Segmentation fault 모니터링을 통한 Foreshadow 및 ZombieLoad 공격 방어 기법 연구

이준연, 서태원
고려대학교 컴퓨터학과
E-mail: ny635@korea.ac.kr, suhtw@korea.ac.kr

A countermeasure against Foreshadow and ZombieLoad attacks based on segmentation fault monitoring

Jun-Yeon Lee, Taeweon Suh
Dept of Computer Science and Engineering, Korea University

요 약

2018년 Meltdown 공격이 발표된 이후 Foreshadow, ZombieLoad 등 다양한 종류의 마이크로아키텍처 기반 부 채널 공격과 방어 기법들이 발표되었다. 그중 Meltdown 공격을 원천 차단할 수 있는 KPTI (Kernel Page Table Isolation)는 커널 영역을 사용자 메모리 영역과 분리하여 커널 정보의 유출을 방어할 수 있으나, 최대 46%의 시스템 성능 저하를 가져온다. 본 연구는 런타임에 시스템콜 발생빈도를 분석하여 낮은 오버헤드로 Meltdown-type 공격을 탐지하고, 방어하는 프로그램을 개발하고 실험하였다. 개발한 Fault Monitoring Tool은 기존 시스템 대비 적은 오버헤드(최대 7%)로 악의적인 사용자를 구분해 내고 방어할 수 있었다.

1. 서 론

2018년 Meltdown[1], Spectre[2] 공격이 발표된 이후 마이크로아키텍처의 취약점을 대상으로 하는 다양한 공격이 발표되었다. 마이크로아키텍처 취약점을 대상으로 하는 공격은 최신 CPU의 비 순차 실행의 특성을 이용하여 Transient Execution을 실행함으로써 커널 정보와 같은 민감한 정보를 유출한다. 심지어 L1TF(L1 Terminal Fault)로 알려진 Foreshadow[3] 공격은 Intel의 신뢰 실행 환경인 SGX(Software Guard eXtensions)를 무력화시키며, 공격을 통해 SGX 인증에 사용되는 Attestation key를 유출할 수 있다.

최신 Intel 및 AMD 프로세서는 이러한 마이크로아키텍처 기반 부 채널 공격을 방어하기 위한 아키텍처 업데이트가 반영되었다. 하지만 하드웨어 기반의 방어 기법은 이전의 CPU를 사용하는 데스크톱이나 서버 등의 환경에는 적용할 수 없으며, 소프트웨어 기반의 방어 기법이 필요하다. Meltdown 공격을 완화하는 대표적인 소프트웨어 방어 기법인 KPTI[4](Kernel Page Table Isolation)는 커널 메모리 영역을 사용자 메모리 영역과 분리함으로써 사용자 영역에서 커널 메모리 정보를 유출할 수 없도록 원천적으로 차단한다. 하지만 이러한 솔루션은 운영체제의 지원을 받아 소프트웨어로 동작하기 때문에 필연적으로 성능 저하를 가져올 수밖에 없다.

본 연구에서는 Meltdown 공격을 방어하기 위한 선행 연구 중 하나인 MeltdownDetector[5] 기법을 응용하여 다양한 Meltdown 기반의 공격들에 적용 가능 여부를 확인해 보았다. MeltdownDetector는 런타임 환경에서 Segmentation Fault 발생을 관찰하고, Fault를 발생시킨 메모리 블록을 캐시 라인에서 Flush 시킴으로써 데이터 유출을 차단하는 방법이다.

Meltdown과 다르게 Foreshadow나 ZombieLoad[6] 공격은 PTE(Page Table Entry)나 Fill Buffer 등을 공격 벡터로 활용하여 캐시에 적재된 데이터를 유출한다. 그러므로 시스템콜을 모니터링하여 캐시 라인을 Flush 하는 방법은 데이터 유출에 노이즈를 줄 수 있으나, 공격을 원천적으로 차단할 수 없다. 본 연구에서는 Foreshadow와 ZombieLoad 공격을 대상으로 시스템콜 모니터링을 통해 악의적인 프로세스를 Kill 함으로써 공격을 효과적으로 방어할 수 있는 기법을 실험하였다.

2. 배경 지식

2.1 Transient Execution Attacks

최신 CPU는 명령어를 프로그램 순서대로 실행하지 않는다. CPU 파이프라인 내부의 실행 유닛(Execution Unit)의 사용을 극대화하기 위해 명령어를 비순차적으로 처리

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)

하는 Tomasulo 알고리즘에 기반을 둔 최적화 기술을 사용하고 있다.

Tomasulo 알고리즘은 Reservation Station(RS)과 Reorder Buffer (ROB)를 활용하여 명령어를 비순차적으로 처리할 수 있도록 한다. Fetch, Decode 단계를 거친 명령어는 실행 유닛에 할당되기 전 RS와 ROB 저장되며, 순서에 상관없이 바로 처리 가능한 명령어부터 실행 유닛에 할당된다. 실행을 마친 명령어는 완료된 결과를 RS에 Broadcast하고, ROB에 명령어를 완료 상태로 표시한다. ROB는 프로그램 순서에 따라 차례대로 명령어를 Commit(Retire)하고, Architectural State(레지스터 및 메모리 정보)를 업데이트한다. 또한, 명령어 실행 중에 발생한 예외(Exception)는 ROB에서 명령어를 Commit 하는 시점에 처리하여 순차적으로 수행될 수 있도록 한다. CPU의 Architectural State는 예외가 발생한 시점에서의 프로그램의 동작과 정확하게 일치하지만, 비순차적으로 미리 실행된 코드는 Microarchitectural State (Cache, TLB 등)를 변경시킨다.

Meltdown 공격의 경우 커널 메모리에 직접 접근하여 Exception을 고의로 발생시키며, 그와 동시에 커널 데이터를 참조하는 Transient Execution을 실행함으로써 L1 데이터 캐시에 커널 정보가 로드될 수 있도록 한다. 이후 Flush+Reload[7], Prime+Probe[8] 등의 캐시 부 채널 공격을 이용하여 캐시에 로드된 데이터를 누출한다. 이렇게 마이크로아키텍처의 특성을 이용하여 일시적으로 명령어를 실행함으로써 데이터를 유출하는 공격을 Transient Execution Attack이라고 한다.

2.2 Meltdown Attack and Variants

2018년 발표된 Meltdown 공격은 Transient Execution을 통해 L1 데이터 캐시에 일시적으로 로드된 커널 데이터를 탈취하는 공격이다. Meltdown 공격을 위해서는 공격자와 피 공격자 프로세스가 같은 물리적 코어를 공유해야 하며, Flush+Reload 공격을 이용하여 캐시에 적재된 데이터를 탈취한다. Flush+Reload 공격은 Inclusive 캐시의 특성을 이용한 공격으로 캐시에 적재된 데이터와 그렇지 않은 데이터 간의 Read Latency 차이를 이용하여 데이터를

```
1  void readbyte (
2              uint8_t secret_addr,
3              uint8_t probe_array) {
4
5    uint8_t value = *secret_addr;
6    value *= 0x1000;
7    uint64_t o = probe_array[value];
8  }
```

(그림 1) Meltdown 공격 코드 예제

유출한다. CPU에서 참조된 적이 있는 데이터의 경우 캐시에 데이터가 적재되어 있게 되므로, Read Latency가 짧게 (L1 Hit) 나타나며, 한 번도 참조되지 않은 데이터의 경우 길게 (L1/L2/LLC Miss) 나타나게 된다.

Meltdown 공격은 크게 세 단계로 이루어져 있다. 첫 번째 단계는 공격자 프로세스가 데이터 유출에 사용할 Probe 배열을 생성하고, 해당 배열을 모든 캐시 라인에서 Flush 시킨다. 데이터를 탐지하기 위한 Probe 배열은 256 × 4KB 크기로 구성하며, 256개의 배열을 통해 8bit의 데이터를 유출한다. 또한, 256개 배열에 4KB만큼의 크기를 곱해 같은 페이지 내에 배열의 항목이 존재하지 않게 한다. 두 번째 단계는 (그림 1)과 같이 커널 주소를 기반으로 Transient Execution을 수행한다. 라인 5에서 공격자는 추출하고자 하는 커널 주소의 데이터를 직접 접근한다. 이때 공격자는 커널 메모리에 대한 접근 권한이 없으므로 Exception이 발생하지만, 라인 6과 7은 비 순차 실행으로 인해 수행될 수 있다. 즉, 커널 데이터를 참조하여 Probe 배열의 한 항목이 캐시에 로드되게 된다. 세 번째 단계는 Probe 배열을 순서대로 Reload 해봄으로써 어떤 Probe 배열이 캐시에 적재되어 있는지 판별하는 단계이다. 라인 6에서 value 변수는 유출하고자 하는 커널 데이터 한 바이트를 담고 있으며, 그 값을 16으로 가정해보면, probe_array[16 × 0x1000] 배열이 캐시에 적재된다. 공격자는 Probe 배열을 0부터 255까지 접근해 보면서 Latency를 측정하고, 가장 Latency가 짧은 16번째 배열을 통해 데이터가 16 임을 유추할 수 있다. 이러한 공격을 반복적으로 수행하게 되면 공격자는 커널 메모리 전체를 읽어볼 수 있다.

이처럼 같은 물리적 코어에서 Transient Execution과 Exception을 활용한 형태의 공격을 Meltdown 변형 공격으로 분류하며[9], 대표적인 예로 Foreshadow(L1TF), ZombieLoad(MDS) 등이 있다. Foreshadow 공격은 PTE(Page Table Entry)의 Present Bit을 0으로 설정하여 고의로 Page Fault를 발생시키며 이를 통해 SGX 권한 검사를 회피하고, Flush+Reload를 통해 데이터를 유출한다. 반면, ZombieLoad 공격은 캐시의 Load Buffer를 통해 메모리로부터 데이터를 읽어올 때 할당하는 Fill Buffer Entry를 통해 데이터를 유출한다.

3. 실험 환경 및 결과

3.1 Fault 발생빈도 및 공격 성공률 분석

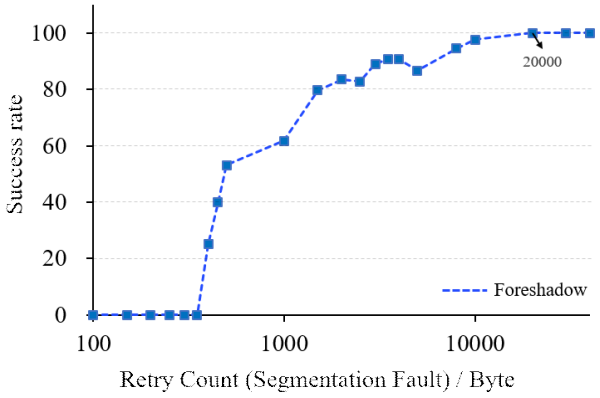
이처럼 Foreshadow와 ZombieLoad 공격은 고의로 Fault를 발생시켜 데이터를 유출한다는 공통점이 있었다. 즉 Segmentation Fault Handler에서 Flush+Reload 공격을 수행함으로써 데이터를 유출한다. 이러한 공격을 방어하기 위해서는 Segmentation Fault를 발생시키는 프로세스를 관찰하여 악의적인 코드를 진행할 수 없도록 하면 된다. 하지만 Segmentation Fault는 정상적인 애플리케이션에서도 발생할 수 있으므로 악의적인 사용자와 일반 사용자를

구분할 방법이 필요하다.

이러한 문제를 해결하고자 각각의 공격들을 수행해보며 Fault 발생빈도와 공격 성공률 간의 관계를 분석하였다. 실험 환경은 데스크톱 PC (Intel® CoreTM i7-6700, 3.4GHz, 16GB DDR4)를 사용하였고, Ubuntu 18.04.3 LTS (kernel 버전 : 5.0.0.27-generic)으로 운영체제를 구성하였다.

KPTI가 없는 환경에서 Meltdown-type 공격을 실험하기 위해 Linux Grub 설정에서 “nopti”, “nokaslr” 옵션을 추가하였다. 또한, 오픈소스로 공개된, PoC(Proof-of-Concept)[10][11]를 활용하여, Foreshadow, ZombieLoad 두 가지 공격의 양상을 분석하였다. 커널에서 발생하는 정보를 런타임에 디버깅 할 수 있게 해주는 Systemtap 툴을 사용하였고, 간단한 스크립트를 작성하여 SIGSEGV 발생빈도를 확인하였다. 비밀 정보는 총 64바이트의 문자열로 설정하였고, 64바이트 모두를 정확하게 유출 시킬 경우를 Success Rate 100%, 모든 데이터를 잘못 읽을 경우를 0%로 설정하였다.

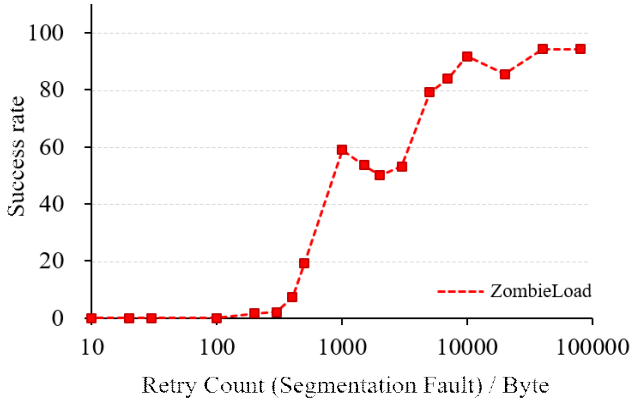
Foreshadow는 공격 노이즈를 필터링하기 위해 동일 주소 영역을 BYTE_READ_ATTEMPTS만큼 반복하여 읽으며, 유출한 데이터를 히스토그램으로 분석하여 최대 빈도의 데이터를 샘플링 한다. (그림 2)는 BYTE_READ_ATTEMPTS 값을 100에서부터 40,000까지 증가시켜가며 Foreshadow 공격을 실행했을 때 데이터 유출의 정확도를 측정한 결과이다. 반복 실행 횟수가 350회 이하에서는 0%, 즉 데이터를 전혀 유출할 수 없었고, 10,000회 이상일 때 100%의 정확도로 데이터를 유출할 수 있었다.



(그림 2) Foreshadow 공격의 반복 횟수와 데이터 유출 성공률 간의 관계

ZombieLoad PoC 역시 THRESHOLD라는 변수를 이용하여 데이터를 Sampling 하여 노이즈를 최소화하고 있었다. (그림 3)은 THRESHOLD를 10에서 100,000까지 증가시켜가며 ZombieLoad 공격을 실행했을 때 데이터 유출의 정확도를 측정한 결과이다. ZombieLoad의 경우 동일 데이터를 10만 회 반복 샘플링 하여도 100% 정확도를 달성하지 못하였고, 약 94%의 최대 정확도를 나타내었다.

두 공격을 Systemtap을 통하여 SIGSEGV를 모니터링 한 결과, 각 공격에서 사용한 반복 횟수는 SIGSEGV 발생 횟수와 정확하게 일치하였다. 즉 한 바이트를 읽기 위해 10,000번의 반복을 실행한다고 했을 때, 64바이트를 읽기 위해서 64만 번의 SIGSEGV가 발생하였다.



(그림 3) ZombieLoad 공격의 반복 횟수와 데이터 유출 성공률 간의 관계

3.2 Fault Monitoring을 통한 공격 차단

공격자가 데이터를 정확하게 읽지 못하도록 방어하기 위해 두 공격 모두 Success Rate가 0%일 때를 주목하였다. Foreshadow의 경우 350회 미만의 Retry Count에서는 데이터를 전혀 유출하지 못하였고, ZombieLoad의 경우 100회 미만의 Retry Count에서 데이터를 전혀 유출하지 못하였다.

```
probe signal.send then
    sig_pid ← pid()
    if signal_count[sig_pid] ≥ threshold then
        kill(sig_pid)
    end

    if sig == SIGSEGV then
        signal_count[pid()]++
    end
end
```

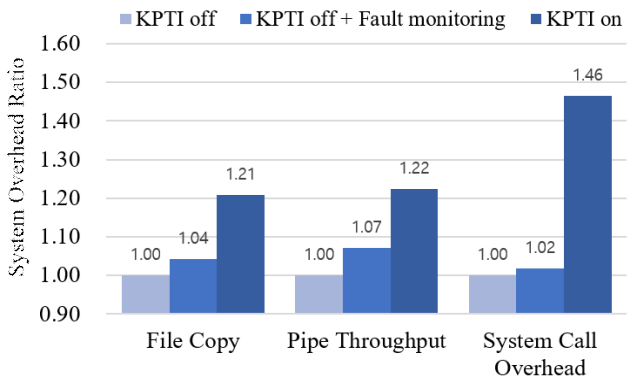
(그림 4) Segmentation Monitoring Tool 예제 코드

이러한 점에 착안하여 (그림 4)와 같은 방어 코드를 구현하였다. signal send probe 함수는 운영체제에서 signal이 발생하였을 때 주기적으로 호출하는 함수이며, 이 중 SIGSEGV 카운트만 저장하였다. 특정 프로세스 아이디의 SIGSEGV 카운트 값이 Threshold 값보다 클 때 (실험 환경은 100으로 세팅) 해당 프로세스를 악의적인 사용자로 판단하고 코드를 수행하지 못하도록 Kill 명령을 통해 프로세스를 중단시켰다.

3.3 Fault Monitoring Tool 성능 분석

구현한 Fault Monitoring Tool의 성능 오버헤드를 분석하기 위해 KPTI 기능을 사용할 때(기존 솔루션), 와 사용하지 않을 때(취약한 환경), 그리고 KPTI를 사용하지 않으면서 Fault Monitoring Tool을 사용할 때를 비교하였다. 벤치마크 도구는 유닉스 계열 시스템의 성능 분석 도구인 Unixbench[12]를 이용하였다. (Unixbench 5.1.3)

(그림 5)는 Unixbench를 사용한 성능 분석 결과를 나타낸다. KPTI가 없을 때(off)의 성능 오버헤드를 1.00으로 두고, 세 가지 애플리케이션에 대해 Fault Monitoring Tool을 사용했을 때와 KPTI를 사용했을 때의 성능을 비교한다. 기존의 솔루션인 KPTI를 사용했을 때 최대 1.46배까지 성능 오버헤드(시스템 콜 오버헤드)가 나타나며, File copy에 대한 성능 또한 1.21배만큼 느려졌다. 본 연구에서 실험한 Fault Monitoring Tool은 벤치마크에서 사용한 File copy, Pipe throughput, System call overhead 모든 측면에서 KPTI 대비 성능이 우수하였으며, KPTI를 사용하지 않을 때 대비 최대 7%의 성능 오버헤드가 있었다.



(그림 5) Fault Monitoring Tool의 성능 오버헤드 분석
[실험 환경: Intel i7-6700, Ubuntu 18.04, single-core]

4. 결론 및 향후 연구 계획

본 연구는 리눅스 환경에서 런타임에 시스템에 발생한 Fault를 모니터링 함으로써 악의적인 애플리케이션을 탐지하고, 프로세스를 중단시키는 방법에 대해 실험하였다. 개발한 Fault Monitoring Tool을 통해 Foreshadow나 ZombieLoad와 같은 Meltdown-type 공격을 효과적으로 방어할 수 있었다. 이는 기존 시스템에 최대 7%의 오버헤드만을 가져오며, Meltdown 완화 기법인 KPTI 대비 최대 약 43% 성능적인 이점이 있었다.

다만, 본 방어 기법은 OS 레벨에서 시스템 콜을 모니터링 하기 때문에 OS의 Exception Handler를 건너뛸 수 있는 Intel TSX를 활용한 공격을 탐지할 수 없다. 향후 TSX를 활용한 공격을 탐지하고, 방어하는 방법에 관해 추가 연구를 진행할 예정이다.

참고문헌

[1] Lipp, M., Schwarz, M., Gruss, D., Prescher, T., Haas, W., Fogh, A., ... & Yarom, Y. (2018). Meltdown: Reading kernel memory from user space. In 27th USENIX Security Symposium. (pp. 973-990).

[2] Kocher, P., Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., ... & Yarom, Y. (2018). Spectre attacks: Exploiting speculative execution. arXiv preprint arXiv:1801.01203.

[3] Van Bulck, J., Minkin, M., Weisse, O., Genkin, D., Kasikci, B., Piessens, F., ... & Strackx, R. (2018). Foreshadow: Extracting the keys to the intel {SGX} kingdom with transient out-of-order execution. In 27th USENIX Security Symposium. (pp. 991-1008).

[4] INTEL, Intel Analysis of Speculative Execution Side Channels, White Paper Revision 1.0, January 2018.

[5] Akyildiz, T. A., Guzgeren, C. B., Yilmaz, C., & Savas, E. “MeltdownDetector: A Runtime Approach for Detecting Meltdown Attacks. IACR Cryptology ePrint Archive, 2019, 613.

[6] Schwarz, M., Lipp, M., Moghimi, D., Van Bulck, J., Stecklina, J., Prescher, T., & Gruss, D. (2019). ZombieLoad: Cross-privilege-boundary data sampling. arXiv preprint arXiv:1905.05726.

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

[8] Liu, F., Yarom, Y., Ge, Q., Heiser, G., & Lee, R. B. (2015). Last-level cache side-channel attacks are practical. In 2015 IEEE Symposium on Security and Privacy. (pp. 605-622).

[9] Canella, C., Van Bulck, J., Schwarz, M., Lipp, M., Von Berg, B., Ortner, P., ... & Gruss, D. (2019). A systematic evaluation of transient execution attacks and defenses. In 28th USENIX Security Symposium. (pp. 249-266).

[10] Gregvish, L1TF(Foreshadow) VM guest to host memory read PoC (2019), Github repository, https://github.com/gregvish/l1tf-poc

[11] IAIK, ZombieLoad PoC (2019), Github repository, https://github.com/IAIK/ZombieLoad

[12] kdlucas, byte-unixbench (2019), Github repository, https://github.com/kdlucas/byte-unixbench