 동향을 살펴보고자 한다.

## Ⅰ. 서 론

Meltdown, Spectre와 같이 마이크로아키텍처의 구현상의 취약점을 이용한 부채널 공격이 등장하면서 마이크로아키텍처에서 이루어지는 부채널 공격에 관한 관심이 세계적으로 증가하였다. 또한, 기존의 소모 전력 분석, 전파 분석 등과 같은 공격과는 달리 캐시의 상태 변화를 관찰하여 공격에 이용하는 캐시 부채널 공격이 Meltdown, Spectre 공격에 사용되면서 이에 관한 다양한 연구가 진행되고 있다[1][2].

특히 이러한 유형의 공격은 완벽하게 방어할 수 있는 대응 패치가 존재하지 않고 일부 대응 패치도 모든 시스템에 적용할 수 없기 때문에 다양한 공격 측면이 존재한다. 또한, 캐시 부채널 공격을 이용하여 Intel SGX와 같은 TEE 플랫폼이나 이미 대응 패치가 완료된 시스템을 공격할 수 있다는 것이 드러났다[3][4][5].

이렇듯 캐시 부채널 공격은 대응도 어렵고 다양한 환경을 공격할 수 있기 때문에 캐시 부채널 공격에 관한 연구가 활발히 이루어지고 있다. 본 논문에서는 캐시 부채널 공격을 이용하는 가장 기본적인 공격인 Meltdown과 Spectre에 대해 소개하고 이후로 진행된 다양한 유형의 캐시 부채널 공격에 대한 동향을 살펴보고자 한다.

## Ⅱ. 관련 연구

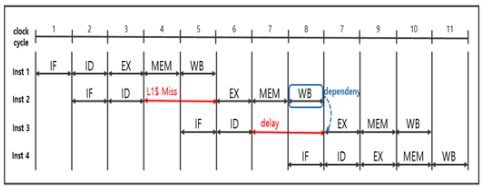
### 2.1. Out-of-Order execution

현대의 모든 프로세서는 명령어 파이프 라이닝(instruction pipelining)을 이용해 여러 개의 instruction을 동시에 병렬 수행한다. 특히 Out-of-Order execution 기술은 pipeline의 대기 상태가 길어져 instruction을 수행하는 데 소모되는 cycle의 수가 증가해 성능저하로 이어지는 현상을 해결하기 위해 고안되었다.

Out-of-Order execution의 핵심은 서로 데이터 의존성이 없는 instruction이 불필요하게 이전 instruction이 완료될 때까지 기다릴 때 pipeline의 대기 상태가 길어져 성능이 저하되기 때문에 서로 데이터 의존성이 없는 여러 개의 instruction이 pipeline으로 들어오면 실행 순서를 바꿔서 실행하는 것이다.

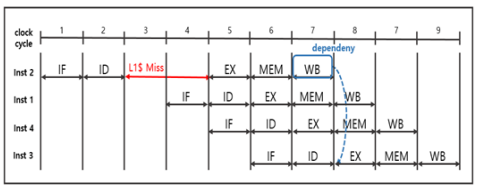
[그림 1]과 같이 서로 데이터 의존성이 없는 instruction 4가 instruction 3이 완료될 때까지 대기 시간이 길어져 4개의 instruction을 처리하는데 총 11 cycle이 소모된다.

이 성과는 2019년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(NRF-2018R1A4A1025632).
* 한남대학교 컴퓨터공학과 (대학원생, hanjae.karoha@gmail.com, 교수, manheelee@hnu.kr)




〔그림 1〕 In-Order execution.

하지만 [그림 2]와 같이 instruction 2를 처리하는 동안 서로 의존성이 없는 instruction 1, 4를 먼저 수행한 후 instruction 3을 수행하면 pipeline의 대기 시간을 최소화되어 4개의 명령을 처리하는데 총 9 cycle이 소모되기 때문에 성능을 향상시킬 수 있다.



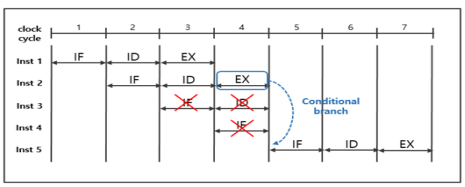

〔그림 2〕 Out-of-Order execution.

## 2.2. Branch prediction & Speculative execution

pipeline의 또 다른 성능저하 요인 중 하나는 조건 분기이다. 조건 분기 대상은 명령어 수행 단계 중 execute 단계가 완료된 후에 확인할 수 있기 때문에 조건 분기에 이어지는 instruction은 반드시 조건 분기가 끝날 때까지 기다려야 하며 그로 인한 성능저하가 불가피하다.

[그림 3]과 같이 instruction 2가 instruction 5로 분기할 경우 2 cycle 동안 수행되었던 instruction 3, 4는 폐기되어 버려진다. 프로세서는 이러한 폐기를 최소화하기 위해 Branch predictor를 이용하여 조건 분기의 결과를 예측한다.

분기하는 instruction이 인출되면 Branch predictor는 BTB(Branch Target Buffer)를 조회하여 이전에 분기




〔그림 3〕 pipeline에서의 조건 분기.

기록이 있는지 조회한다. 조회 결과가 없다면 새로 분기한 결과를 기록하고 기존의 방식대로 instruction을 처리한다.

조회 결과가 있다면 프로세서는 이전에 분기했던 주소를 참조하여 다음 instruction을 예측 실행한다. 예측 실행(Speculative execution)이란 Branch prediction에 기반을 두어 앞으로 실행될 것으로 추측되는 instruction을 미리 실행하는 기술이다. 만약 Branch prediction이 맞으면 미리 수행한 명령을 진행하지만, prediction이 틀릴 경우, 예측 실행한 instruction을 폐기하기 때문에 Branch predictor의 예측 정확도가 높을수록 프로세서의 전체적인 성능이 향상된다.

## 2.3. 캐시 부채널 공격

캐시 부채널 공격은 부채널 공격의 한 종류로 기존의 부채널 공격에서는 소모 전력, 전자파 등을 분석해 CPU의 연산을 추적한 후 데이터를 유출하였다면 캐시 부채널 공격에서는 캐시의 상태 변화를 모니터링하여 데이터를 유출한다[6].

캐시 부채널 공격의 대표적인 예는 Flush + Reload 공격이 있다. Flush + Reload 공격은 캐시에 적재된 데이터와 그렇지 않은 데이터에 접근할 때 접근 속도에 차이가 있다는 점을 이용한 공격 방법이다. 희생자가 접근한 데이터는 이미 캐시에 적재되어 있어 공격자가 다시 접근할 때 접근 속도가 매우 빠르지만, 희생자가 접근하지 않았던 데이터라면 접근 속도가 전자의 경우보다 훨씬 느리기 때문에 공격자는 희생자의 데이터 접근 여부를 확인할 수 있다.

Flush + Reload 공격을 위해 공격자는 LLC(Last Level Cache)를 모든 코어에서 공유하고 있다는 점을 이용한다. LLC는 모든 코어에서 공유하고 있기 때문에 캐시의 일관성을 유지하기 위해 LLC에서 데이터가 제거되면 다른 모든 코어에서도 데이터가 제거된다.

이러한 특성을 악용하여 공격자는 LLC의 모든 Cache Line을 *clflush()* 명령을 이용하여 제거한다. 이 상태에서 희생자가 어떤 메모리에 접근하면 희생자가 접근한 메모리 주소만 유일하게 캐시에 적재된다. 공격자는 일반적인 방법으로는 캐시에 적재된 데이터를 유출할 수 없기 때문에 메모리 접근 속도를 이용한다.

공격자는 데이터를 유출하기 위해 희생자가 접근한

메모리 주소에 접근한다. 공격자가 접근한 메모리 주소가 희생자가 접근했던 메모리 주소라면 캐시에 적재되어 있기 때문에 접근 속도가 매우 빠르지만, 희생자가 접근하지 않았던 메모리 주소라면 접근 속도가 전자보다 상대적으로 느리다. 공격자는 유출하려는 데이터를 메모리 주소의 index로 이용하여 유출하려는 데이터가 캐시에 적재되도록 만든 후 메모리 주소의 index에 다시 접근하여 데이터를 유출한다.

## Ⅲ. Meltdown & Spectre

### 3.1. Meltdown

Meltdown은 Out-of-Order execution과 Flush + Reload 공격을 이용하여 일반적인 프로세스는 접근할 수 없는 영역인 커널 메모리의 데이터를 유출할 수 있는 공격이다. 별도의 대응 패치가 적용되지 않은 32bit 시스템에서 각 프로세스에 할당되는 가상 메모리의 크기는 4GB이다. 4GB 중 3GB는 일반 프로세스가 사용할 수 있는 주소가 매핑되고 나머지 1GB에는 커널 영역의 주소가 매핑된다. 일반적인 상황에서 사용자는 커널 영역에 접근할 수 없지만, Meltdown 공격을 이용하여 커널 영역의 데이터를 유출할 수 있다[1].

```
1 raise_exception();
2 // the line below is never reached
3 access(probe_array[data * 4096]);
```

〔그림 4〕 Meltdown 예시〔1〕.

[그림 4]는 Meltdown 공격의 예제이다. 예외를 발생시키는 *raise_exception()* 함수가 실행되면 결과적으로 프로세스는 종료되고 3번째 줄은 실행되지 않는다. 하지만 1번째 줄과 3번째 줄 사이에 서로 데이터 의존성이 없기 때문에 프로세서는 Out-of-Order execution을 수행해 3번째 줄이 간접적으로 실행되고 *probe_array[data * 4096]*의 결과가 캐시에 적재된다. 이후 공격자는 Flush + Reload 공격을 이용하여 캐시에 적재된 데이터를 유출한다.

### 3.2. Spectre

Spectre는 Branch prediction과 Speculative execution을 이용해 일반적인 프로세스는 접근할 수 없는 영역의 데이터를 유출하는 공격이다. 공격자는 희생자 프로세스에서 분기 명령을 수행할 때 분기할 목적지를 잘못 예측하게 유도하고 잘못 예측된 데이터가 Speculative execution에 의해 캐시에 적재되면 Flush + Reload 공격을 이용하여 데이터를 유출한다[2].

```
if (x < array1_size)
    y = array2[array1[x] * 4096];
```

〔그림 5〕 분기 조건의 예시〔2〕.

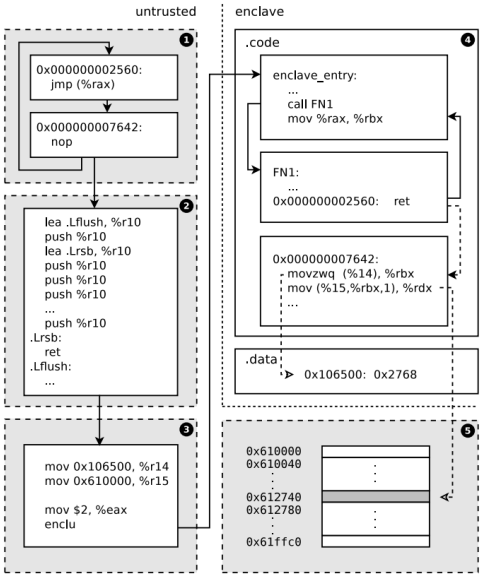
[그림 5]는 Spectre 공격에 사용되는 분기 조건의 예시이다. 공격자는 [그림 5]에서 분기 조건을 참(True)으로 만족하는 x를 이용하여 Branch predictor가 분기 결과를 항상 참으로 예측하도록 학습시킨다. 충분한 학습 이후 공격자는 실제 array1_size보다 큰 x를 대입한다. 결과적으로 프로세스는 예외를 발생시키며 종료되지만, Branch prediction과 Speculative execution에 의해 array2[array1[x] * 4096]의 결과가 캐시에 적재되고 공격자는 Flush + Reload 공격을 이용하여 캐시에 적재된 데이터를 유출한다.

## Ⅳ. 동 향

### 4.1. SGXPECTRE

SGXPECTRE는 Branch prediction과 Speculative execution을 이용하여 Intel의 SGX에서 데이터를 유출할 수 있는 공격 방법이다. SGX는 Enclave라 불리는 안전하고 독립적인 실행환경을 제공하기 때문에 OS, 커널 등이 감염되어 있어도 Enclave 내부의 데이터, 코드를 안전하게 보호할 수 있다. 하지만 SGXPECTRE를 이용하면 SGX SDK를 이용하여 개발된 모든 Enclave에서 데이터를 유출할 수 있다[4].

SGXPECTRE는 [그림 6]과 같이 공격이 수행된다. 먼저 공격자는 희생자의 Enclave 내부에서 분기 조건을 수행할 때 Branch prediction에 의해 공격자가 원하는




〔그림 6〕 SGXPECTRE 공격 시나리오[4].

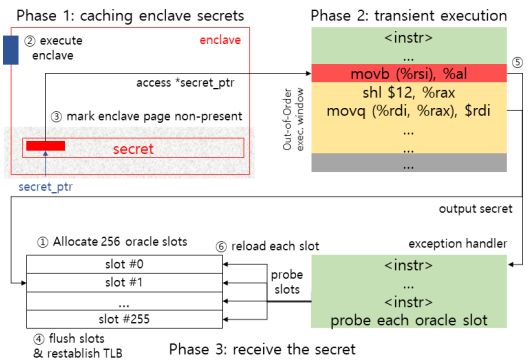
곳으로 잘못된 예측이 발생하도록 Branch predictor를 충분히 학습시킨다. 이 과정을 Poisoning이라 부른다. 희생자의 Enclave가 실행되고 Branch predictor는 Enclave 안에서 분기 조건을 수행할 때 공격자에 의해 학습된 대로 잘못된 예측을 수행하게 되고 이후 Speculative execution에 의해 Enclave 내부의 데이터가 캐시에 적재된다. 공격자는 Flush + Reload 공격으로 캐시에 적재된 데이터를 유출한다.

## 4.2. FORESHADOW

이전의 SGXPECTRE에서는 Enclave 내부의 데이터를 유출하기 위해서 Branch predictor가 잘못된 예측을 하도록 학습시키는 과정인 Poisoning이 필요했다. 하지만 FORESHADOW 공격에서는 별도의 Poisoning 없이도 Enclave 내부의 데이터를 유출하는 것이 가능하다[3].

Enclave는 Enclave 외부에서 Enclave에 접근하는 것을 막기 위해 외부 영역에서 Enclave에 접근하면 page fault를 발생시킨다. Enclave에서 page fault와 같은 예외가 발생하면 SGX는 예외를 직접 처리하지 않고 AEX(Asynchronous Enclave eXit)를 호출한다. AEX는 예외를 처리하기 전에 공격자에 의해 register 값이 유출되는 것을 방어하기 위해 예외가 발생한 시점의 register 값을 SSA(Save State Area)라 부르는 안전한 영역에 현재 register의 값을 백업하고 임의의 데이터로 register를 덮어쓴다. 이후 AEX는 시스템 소프트웨어의 예외 처리기를 호출하여 정상적으로 예외처리를 수행한다.

FORESHADOW는 [그림 7]과 같이 수행된다. FORESHADOW에서 공격자는 AEX에 의해 register의 값이 임의의 데이터로 덮어 씌워지는 것을 막기 위해 *mprotect()* 함수를 이용한다. *mprotect()* 함수는 해당 메모리 영역의 접근 권한을 설정할 수 있는 함수로 Enclave가 적재된 메모리 영역의 접근 권한을 모두 제거하면 AEX가 호출되지 않고 곧바로 시스템 소프트웨어의 예외 처리기가 호출된다. 이때 공격자가 미리 작성한 예외 처리기가 발생한 예외를 감지해 Enclave 내부의 데이터에 접근하는 공격 코드를 실행시킨다. 공격 코드는 Out-of-Order execution에 의해 실행되고 캐시에 그 데이터가 적재된다. 마지막 단계에서 공격자는 Flush + Reload 공격을 이용해 캐시에 적재된 데이터를 유출한다.




〔그림 7〕 FORESHADOW의 공격 시나리오[3].

## 4.3. Fallout

위와 같은 공격을 방어하기 위해 여러 대응 패치가 등장했지만 다양한 변종 공격이 계속 등장하고 있기 때문에 하나의 대응 패치로 유사한 모든 공격을 방어하기엔 어려운 실정이다. Fallout 또한 변종 공격 중 하나로 Meltdown 대응 패치가 이루어진 시스템에서도 Meltdown과 유사한 방법을 이용하여 시스템에서 데이터를 유출하는 공격이다[5].

```
char* victim_page = mmap(..., PAGE_SIZE, PROT_READ | PROT_WRITE,
                                          MAP_POPULATE, ...);
char* attacker_address = 0x9876543214321000ull;

int offset = 7;
victim_page[offset] = 42;

if (tsx_begin() == 0) {
  memory_access(lut + 4096 * attacker_address[offset]);
  tsx_end();
}

for (i = 0; i < 256; i++) {
  if (flush_reload(lut + i * 4096)) {
    report(i);
  }
}
```

〔그림 8〕 Fallout의 예시[5].

그중 Fallout은 Write Transient Forwarding을 이용하는 공격 방법이다. 프로세스가 메모리에 데이터를 쓰기 위해서는 가상 메모리 주소를 물리 메모리 주소로 변환한 후 그 주소에 데이터를 쓸 수 있는 권한을 얻어야 한다. 하지만 이 과정이 복잡하고 많은 단계를 수행해야 하기 때문에 데이터를 쓰는 작업의 대기 시간이 길어진다. 이러한 대기 시간을 최소화하기 위해 프로세서는 권한을 획득한 후에 메모리에 쓸 데이터와 메모리 주소를 CPU 내부의 저장 버퍼에 같이 저장해둔다. 이후 접근 권한을 획득하면 프로세서는 저장해둔 데이터를 이용하여 데이터를 쓴다.

프로세서는 저장 버퍼에 저장된 값을 실제로 메모리에 쓰기 위해 저장 버퍼에 저장된 메모리 주소와 실제로 데이터를 쓰려는 주소를 확인한 후 일치하면 데이터를 쓴다. 이 과정에서 프로세서는 효율성을 높이기 위해 저장 버퍼에 저장된 주소와 실제로 데이터를 쓰려는 주소가 일치하는지 전체를 비교하지 않고 하위 비트 일부만 비교한다. 공격자는 이점을 악용하여 공격자가 모니터링할 수 있는 영역이면서 동시에 프로세서에서 실제로 데이터를 쓰려는 주소와 하위 비트 일부가 같은 영역에 배열을 적재한다. 이후 Write Transient Forwarding에 의해 데이터가 공격자의 배열로 전달되면 공격자는 Flush + Reload와 같은 공격을 이용하여 데이터를 유출한다.

## Ⅴ. 결 론

Meltdown, Spectre와 같은 마이크로아키텍처 상의 취약점을 이용한 캐시 부채널 공격이 등장하면서 다양한 자원과 공격 측면을 활용하는 변종 공격이 등장하고 있다.

이러한 Meltdown, Spectre에 대응하기 위해 대응 패치를 배포했지만 모든 시스템에 적용하기 어렵고 다양한 변종 공격을 모두 방어할 수 없으며 Architecture 또는 Kernel 수정이 불가피한 실정이다. 또한, 대부분의 대응 패치들은 모두 성능저하를 유발하기 때문에 성능저하에 민감한 시스템의 경우 대응 패치를 적용하기가 어렵다.

다양한 변종 공격이 계속해서 등장하는 만큼 성능저하를 최소화하고 발생할 수 있는 다양한 변종 공격에 유동적으로 대응할 수 있는 연구가 활발히 이루어져야 할 것으로 여겨진다.

## 참 고 문 헌


[1] L. Moritz, S. Michael, G. Daniel, P. Thomas, H. Werner, F. Anders, H. Jann, M. Stefan, K. Paul, G. Daniel, Y. Yuval, H. Mike, “Meltdown: Reading kernel memory from user space.” In: 27th USENIX Security Symposium ({USENIX} Security 18). p. 973-990. 2018.

[2] K. Paul, H. Jann , F. Anders, G. Daniel, G. Daniel, H. Werner, H. Mike, L. Moritz, M. Stefan, P. Thomas, S. Michael, Y. Yuval, “Spectre attacks: Exploiting speculative execution.” In: 2019 IEEE Symposium on Security and Privacy (SP). IEEE, p. 1-19. 2019.

[3] B. V. Jo, M. Marina, W. Ofir, G. Daniel, K. Baris, P. Frank, S. Mark, W. F. Thomas, Y. Yuval, S. Raoul, “Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution.” In: 27th USENIX Security Symposium (USENIX Security 18). p. 991–1008. 2018.

[4] CHEN, G., CHEN, S., XIAO, Y., ZHANG, Y., LIN, Z., AND LAI, T. H. “Sgxpectre attacks: Leaking enclave secrets via speculative execution.” arXiv preprint arXiv:1802.09085. 2018.

[5] C. Claudio, G. Daniel, G. Lukas, G. Daniel, L. Moritz, M. Marina, M. Daniel, P. Frank, S. Michael, S. Berk, B. V. Jo, Y. Yuval, “Fallout: Leaking data on meltdown-resistant cpus.” In:

Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. p. 769-784. 2019.

[6] Y. Yuval, F. Katrina, “FLUSH+ RELOAD: a high resolution, low noise, L3 cache side-channel attack.” In: 23rd USENIX Security Symposium (USENIX Security 14). p. 719-732. 2014.

## 〈저 자 소 개〉

**윤 한 재 (HanJae Yoon)**

정회원

2016년 2월 : 한남대학교 컴퓨터공학과 졸업

2018년 2월 : 한남대학교 컴퓨터공학과 석사

2019년 3월~현재 : 한남대학교 컴퓨터공학과 박사과정

<관심분야> SGX, 아키텍처 보안, 부채널 공격

**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