@daebalprime

HeartBleed & Spectre : 유명한 취약점

HeartBleed

- OpenSSL에서 발생하는 취약점 (SSL 프로토콜을 구현한 오픈소스)
- Application Layer에서 외부로 보낼 데이터를 TCP로 보내지 않고 SSL에 보내면, SSL이 암호화하여 TCP로 전송함
- 여러 암호화 기술 등이 있으나 생략

HeartBleed

- OpenSSL에서 발생하는 취약점 (SSL 프로토콜을 구현한 오픈소스)
- Application Layer에서 외부로 보낼 데이터를 TCP로 보내지 않고 SSL에 보내면, SSL이 암호화하여 TCP로 전송함
- 여러 암호화 기술 등이 있으나 생략

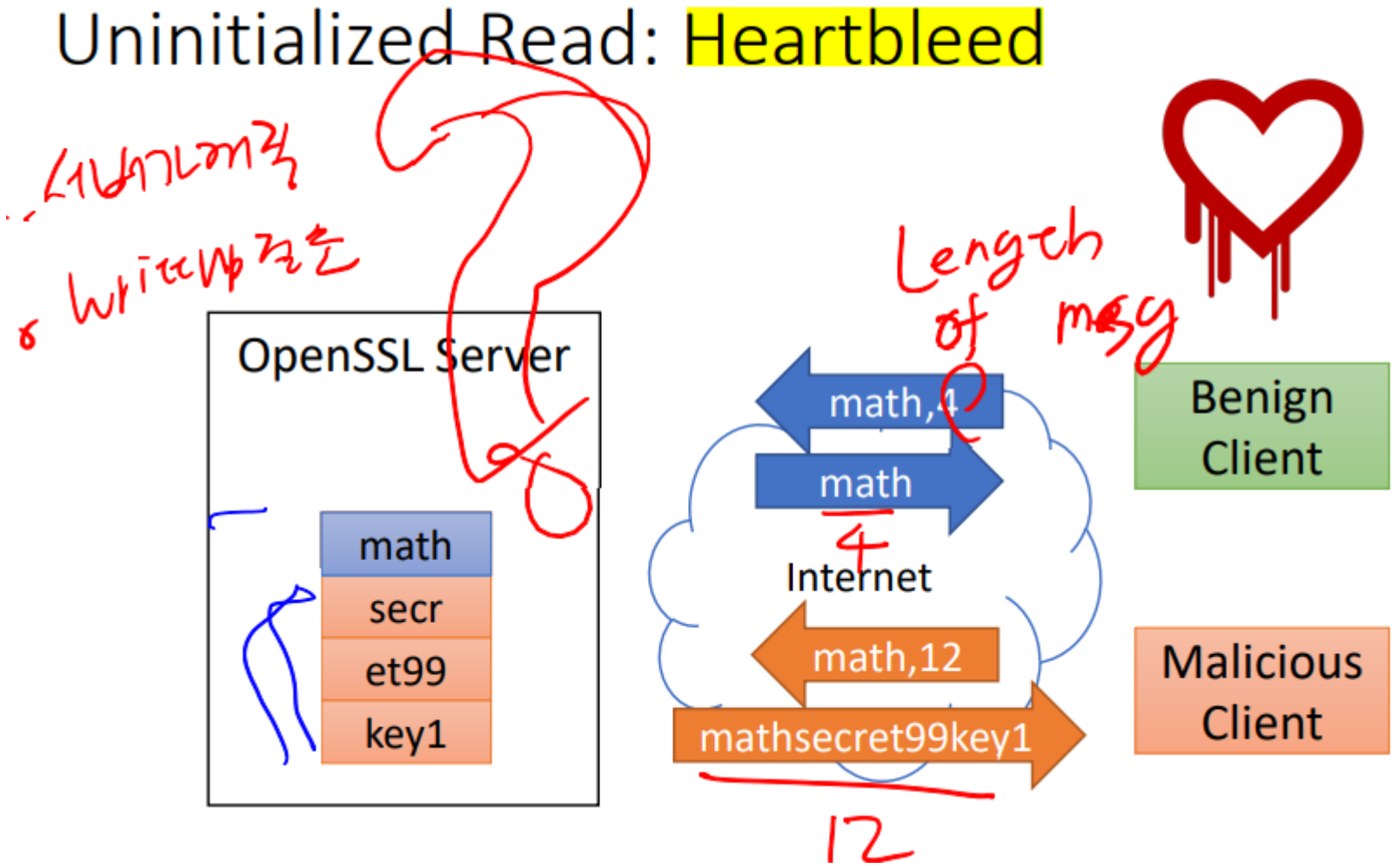
HeartBleed - HeartBeat

- OpenSSL의 확장 규격 중 하나인 HeartBeat는 연결을 유지하기 위한 목적으로 주기적인 신호를 보낼 때 사용
- 임의의 정보를 정보의 길이와 함께 서버에 전송하면, 서버는 전달 받은 정보를 다시 클라이언트에 응답함으로써 서버가 살아있음을 증명
- 문제는 정보의 실제 길이와 주장하는 길이가 다를 때 문제
- 실제 길이가 10인 정보와 가정할 때, 길이가 1000이라고 주장하는 패킷을 보내게 되면 서버가 나머지 990만큼의 정보를 메모리로부터 더 읽어서 보내줌
- C/C++의 경계검사를 생각하면 편할 듯

HeartBleed - HeartBeat

- OpenSSL의 확장 규격 중 하나인 HeartBeat는 연결을 유지하기 위한 목적으로 주기적인 신호를 보낼 때 사용
- 임의의 정보를 정보의 길이와 함께 서버에 전송하면, 서버는 전달 받은 정보를 다시 클라이언트에 응답함으로써 서버가 살아있음을 증명
- 문제는 정보의 실제 길이와 주장하는 길이가 다를 때 문제
- 실제 길이가 10인 정보와 가정할 때, 길이가 1000이라고 주장하는 패킷을 보내게 되면 서버가 나머지 990만큼의 정보를 메모리로부터 더 읽어서 보내줌
- C/C++의 경계검사를 생각하면 편할 듯
- 패킷은 64kb의 용량 제한이 있지만, 반복적으로 수행하여 중요 정보가 노출될 수 있음

HeartBleed - HeartBeat



HeartBleed - HeartBeat

```
bp = buffer;

/* Enter response type, length and copy payload */
*bp++ = TLS1_HB_RESPONSE;
s2n(payload, bp);
memcpy(bp, pl, payload);
bp += payload;
/* Random padding */
RAND_pseudo_bytes(bp, padding);
```

```
if (write_length > SSL3_RT_MAX_PLAIN_LENGTH)
return 0;

/* Allocate memory for the response, size is 1 byte
* message type, plus 2 bytes payload length, plus
* payload, plus padding
*/
buffer = OPENSSL_malloc(write_length);
bp = buffer;

/* Enter response type, length and copy payload */
*bp++ = TLS1_HB_RESPONSE;
s2n(payload, bp);
memcpy(bp, pl, payload);
bp += payload;
/* Random padding */
RAND_pseudo_bytes(bp, padding);
```

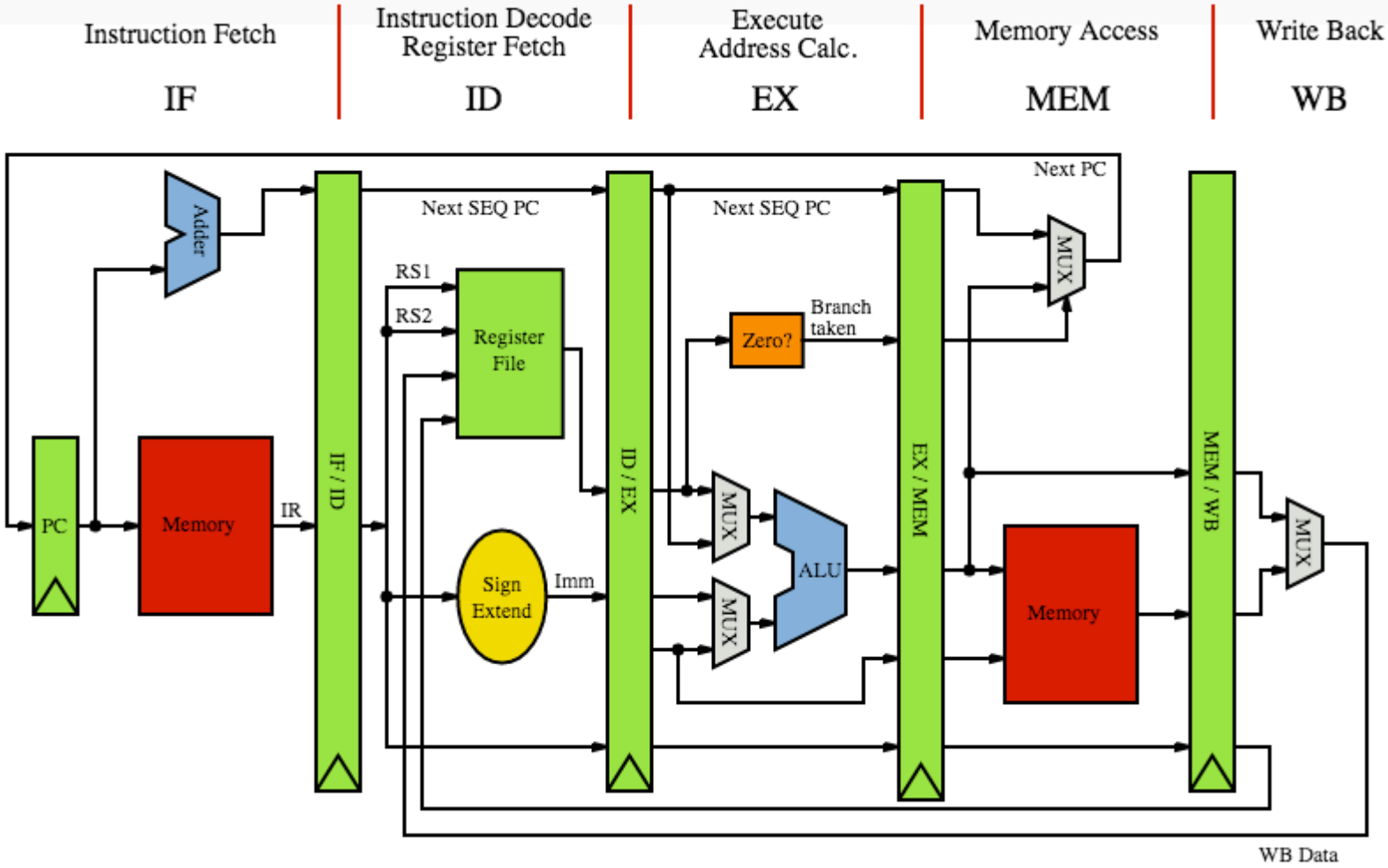
HeartBleed – 무엇이 문제인가?

- 패킷의 손상 여부나 데이터 무결성 등의 검사 등의 이유로 전달받는 데이터의 길이를 메타 데이터로 받는다면, 실제 길이와 주장하는 길이가 일치하지 않는다면 아무 응답을 하지 않는다던지 했어야 했음
- Array Bound Check를 제대로 구현하지 않았음
- 단순한 실수가 불러온 중대한 재앙
- 백엔드 개발자로써 교훈 : 클라이언트가 보내는 정보는 믿지 않아야 한다. 프론트단에서 유효성 검사를 할 지라도!

Spectre

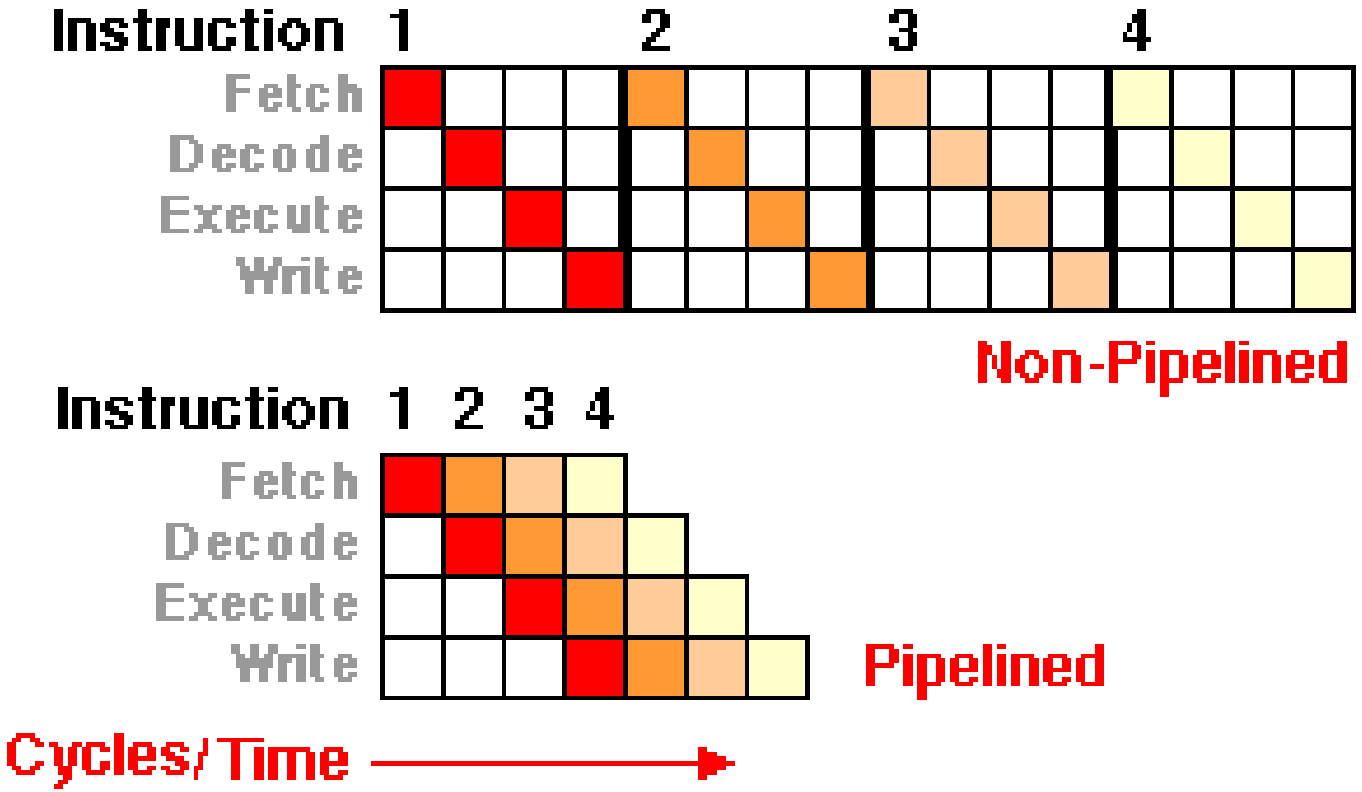
- CPU 파이프라인, 캐시, 분기 예측으로 발생하는 취약점
- CPU는 최적화를 위해 Pipeline과 Superscalar를 이용하여 CPU Cycle을 획기적으로 줄임

Spectre


Instruction Fetch
IF
Instruction Decode
Register Fetch
ID
Execute
Address Calc.
EX
Memory Access
MEM
Write Back
WB
Adder
Next SEQ PC
Next SEQ PC
Next PC
MUX
RS1
RS2
Register
File
Branch
taken
Zero?
PC
Memory
IR
IF / ID
ID / EX
EX / MEM
MEM / WB
MUX
MUX
ALU
Sign
Extend
Imm
Memory
MUX
WB Data

Spectre – CPU Pipeline

- 하나의 명령어가 먼저 실행되는 명령어를 처리가 끝날 때 까지 대기하지 않고, 동시에 수행하는 방법.



Spectre – Speculative Execution

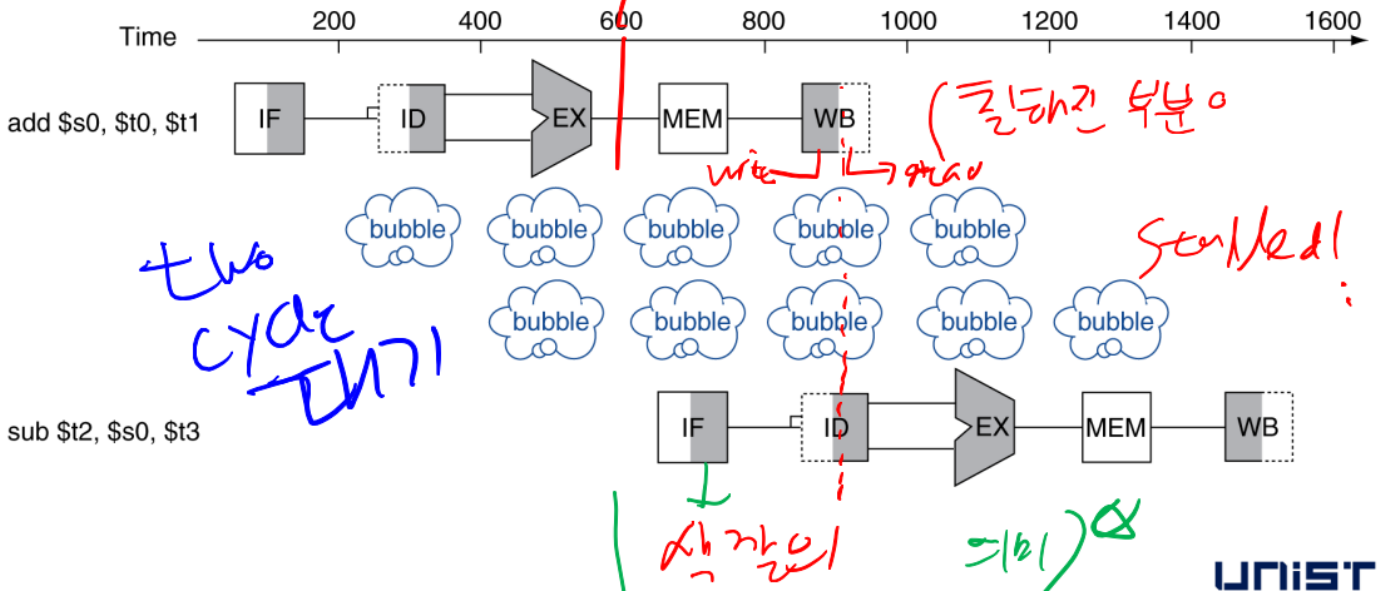
- 파이프라인을 적용하면, 예를 들어 조건문의 조건이 참/거짓이 판별되기도 전에 다음 코드가 파이프라인에 들어와 이미 실행중일 것
- 프로그램의 논리 흐름 상 실행여부가 불확실한 상태에서, branch prediction 결과에 따라 미리 다음 코드를 실행하는 것
- 만약 미리 실행했는데 예측이 틀렸다면 모든 변경점은 취소된다.
- getSomething()을 계산하는데 파이프라인 크기 이상의 연산이 필요하다면, y=3을 getSomething의 끄트머리 파이프라인에서 미리 실행하게 된다.

```
if (getSomething() > 10)
y = 3
```

Spectre – Hazard

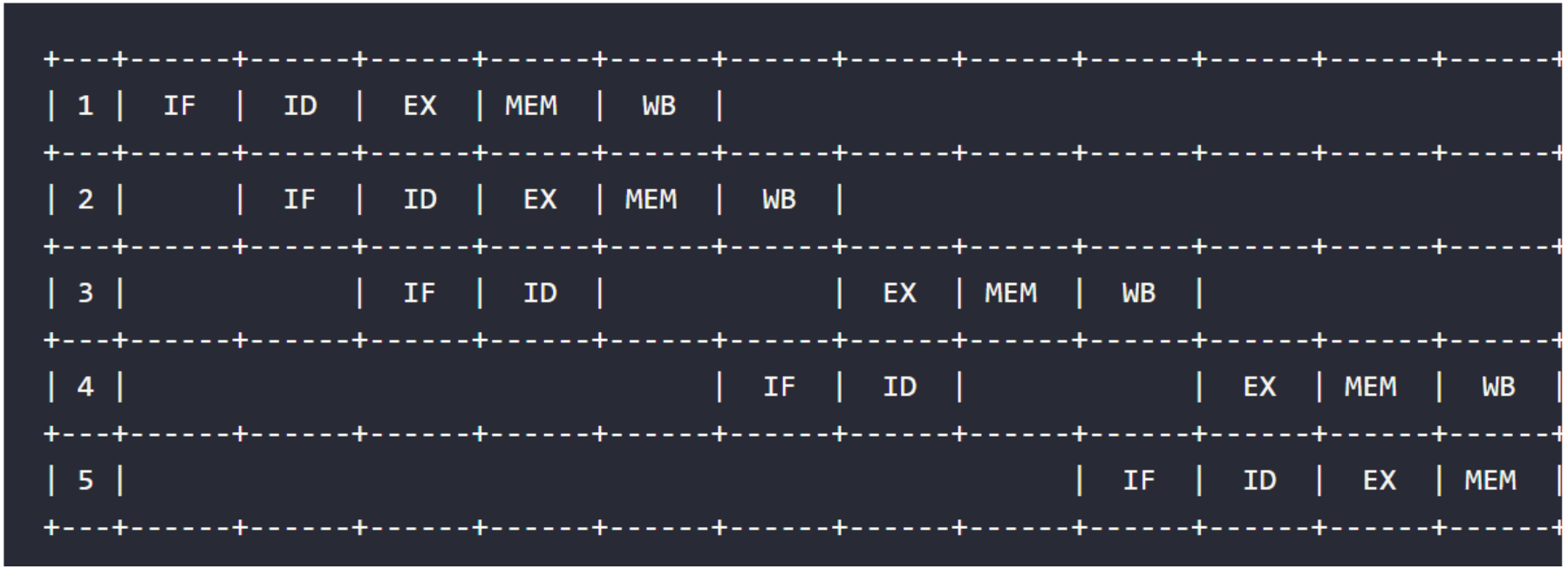
Data Hazards

- An instruction depends on completion of data access by a previous instruction
  - `add  $s0, $t0, $t1`
    `sub  $t2, $s0, $t3`



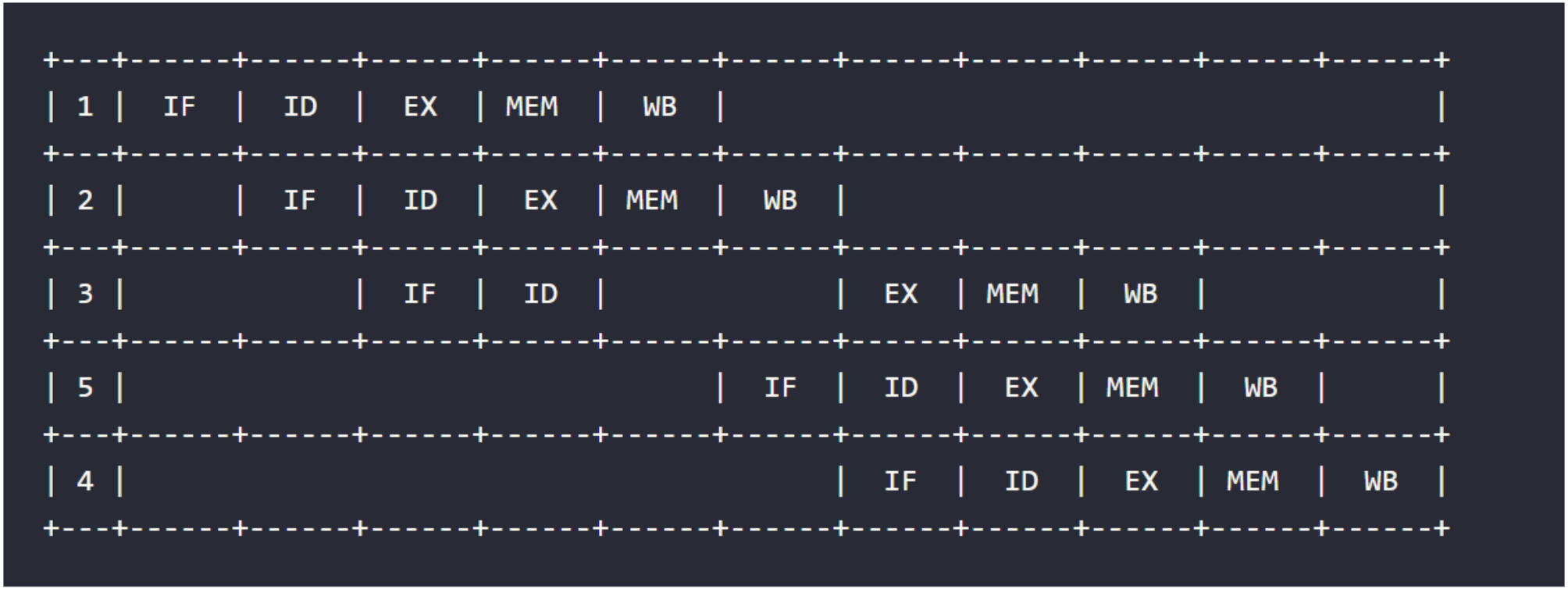
Spectre – Superscalar

- 의존성이 없는 명령어를 먼저 실행함으로써 파이프라인의 효율을 높이고 같은 결과를 보장하는 기법



Spectre – Superscalar

- 의존성이 없는 명령어를 먼저 실행함으로써 파이프라인의 효율을 높이고 같은 결과를 보장하는 기법



Spectre – 실제 취약점 공략

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

변수 `x` 는 공격자가 설정하는 임의의 값으로, 배열의 메모리 범위를 넘는 값(Out of Bounds)을 넣는다. 따라서 `array1[x]` 는 허용되지 않은 메모리 공간에 접근한다. 여기에 메모리에 있는 다른 페이지들을 읽기 위해 페이지 사이즈인 `4096` 을 곱한다. 이때 공격자가 노리는 `array1[x]` 를 시크릿 바이트(Secret byte) `k` 라고 한다.

하지만 허용되지 않은 메모리 공간에 접근하면 세그먼트 폴트(Segment faults) 오류가 발생한다. 그래서 직접 해당 구문을 실행하지 않고 브랜치 예측을 이용한다.

1. 프로세서가 조건문의 조건을 참으로 예측하도록 유도한다. (조건이 참인 상황을 여러 번 반복한다.)

2. 이제 조건이 거짓이 되도록 하면 브랜치 예측이 실패한다. 코드상으로는 조건문 내부의 명령이 실행되지 않는 것처럼 보이지만, 추측 실행으로 인해 실제로는 명령이 실행된다.

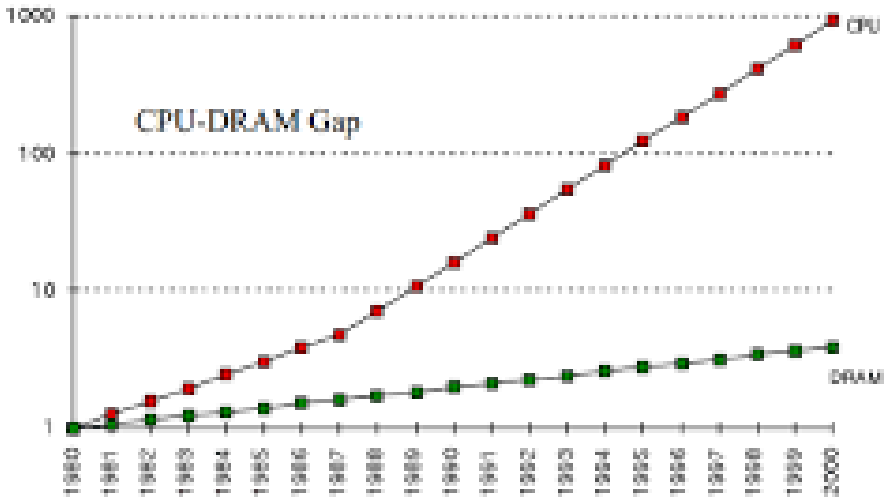
3. 예측이 실패했으므로 프로세서는 실행한 명령을 되돌린다. `y` 에는 아무런 값도 저장되지 않는다.

4. 명령이 취소되기는 했지만, 실제로는 데이터를 읽었기 때문에 취소된 명령이 접근한 데이터 `k * 4096` 은 캐시에 올라가 남아있게 된다. 이때 `k` 는 허용된 메모리 범위를 넘는 값이기 때문에 해당 프로세스에서 접근할 수 없다.

5. 데이터에 직접 접근할 수 없기 때문에 공격자는 캐싱된 데이터를 이용한다. 공격자는 `array2` 의 모든 요소를 무작위로 접근하면서 접근 시간을 측정한다. 접근 시간이 특히 짧은 데이터는 캐시에 저장되어 있다는 의미이므로, 이는 추측 실행으로 캐싱된 `k * 4096` 를 뜻한다. 멜트다운 취약점을 공격할 때 사용한 방법과 동일하다.

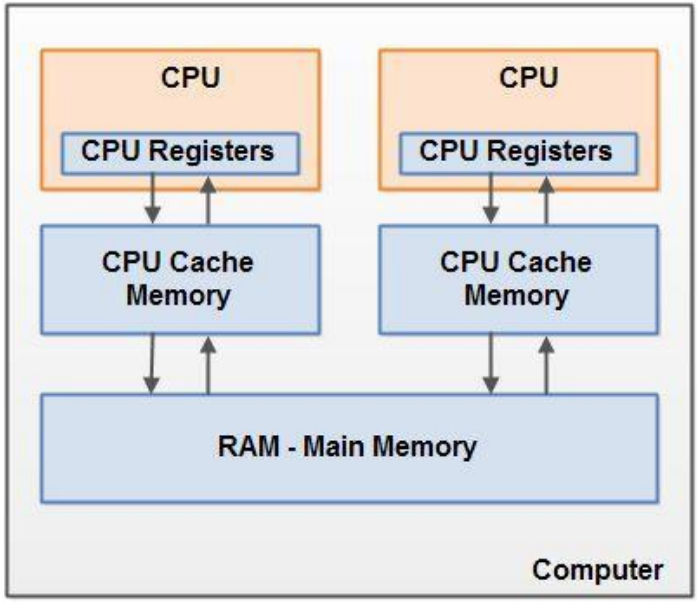
Spectre – 실제 취약점 공략

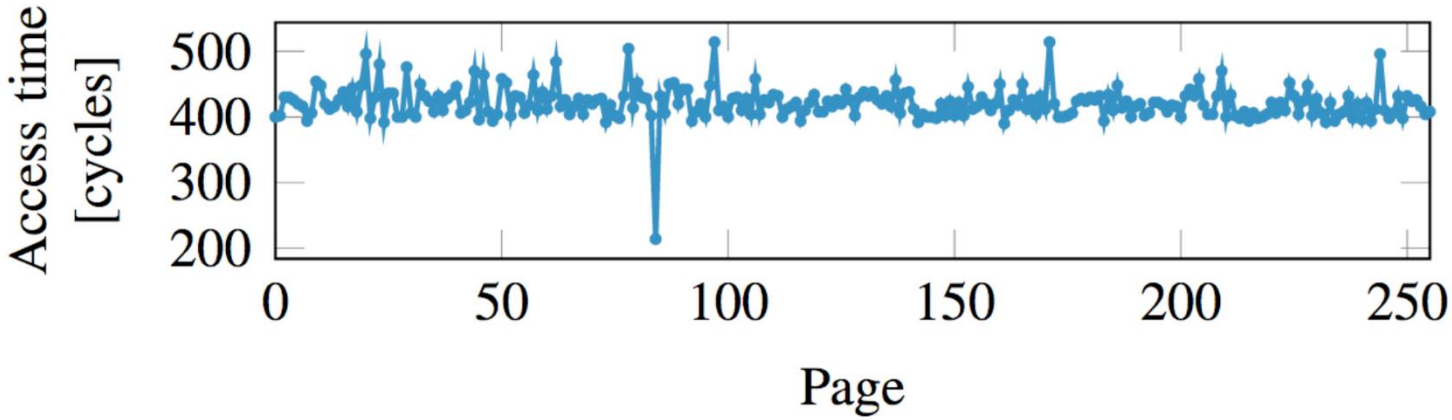
- Processor vs Memory Performance



1980: no cache in microprocessor;

1995 2-level cache





Reference

- https://blog.alyac.co.kr/76
- https://parksb.github.io/article/31.html