Processing-In Memory 시간적 접근 취약점 분석 및 완화에 대한 연구

김태욱[1], 조영필[2]
[1]한양대학교 미래자동차-SW융합전공 석사과정
[2]한양대학교 컴퓨터소프트웨어학과 조교수
qkenr7895@hanyang.ac.kr, ypcho@hanyang.ac.kr

A Study on the Analysis and Mitigation of Temporal Access Vulnerability in Processing-In Memory

Tae-Wook Kim[1], Yeongpil Cho[2]
[1]Dept. of Computer and Software (Automotive-Computer Convergence), Hanyang University
[2]Dept. of Computer and Software, Hanyang University

요 약

많은 양의 데이터 처리를 요구하는 오늘날, 메모리 입/출력 없이 데이터를 처리할 수 있는 Processing-In Memory가 많은 관심을 받고 있다. Processing-In Memory는 소프트웨어 라이브러리를 통해 접근할 수 있는데, 적절히 구현되지 않은 라이브러리는 공격 대상이 된다. 본 논문에서는 Processing-In Memory 소프트웨어 라이브러리에 존재하는 시간적 접근 취약점을 분석하고 그에 대한 완화기법을 제시한다.

1. 서론

Processing-In Memory는 메모리 칩 내부에 연산기가 부착되어 메모리 입/출력 없이 데이터를 처리할 수 있는 새로운 형태의 메모리이다. 대용량 데이터 처리의 병목이 되는 메모리 입/출력을 없애 빠른 연산을 가능케 하여 Processing-In Memory는 빅데이터, AI 분야 등에서 많은 관심을 받고 있다.

Processing-In Memory의 사용은 제조사가 제공하는 소프트웨어 라이브러리를 통해 가능하다. 이러한 소프트웨어 라이브러리는 Processing-In Memory 사용을 편리하게 해주지만 적절히 구현되지 않을 경우 공격 경로로서 활용될 수 있다.

따라서 본 연구에서는 Processing-In Memory 라이브러리에 존재하는 취약점을 분석하고 완화기법을 제시하고자 한다.

2. Processing-In Memory 구조

Processing-In Memory (이하 PIM)를 유일하게 상용화한 UPMEM사의 PIM 구조를 그림 1에서 확인할 수 있다 [1]. PIM 시스템은 가장 작은 연산 단위인 DPU의 집합으로 이루어진다. DPU는 다시 RISC 기반의 프로세서와 24KB의 명령어 메모리 (IRAM), 64KB의 스크래치패드 메모리 (WRAM), 64MB의 DRAM (MRAM)으로 구성된다.

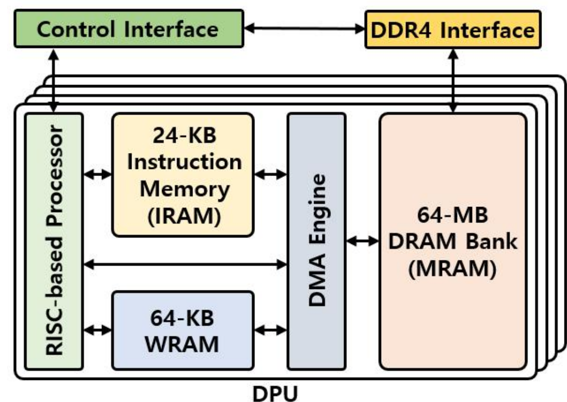

그림 1 UPMEM Processing-In Memory 구조

DPU는 자신의 메모리 (IRAM과 WRAM, MRAM)만 접근할 수 있고 다른 DPU의 메모리나 호스트 CPU의 메모리 (예: DRAM)는 접근이 불가능하다. 호스트 CPU는 DPU의 메모리에 접근할 수 있고 그 역은 불가능하기 때문에, 호스트 CPU-DPU 그리고 DPU-DPU간 통신은 호스트 CPU가 전적으로 관리한다.

UPMEM사는, 그림 2에서 보이는 것처럼, DPU관리 및 DPU 통신을 편리하게 해주는 PIM 소프트웨어 라이브러리를 제공하고 있다. 이러한 라이브러리를 통해 호스트 CPU에서 DPU를 사용하는 일반적인 과정은 다음과 같다. (1) DPU 할당 (2) DPU 바이너리 및 데이터 적재 (3) DPU 실행 (4) DPU 결과 읽기 (5) DPU 할당 해제. 구체적으로, 호스트 CPU가 DPU 사용을 요청하면 라이브러리는 현재 사용중이지 않은 DPU를 할당한다. 성공적으로 DPU를 할당받았으면, 호스트 CPU는 DPU에서 실행할 DPU 바이너리와 실행에 필요한 데이터를 DPU에 적재한다. 그리고 DPU를 실행하여 원하는 작업을 수행한 후, 라이브러리를 통해 수행 결과를 얻는다. 마지막으로, 모든 DPU 사용이 끝났으면, DPU를 할당 해제 한다.

```
int main() {
    // DPU 할당
    dpu_alloc(1024, NULL, &dpu);

    // DPU 바이너리 및 데이터 적재
    dpu_load(dpu, DPU_BINARY, NULL);

    // DPU 실행
    dpu_launch(dpu, DPU_ASYNCHRONOUS);

    // DPU 결과 읽기
    dpu_log_read(dpu, stdout);

    // DPU 할당 해제
    dpu_free(dpu)
}
```

그림 2 UPMEM PIM 라이브러리

3. 시간적 접근 취약점 분석

현재의 UPMEM PIM 라이브러리는 DPU를 할당하고 해제하는 과정에서 DPU 메모리를 초기화하지 않는다. 즉, 이전 호스트 CPU가 사용했던 DPU의 데이터가 그대로 남는, 시간적 접근 취약점이 존재한다. 물론 DPU 바이너리와 데이터를 적재하는 과정에서, 이전 호스트 CPU가 사용했던 DPU 데이터 일부가 지워지긴 한다. 하지만, 호스트 CPU는 DPU에 직접 접근이 가능하기 때문에, DPU 할당 후 DPU에 바로 접근하면 이전 호스트 CPU의 데이터를 고스란히 얻을 수 있다.

이에 더해, UPMEM PIM 라이브러리의 DPU 할당 메커니즘은 공격자로 하여금 시간적 접근 공격을 용이하게 해준다. 각 DPU는 고유의 ID를 가지는데, UPMEM PIM 라이브러리는 가장 낮은 ID를 가지는 DPU부터 먼저 할당한다. 예를 들어 호스트 A가 0번 ID의 DPU를 사용하고 반납했다고 하자. 이런 상황에서 호스트 B가 DPU 할당을 요청하면 PIM 라이브러리는 가장 낮은 ID를 가지는 0번 DPU를 다시 할당해준다. 이렇듯, UPMEM PIM 라이브러리의 DPU 할당 메커니즘은 직전 호스트가 사용했던 DPU를 다시 할당해주기 때문에, 간단하게 시간적 접근 공격이 가능하다.

4. 시간적 접근 취약점 완화

DPU 할당 시, DPU 메모리를 초기화하면 시간적 접근 취약점을 완화할 수 있다. 메모리 초기화를 통한 완화 전과 후의 DPU 할당 시간 소요를 표 1에서 확인할 수 있다. DPU 할당 개수에 따라 할당시간이 980%, 1275%, 2153%, 1405% 만큼 증가한 것을 볼 수 있는데, 왜냐하면 각 DPU 마다 64MB 크기에 달하는 메모리를 초기화해야 하기 때문이다. 이렇듯, 메모리 초기화를 통한 시간적 접근 취약점 완화는 엄청난 오버헤드를 만들지만 실제 상황에서는 감당가능하다. 왜냐하면 메모리 초기화는 DPU 할당시에만 이루어지고, DPU 할당은 DPU 사용 과정에서 한번만 수행되기 때문이다. 설령 DPU 할당 및 해제를 빈번히 하는 호스트가 있다고 하더라도, 이전에 DPU를 사용했던 호스트와 현재 DPU를 사용할 호스트가 같을 경우 메모리 초기화를 뛰어 넘으면 오버헤드를 더욱 줄일 수 있다.

	DPU 할당 개수			
	1	64	256	1024
완화 전	0.05초	0.08초	0.15초	0.54초
완화 후	0.49초	1.02초	3.23초	7.59초

표 1 DPU 할당 시간 소요 (단위: 초)

5. 결론

본 연구에서는 Processing-In Memory 소프트웨어 라이브러리에 존재하는 취약점을 분석하고 완화기법을 제시하였다. 보다 구체적으로, 이전 호스트 CPU가 사용했던 DPU의 데이터가 그대로 남아있는, 시간적 접근 취약점이 존재한다는 것을 밝혔고 메모리 초기화를 통한 완화 기법을 제시하였다. 앞으로의 연구에서는 자동화되고 체계적인 취약점 분석 도구

를 활용해 보다 다양한 취약점을 분석할 계획이다.

이 논문은 과학기술정보통신부의 재원으로 정보통신기획평가원(No. 2020-0-01840, 스마트폰의 내부데이터 접근 및 보호 기술 분석)과 한국연구재단(No. NRF-2022R1A4A1032361, Processing-in-Memory 보안 기술 개발)의 지원을 받아 수행된 연구임

참고문헌

[1] UPMEM, “Introduction to UPMEM PIM. Processing-in-memory (PIM) on DRAM Accelerator (White Paper),” 2018.