Runtime Randomized Relocation of Crypto Libraries for Mitigating Cache Attacks

신 영 주

July 28, 2021

고려대학교 정보보호대학원
Korea University
School of Cybersecurity

CSSLab
Computer System Security Laboratory

Moving Target Defense (MTD) 전략을 이용한

캐시 부채널 공격 대응 기법

고려대학교 정보보호대학원
Korea University
School of Cybersecurity

CSSLab
Computer System Security Laboratory

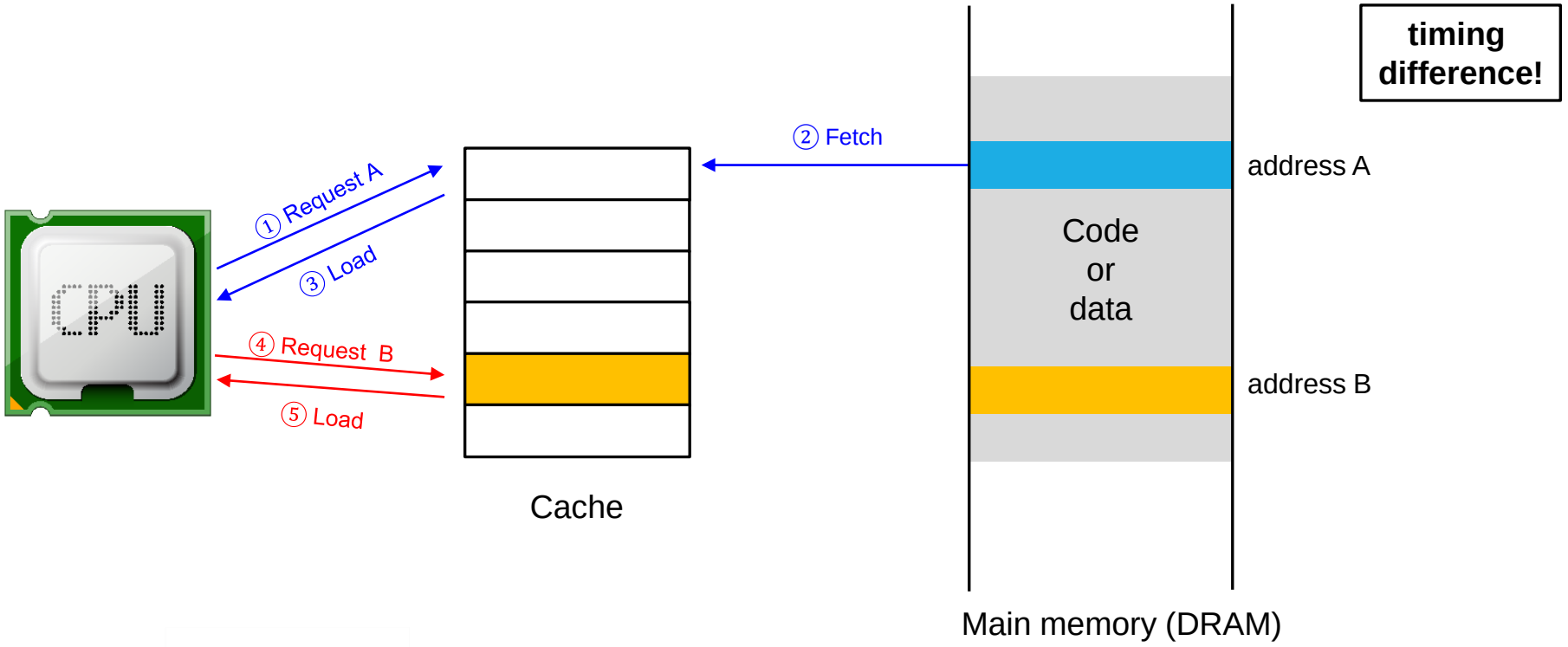
Moving Target Defense

- 개발자가 아니라 공격자(해커 등)를 괴롭히는 형태의 보안 기법
- *"공격의 대상이 되는 시스템 자체의 구성을 능동적이고 지속적으로 변화시킴으로써 공격자가 대상 시스템의 취약점 자체를 찾기 어렵게 하기 위한 전략" - KISA*

캐시 부채널 공격 (CSA) 이란?

CPU 캐시

- Exploits principle of locality
- Request for data at location A is served with high latency (Cache miss)
- Request for data at location B is served with low latency (Cache hit)

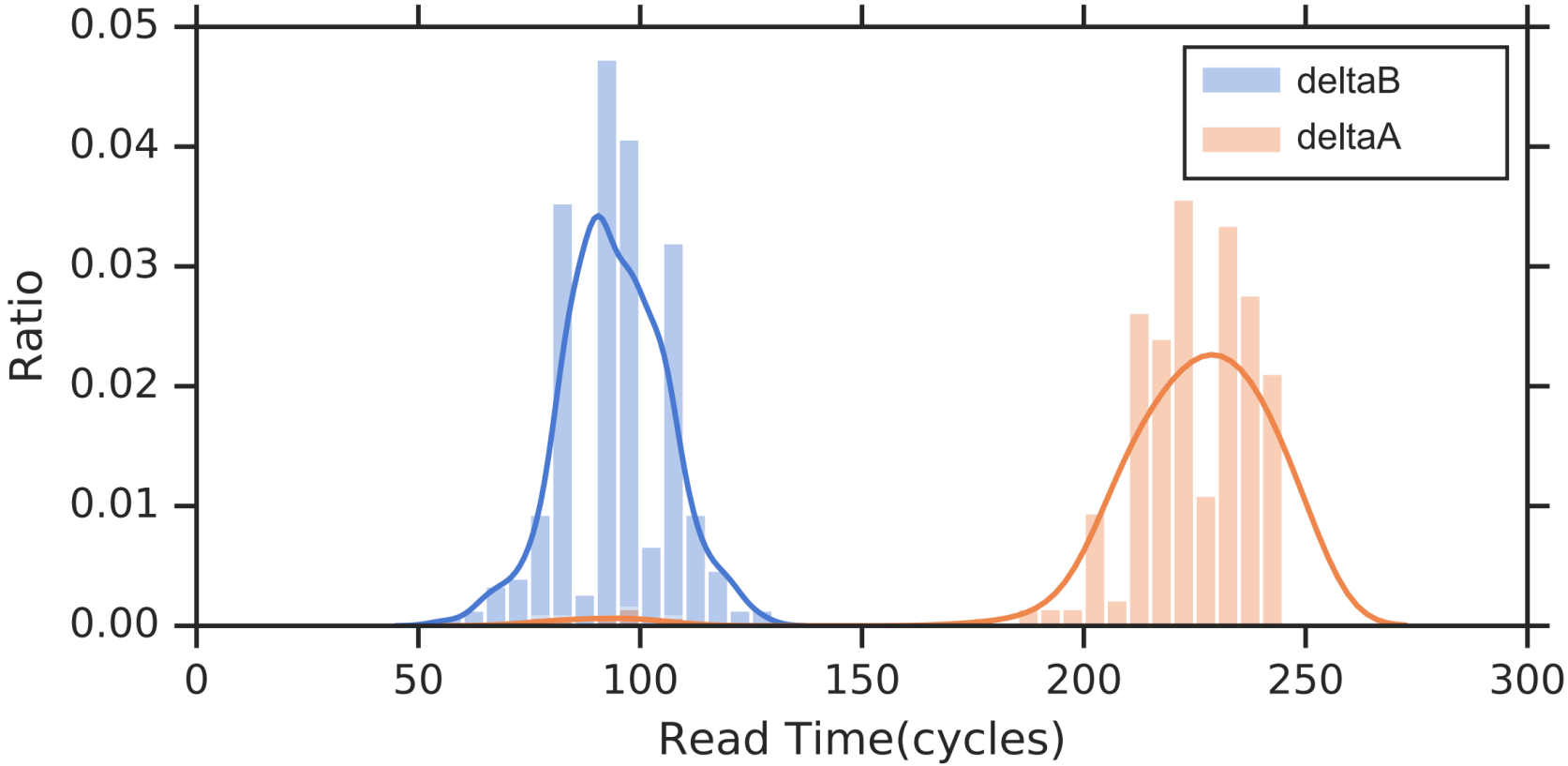


Timing measurement

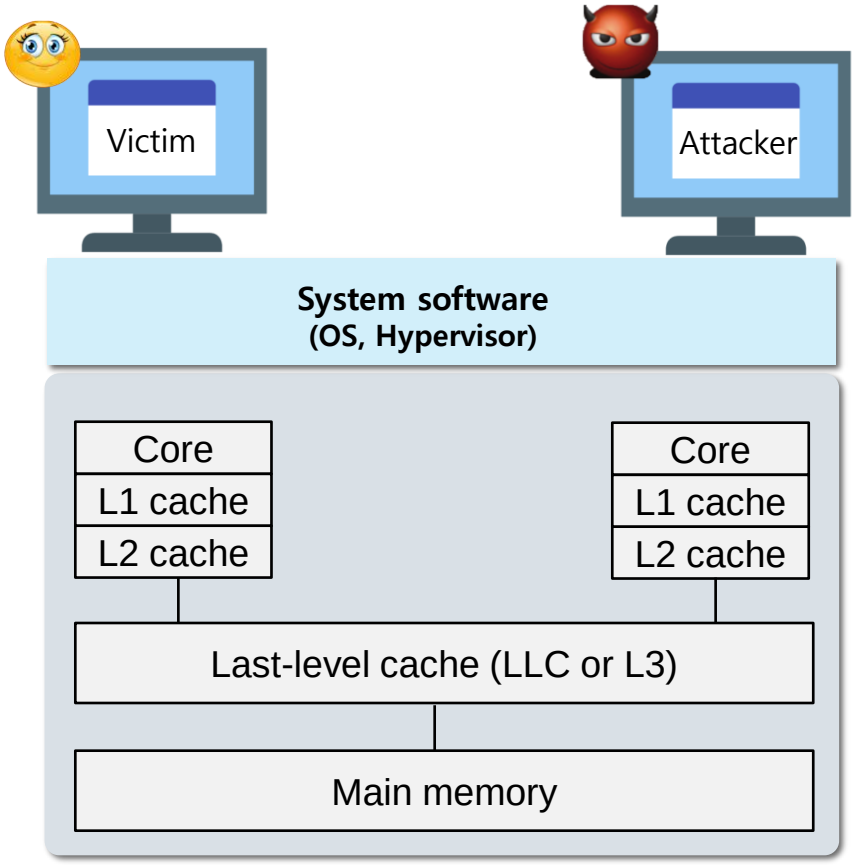
```
time = rdtsc();
maccess(&addressA);
deltaA = rdtsc() - time;
```

```
time = rdtsc();
maccess(&addressB);
deltaB = rdtsc() - time;
```

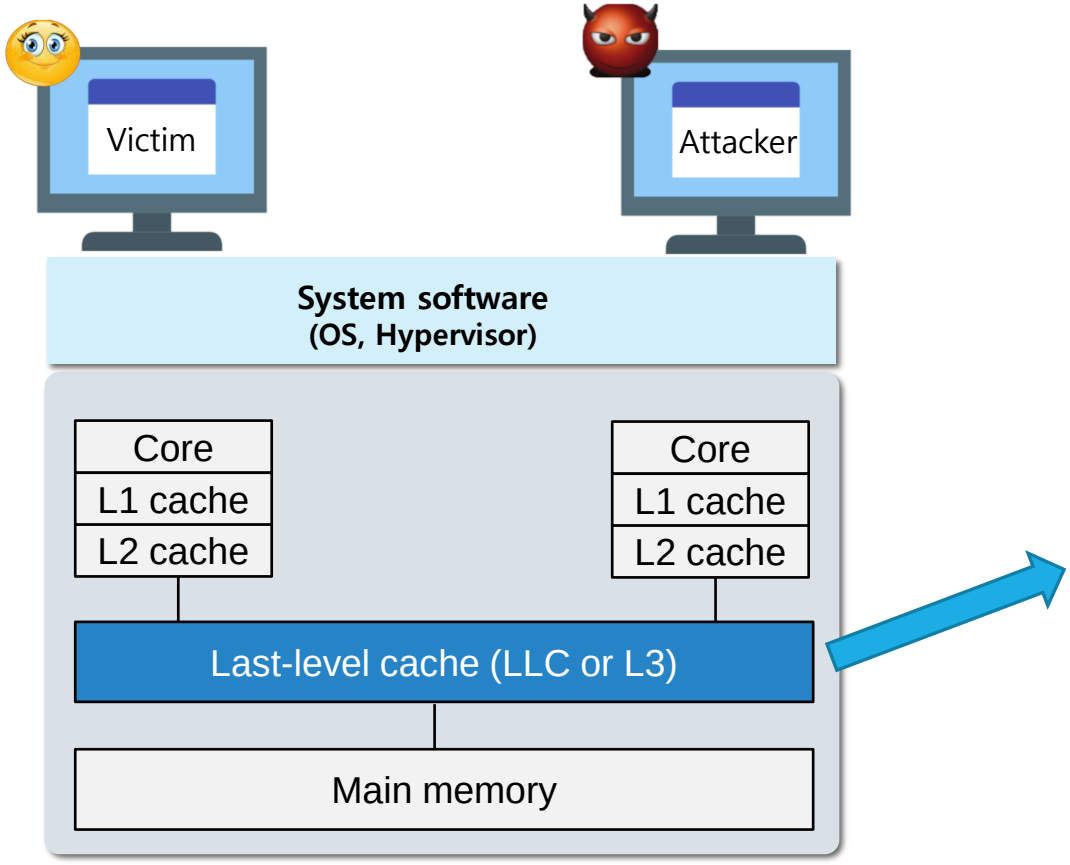
Timing measurement

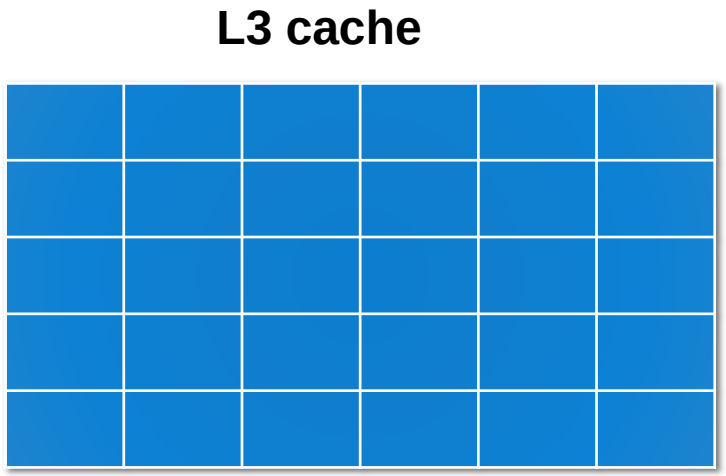


CPU 캐시

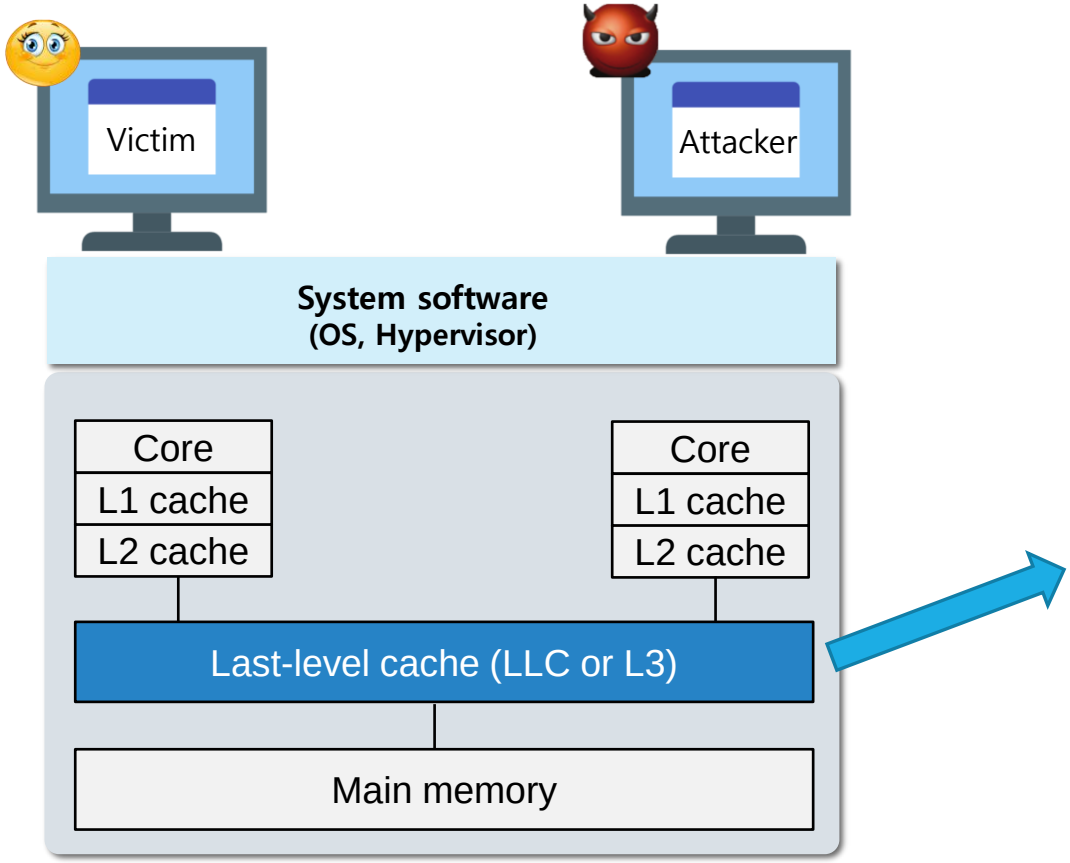

Victim
Attacker
System software
(OS, Hypervisor)
Core
L1 cache
L2 cache
Core
L1 cache
L2 cache
Last-level cache (LLC or L3)
Main memory

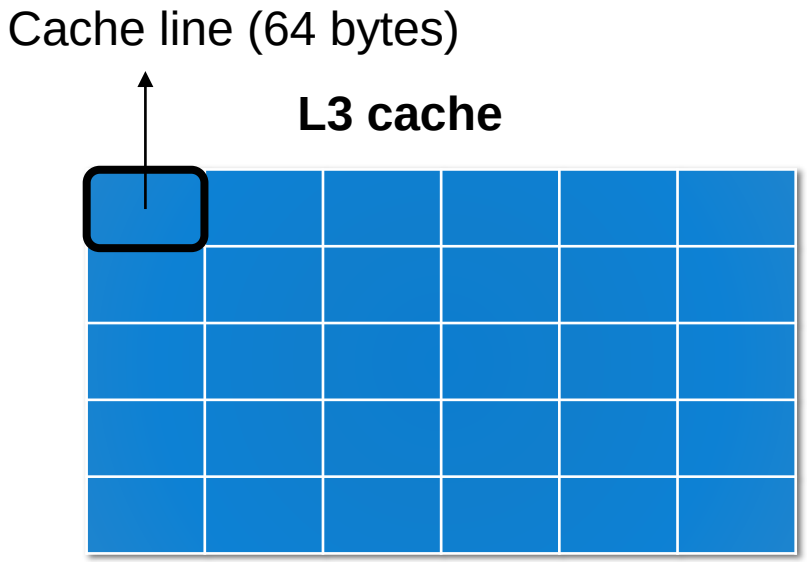
캐시 구조


Victim
Attacker
System software
(OS, Hypervisor)
Core
L1 cache
L2 cache
Core
L1 cache
L2 cache
Last-level cache (LLC or L3)
Main memory

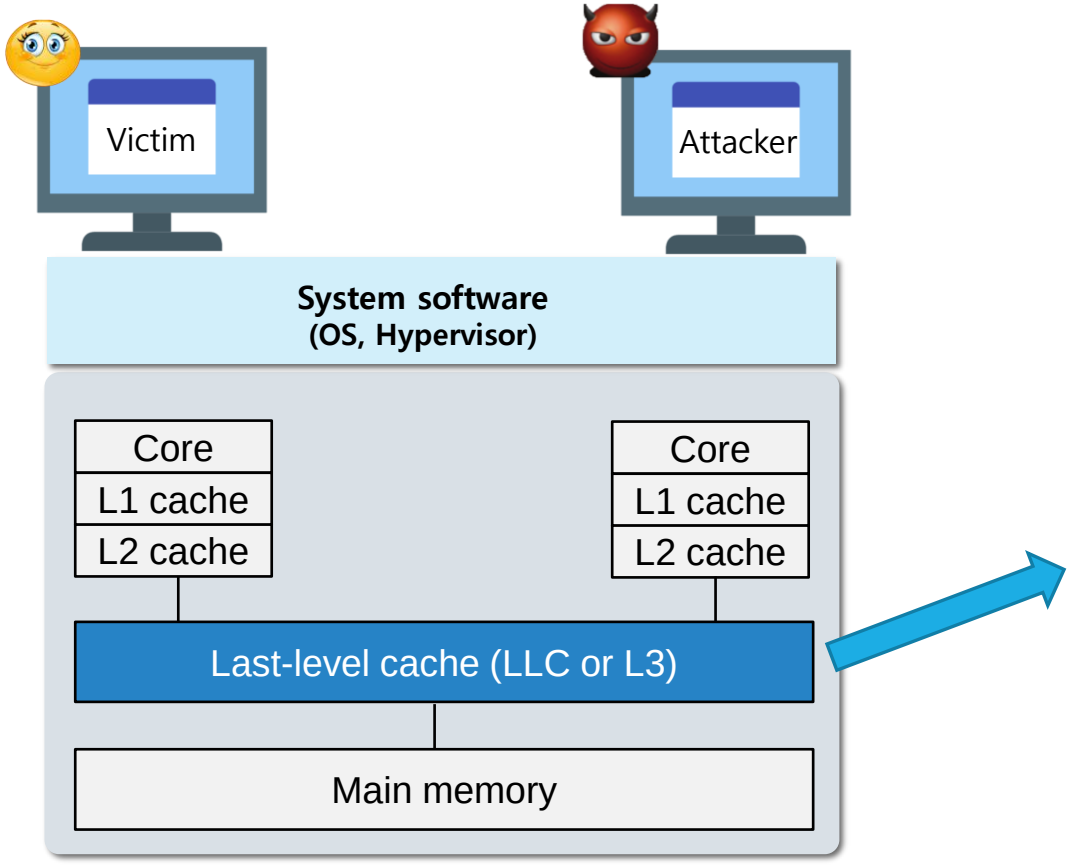

L3 cache

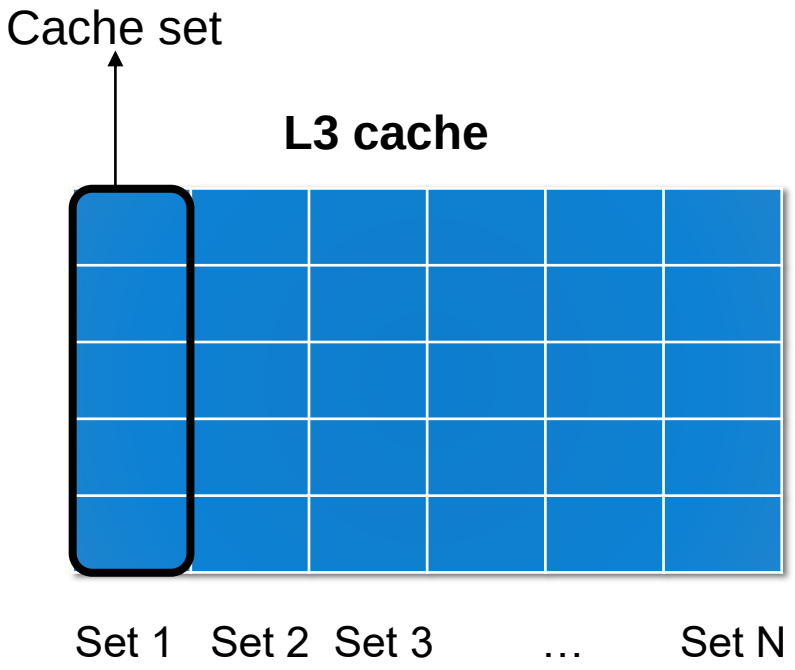
캐시 구조


Victim
Attacker
System software
(OS, Hypervisor)
Core
L1 cache
L2 cache
Core
L1 cache
L2 cache
Last-level cache (LLC or L3)
Main memory

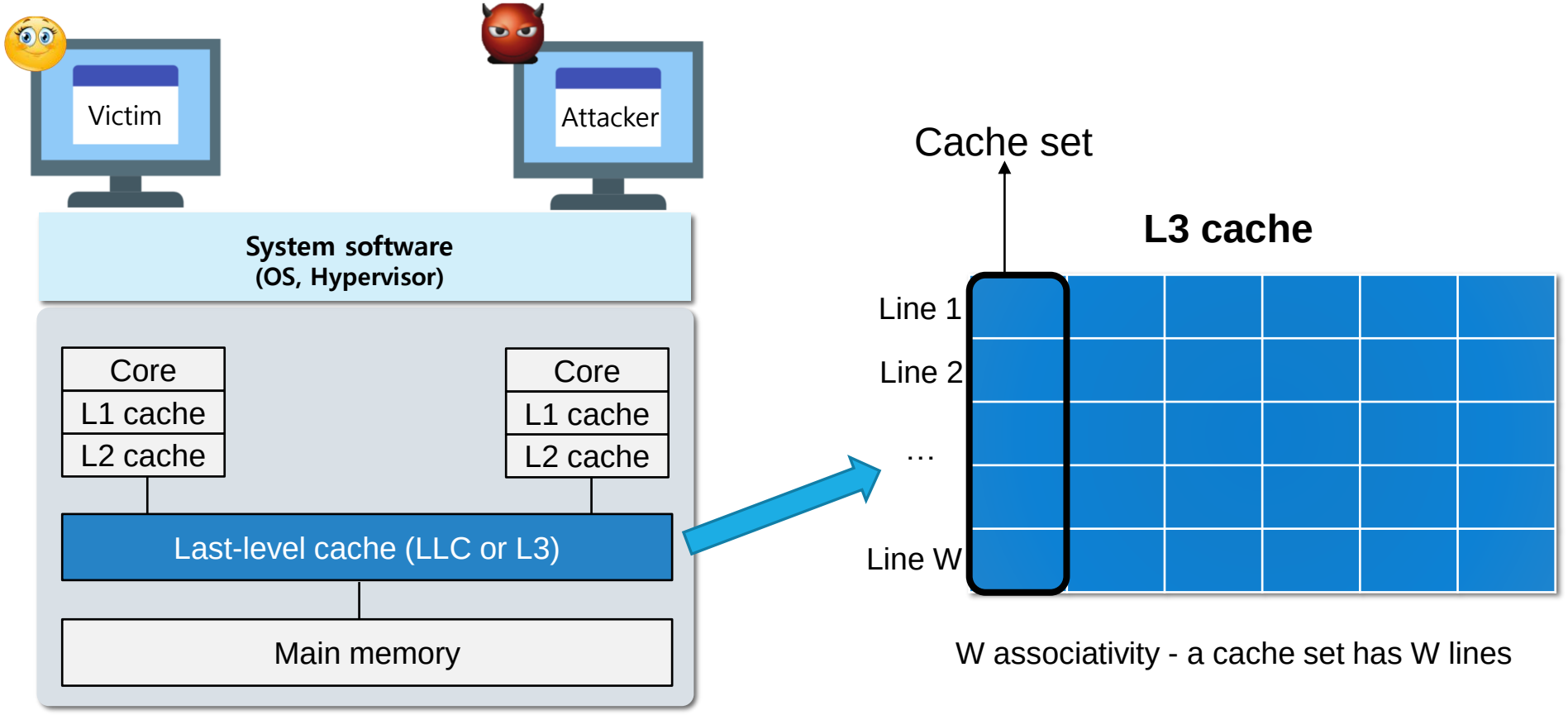

Cache line (64 bytes)
L3 cache

캐시 구조


Victim
Attacker
System software
(OS, Hypervisor)
Core
L1 cache
L2 cache
Core
L1 cache
L2 cache
Last-level cache (LLC or L3)
Main memory

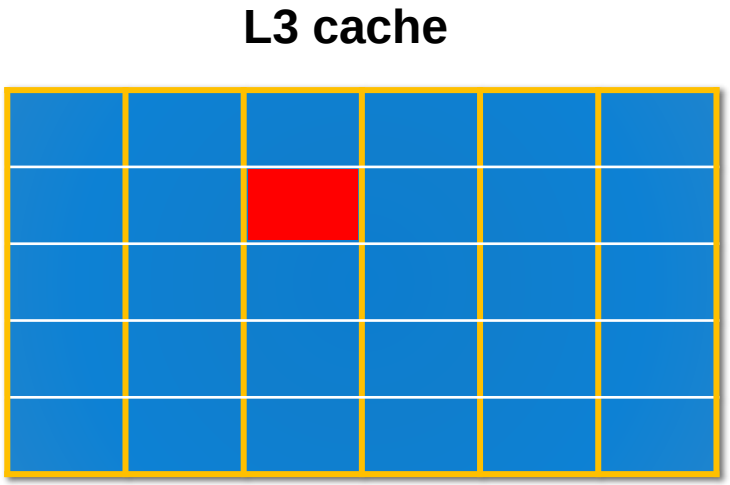

Cache set
L3 cache
Set 1
Set 2
Set 3
…
Set N

캐시 구조


Victim
Attacker
System software
(OS, Hypervisor)
Core
L1 cache
L2 cache
Core
L1 cache
L2 cache
Last-level cache (LLC or L3)
Main memory
Cache set
L3 cache
Line 1
Line 2
…
Line W
W associativity - a cache set has W lines

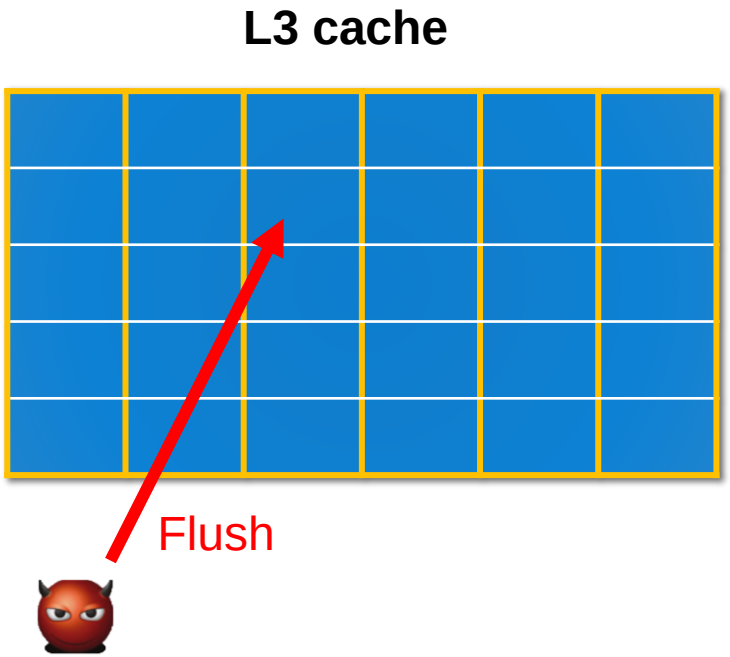
Flush+Reload

- Attacker flushes a target cache line



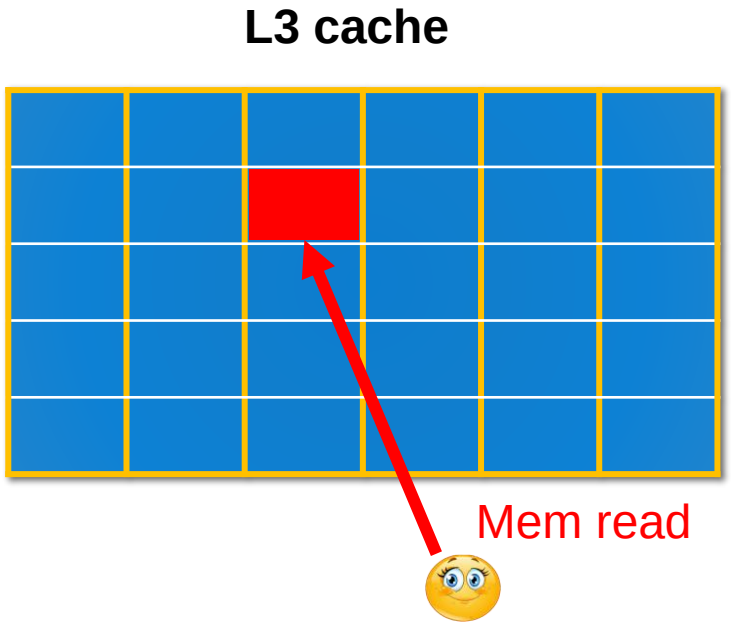
Flush+Reload

- Attacker flushes a target cache line



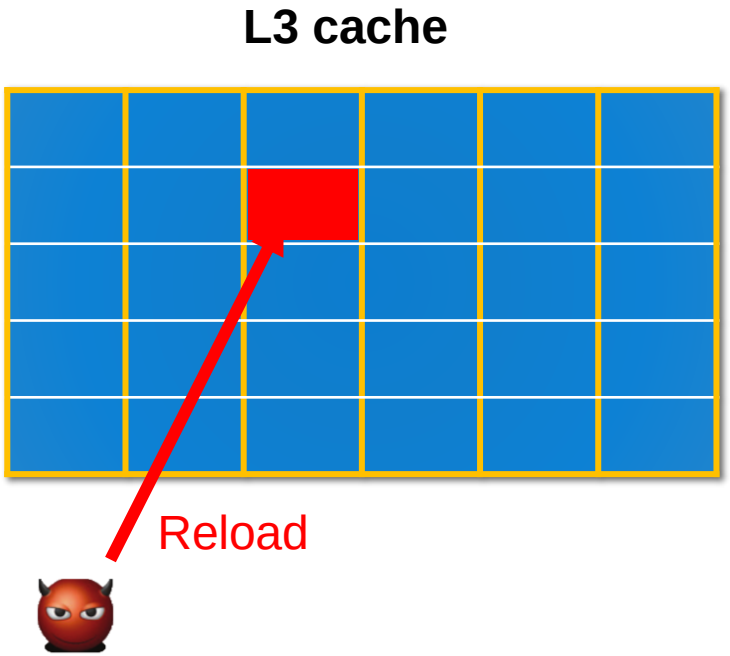
Flush+Reload

- Attacker flushes a target cache line
- Victim accesses/does not access (depending on secret)



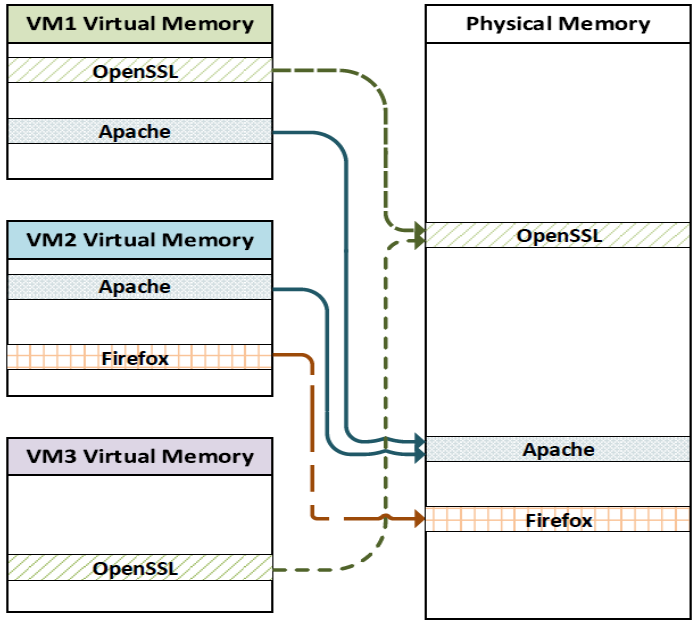
Flush+Reload

- Attacker flushes a target cache line
- Victim accesses/does not access depending on secret
- Attacker re-accesses (reload) memory
  - Fast access time → victim accessed
  - Slow access time → victim did not



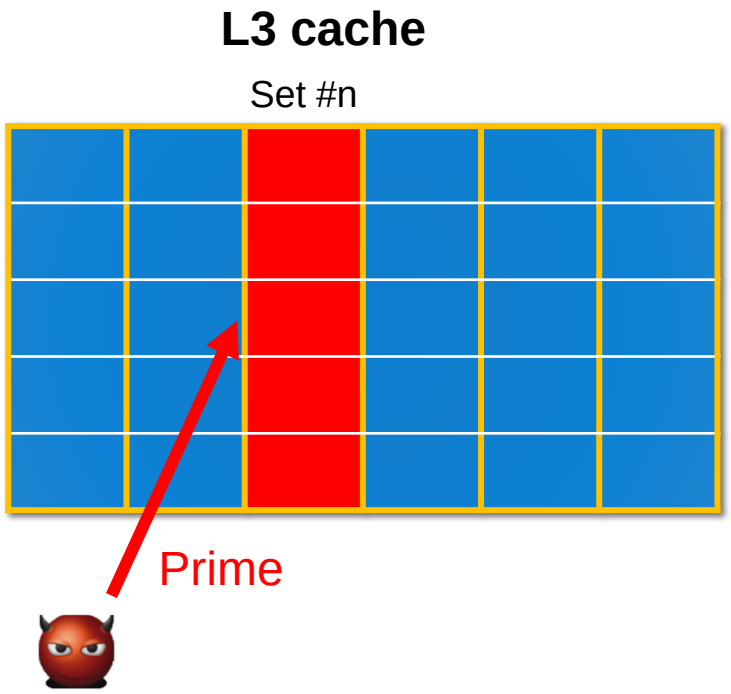
Flush+Reload

- Prerequisite
  - Memory sharing between attacker and victim required
- Memory sharing among virtual machines
  - Hypervisor supports memory deduplication
  - E.g., TPS (Transparent Page Sharing) in VMWare, KSM (Kernel Samepage Merging) in KVM



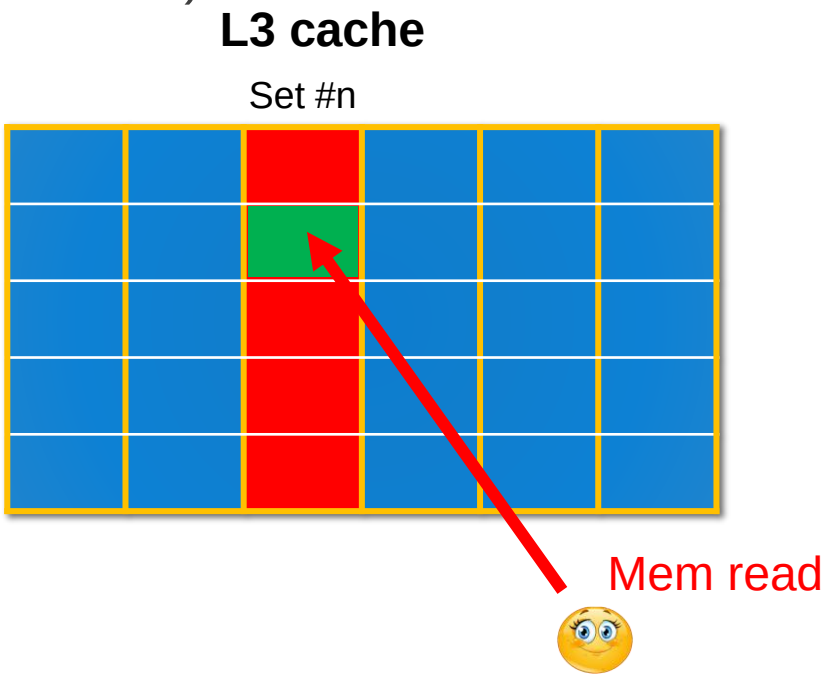
Prime+Probe

- Attacker primes



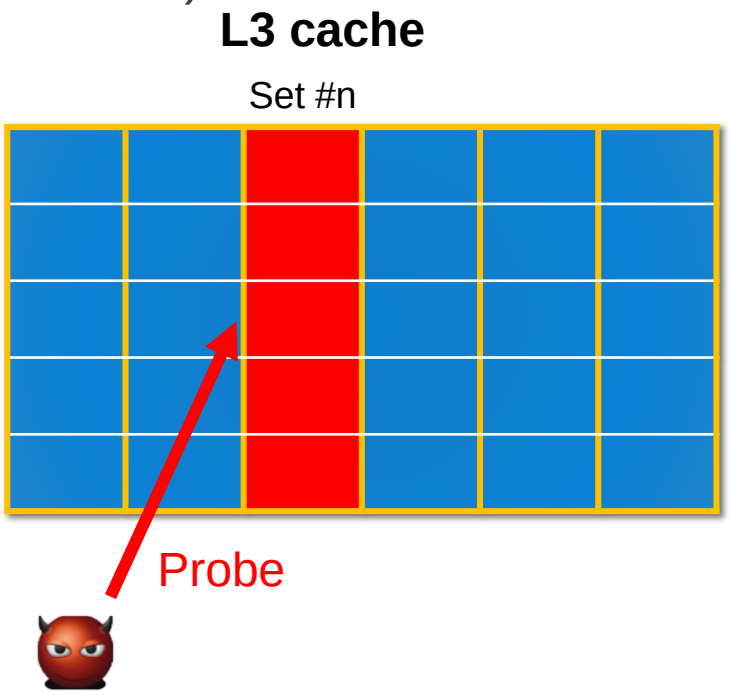
Prime+Probe

- Attacker primes
- Victim accesses/not accesses (depending on secret)



Prime+Probe

- Attacker primes
- Victim accesses/not accesses (depending on secret)
- Attacker re-access (probe)
  - Slow access time → victim accessed
  - Fast access time → victim did not



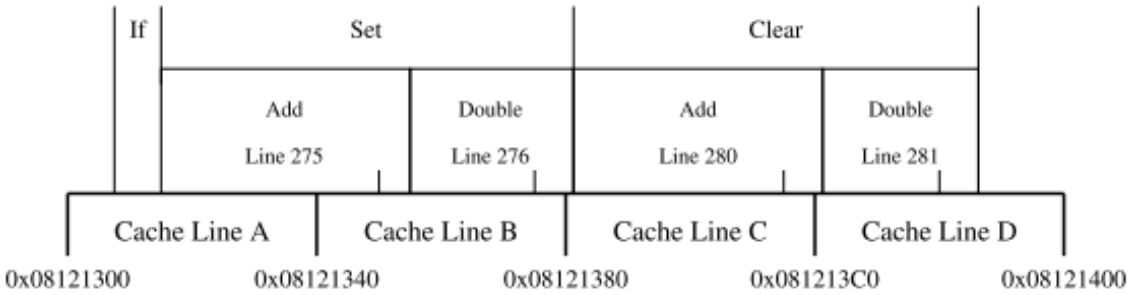
Comparison

Flush+Reload	Prime+Probe
High resolution (500 cycles) Low noise	Low resolution (> 5,000 cycles) High noise
Memory sharing required	Memory sharing not necessary

Attack on ECDSA

- Montgomery ladder multiplication in OpenSSL 1.0.1e [YN14]

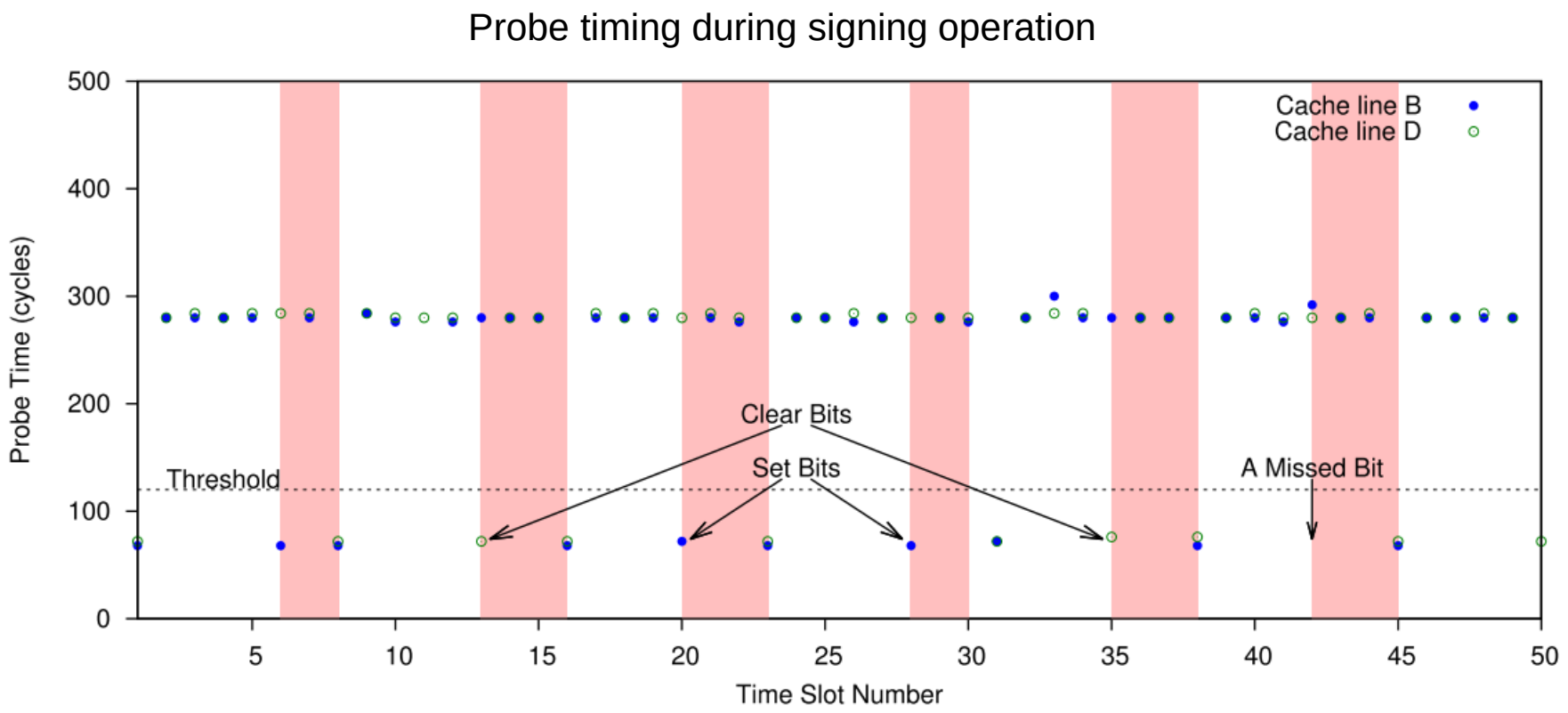
```
268 for (; i >= 0; i--)
269 {
270     word = scalar->d[i];
271     while (mask)
272     {
273         if (word & mask)
274         {
275             if (!gf2m_Madd(group, &point->X, x1,
                        z1, x2, z2, ctx)) goto
                        err;
276             if (!gf2m_Mdouble(group, x2, z2, ctx)
                        ) goto err;
277         }
278         else
279         {
280             if (!gf2m_Madd(group, &point->X, x2,
                        z2, x1, z1, ctx)) goto
                        err;
281             if (!gf2m_Mdouble(group, x1, z1, ctx)
                        ) goto err;
282         }
283         mask >>= 1;
284     }
285     mask = BN_TBIT;
286 }
```



[YN14] Yarom, Y., & Naomi, B.. Recovering OpenSSL ECDSA Nonces Using the FLUSH+RELOAD Cache Side-channel Attack. ePrint archive 2014

Attack on ECDSA

- Montgomery ladder multiplication in OpenSSL 1.0.1e [YN14]



[YN14] Yarom, Y., & Naomi, B.. Recovering OpenSSL ECDSA Nonces Using the FLUSH+RELOAD Cache Side-channel Attack. ePrint archive 2014

자, 그렇다면 대응 방법은?

Crypto Library **개발자들의** CSA **대응 방식**

- 보고된 CSA 취약점에 대한 bug fixing & patch

- Constant time programming 기법의 적용

Constant time programming

- 비밀값Secret 에 관계없이 항상 일정한 실행패턴을 갖도록 프로그래밍 하는 방식
  - 실행 시간이 항상 동일
  - 메모리 접근 패턴이 항상 동일
  - 실행 흐름이 항상 동일

Non-constant 한 프로그램의 예

Algorithm 1 Montgomery ladder algorithm

Input: x-coordinate X/Z for the point $P(X, Z)$ and a scalar $k > 0$
Output: The affine coordinates of the point $Q = kP$

```
1:  procedure MONTGOMERRY_LADDER(k, X, Z)
2:      (k_{l-1}, ..., k_1, k_0) ← k            ▷ k_i ∈ {0, 1} (0 ≤ i ≤ l − 1)
3:      X_1 ← x, Z_1 ← 1, X_2 ← x^4 + b, Z_2 ← x^2   ▷ b is a constant
4:      for i ← l − 2 to 0 do
5:          if k_i = 1 then
6:              MADD(X_1, Z_1, X_2, Z_2), MDOUBLE(X_2, Z_2)
7:          else
8:              MADD(X_2, Z_2, X_1, Z_1), MDOUBLE(X_1, Z_1)
9:          end if
10:     end for
11:     return Q = MXY(X_1, Z_1, X_2, Z_2)   ▷ Transform to an affine
    coordinate
12: end procedure
```

Constant time 한 프로그램의 예

Algorithm 2 Branchless Montgomery ladder algorithm

Input: x-coordinate X/Z for the point $P(X, Z)$ and a scalar $k > 0$
Output: The affine coordinates of the point $Q = kP$

1: **procedure** MONTGOMERRY_LADDER(k, P)
2: $(k_{l-1}, \ldots, k_1, k_0) \leftarrow k$
3: $X_1 \leftarrow x, Z_1 \leftarrow 1, X_2 \leftarrow x^4 + b, Z_2 \leftarrow x^2$
4: **for** $i \leftarrow l - 2$ **to** 0 **do**
5: $\beta \leftarrow k_i$
6: CONST_SWAP(X_1, X_2, β), CONST_SWAP(Z_1, Z_2, β)
7: MADD(X_2, Z_2, X_1, Z_1), MDOUBLE(X_1, Z_1)
8: CONST_SWAP(X_1, X_2, β), CONST_SWAP(Z_1, Z_2, β)
9: **end for**
10: **return** Q = MXY(X_1, Z_1, X_2, Z_2)
11: **end procedure**

현재 CSA 대응 방식의 한계

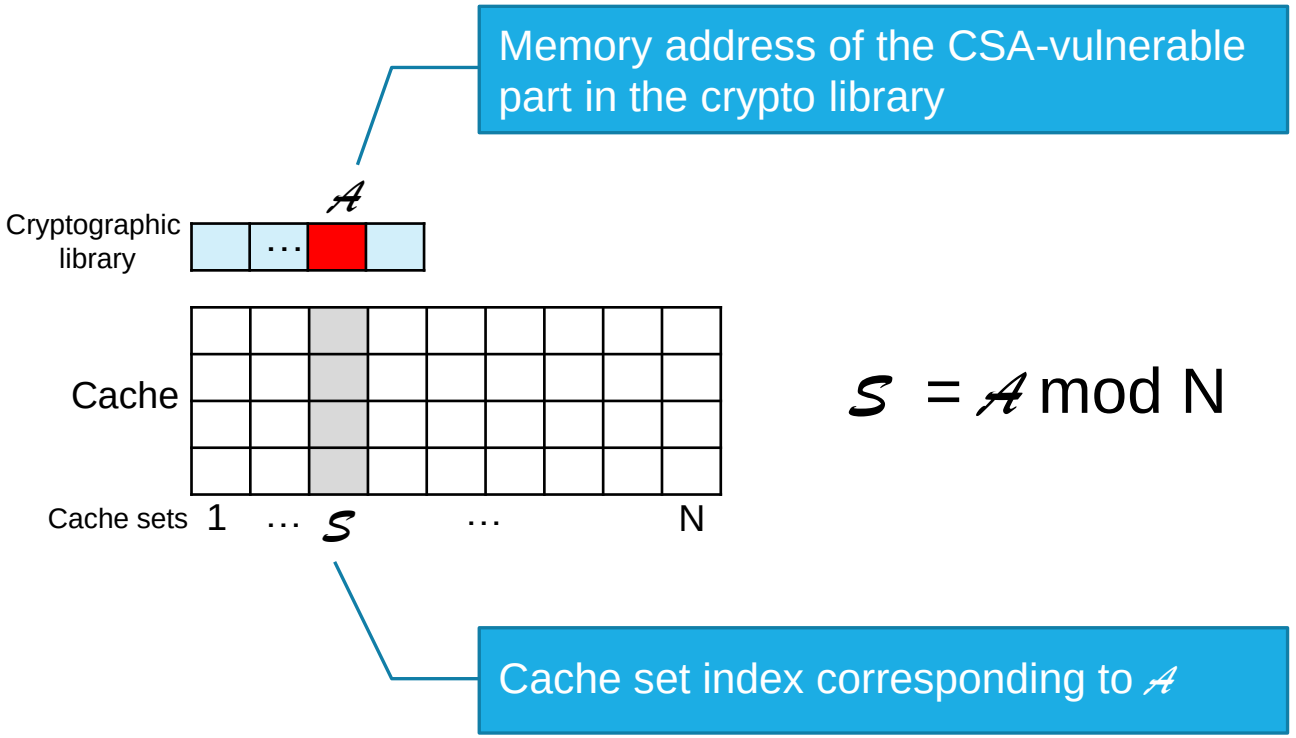
- 구현이 어려움, 전문적인 지식이 요구됨
- 모든 캐시 부채널 공격에 안전하다고 장담하지 못함
- 라이브러리 소스코드의 변경이 필요함

라이브러리 개발자들의 수고를 덜어줄 수 없을까?

- 모든 최신 부채널 공격 기법들을 이해하지 않아도 되고,
- Constant time 구현에 통달하지 않아도 되고,
- 이미 개발된 소스코드를 갈아엎지 않아도 되고,
- 취약한 라이브러리를 패치하기 위해 가동중인 서비스를 중단할 필요가 없으면 더욱 좋고..

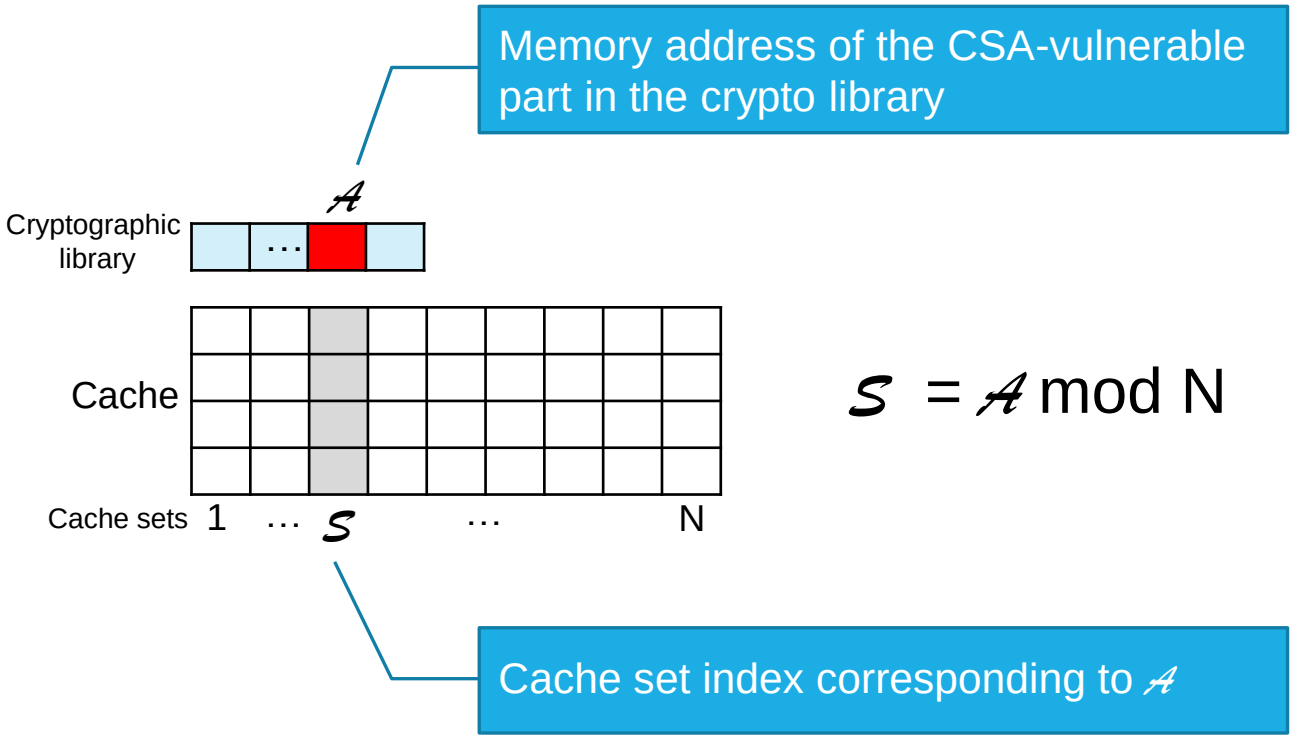
Runtime Randomized Relocation (R2-relocator)

CSA 기본 전략의 이해



$$\mathcal{S} = \mathcal{A} \bmod N \quad \text{or} \quad \mathcal{S} = H(\mathcal{A})$$

CSA 기본 전략의 이해



$$\mathcal{S} = \mathcal{A} \bmod N \quad \text{or} \quad \mathcal{S} = H(\mathcal{A})$$

캐시 부채널 공격을 하기 위해서는 먼저 eviction set 구성에 필요한 S 를 알아내야 함

Address Space Layout Randomization

$$\mathcal{S} = \mathcal{A} \bmod N \quad \text{or} \quad \mathcal{S} = H(\mathcal{A})$$

- S 를 구하기 위해서는 먼저 메모리 주소 A 를 알아내야 함

- 메모리 주소 A 는 ASLR 에 의해 무작위로 결정됨

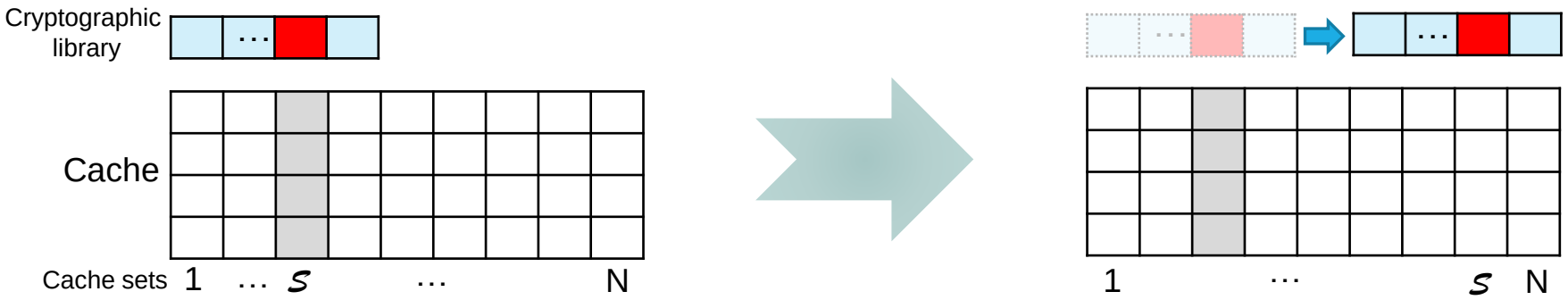
Address Space Layout Randomization

- ASLR 은 기본적으로 로드 타임에만 실행하는 방식

- 즉, 라이브러리가 메모리에 한번 로드된 이후에는 주소 A 는 라이브러리 실행 동안 항상 고정

- 공격자는 일단 A 를 알아내기만 하면 이후에는 수월하게 공격할 수 있음

Runtime randomized relocation



- 프로그램이 실행중에도 ASLR 이 작동하는 방식
- 일정 시간 간격마다 ASLR 에 의해 라이브러리 무작위 재배치 실행
- Eviction set 의 위치 S 가 계속 변경되므로 원활한 공격 수행 불가
- 런타임 재배치 간격이 짧아질 수록 공격 난이도 증가

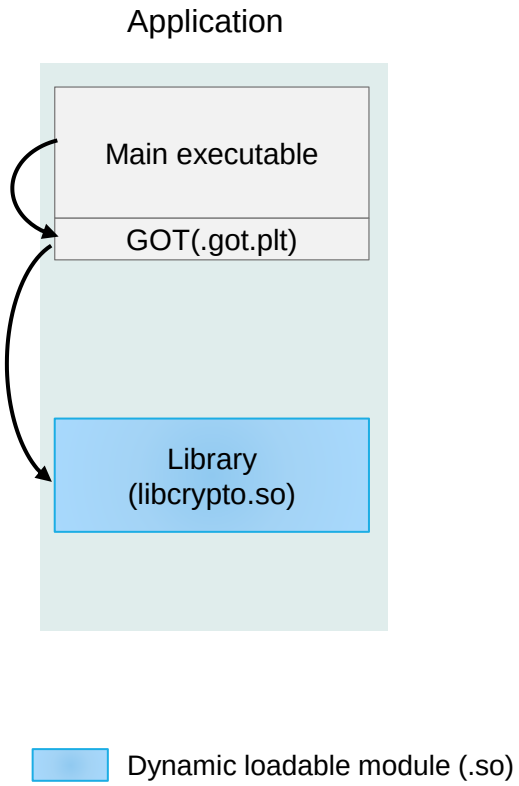
Challenges

- 정적 재배치Static relocation 를 동적 재배치로 전환하는데 따르는 각종 문제..
  - Broken 포인터
  - 동기화 문제 등

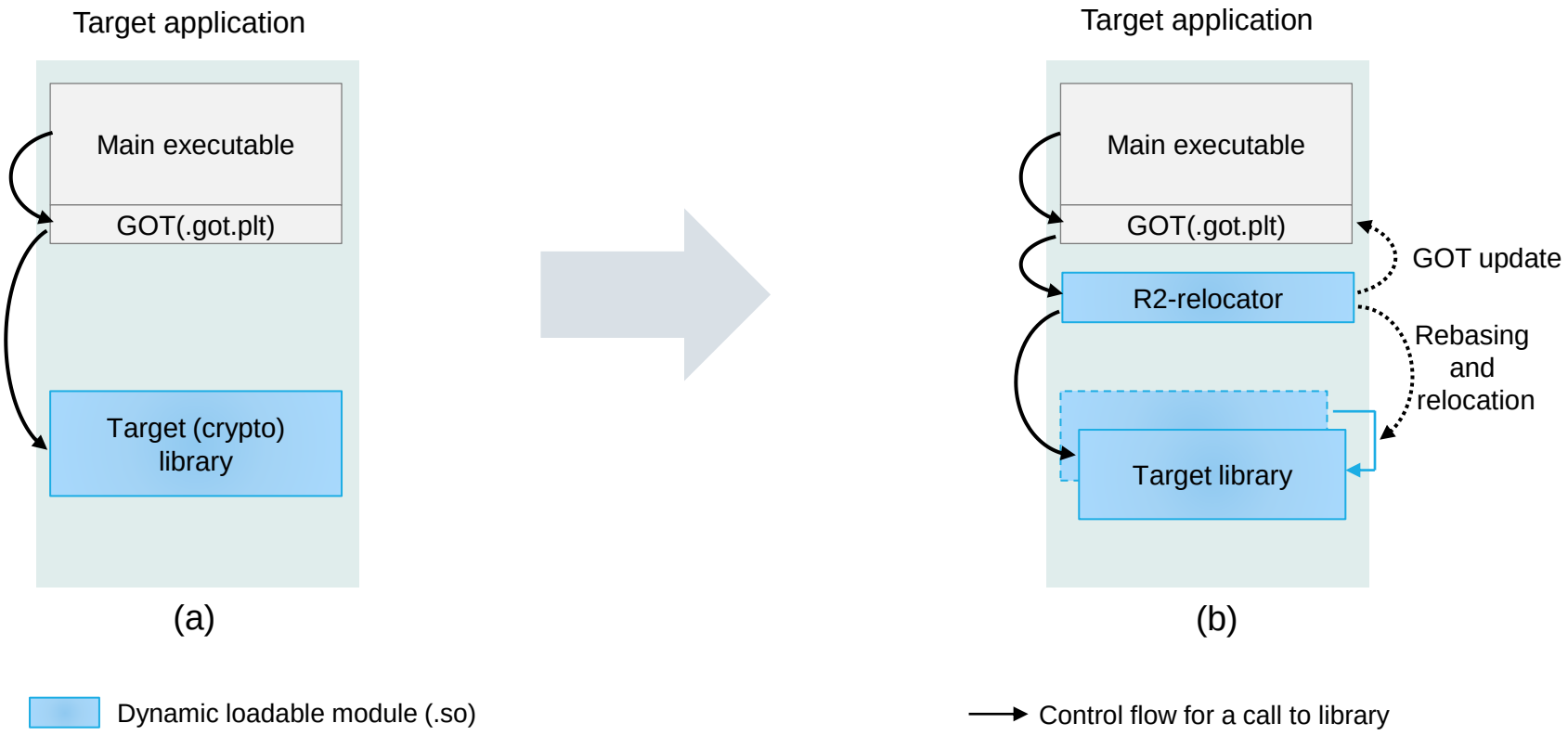
- 애플리케이션 또는 라이브러리의 소스코드 변경없이 어떻게?

정적 재배치 → 동적 재배치

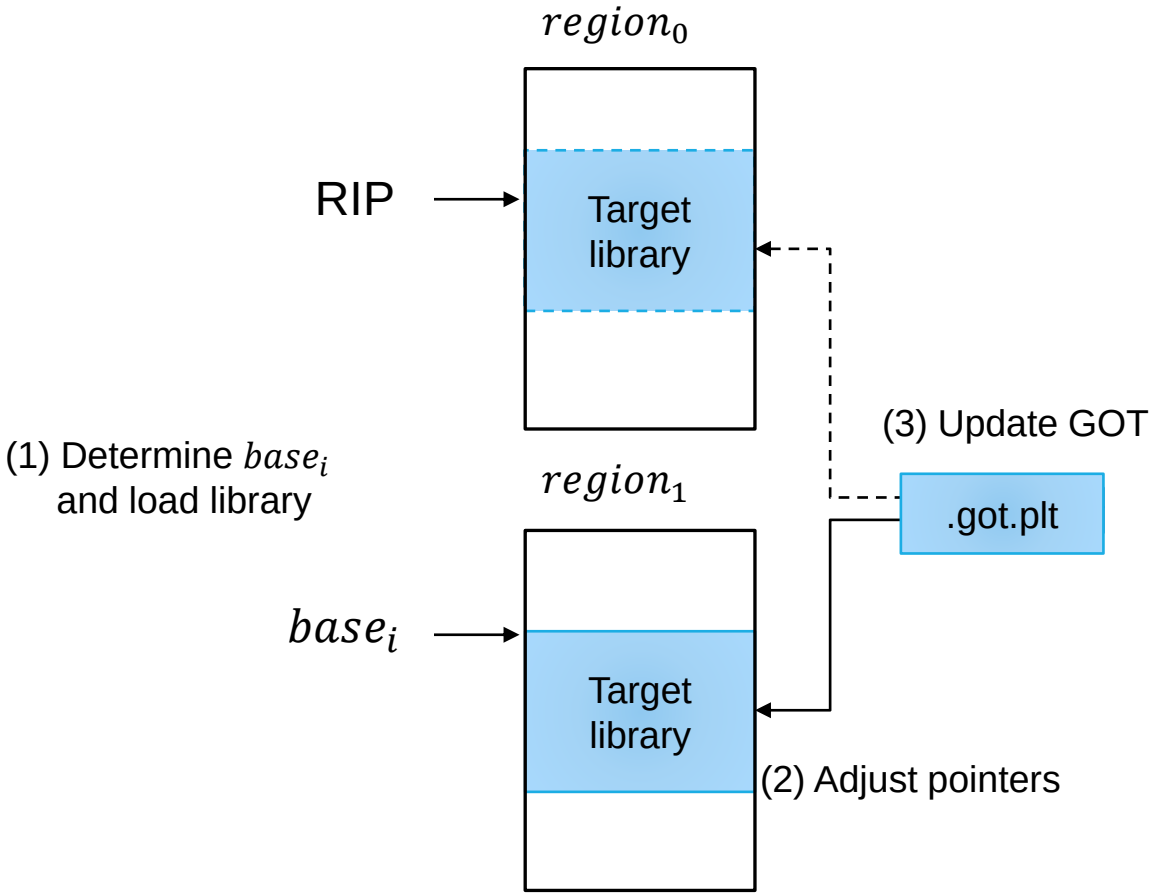
ELF 실행파일 구조



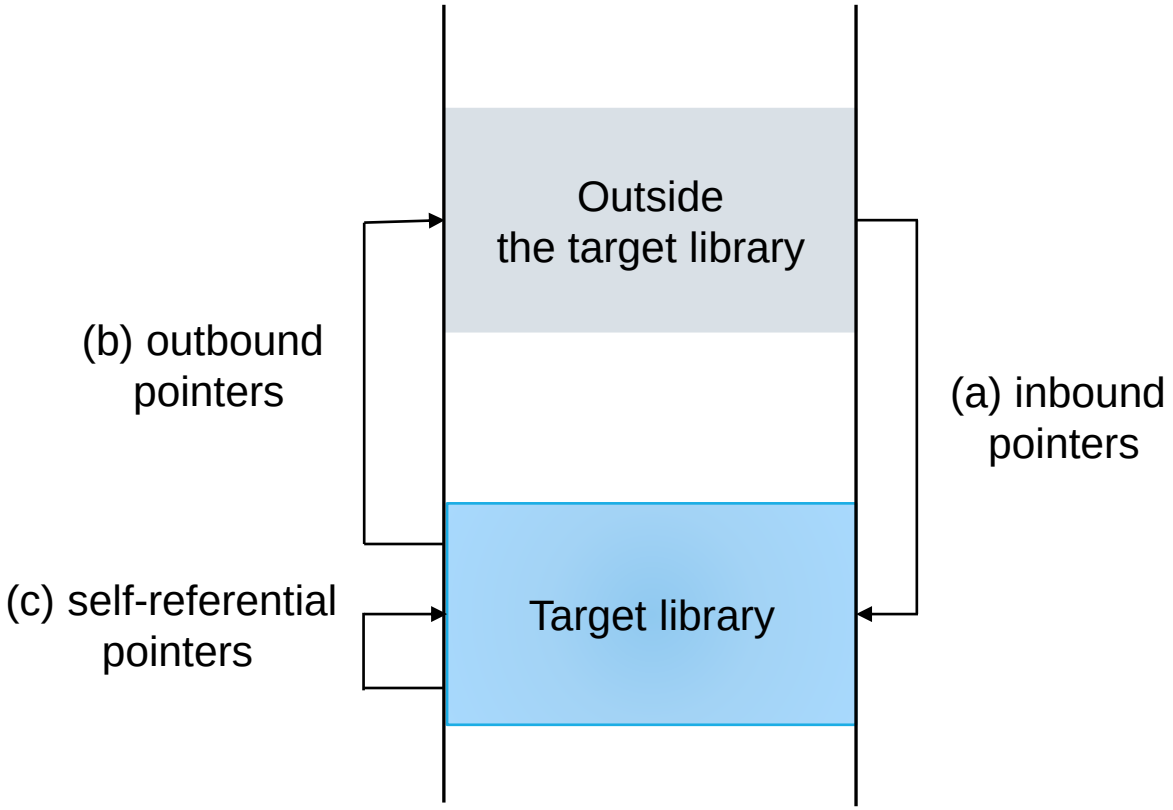
정적 재배치 → 동적 재배치


Target application
Main executable
GOT(.got.plt)
Target (crypto) library
(a)
Target application
Main executable
GOT(.got.plt)
R2-relocator
GOT update
Rebasing and relocation
Target library
(b)
Dynamic loadable module (.so)
Control flow for a call to library

정적 재배치 → 동적 재배치


$region_0$
RIP
Target library
(3) Update GOT
(1) Determine $base_i$ and load library
$region_1$
.got.plt
$base_i$
Target library
(2) Adjust pointers

정적 재배치 → 동적 재배치

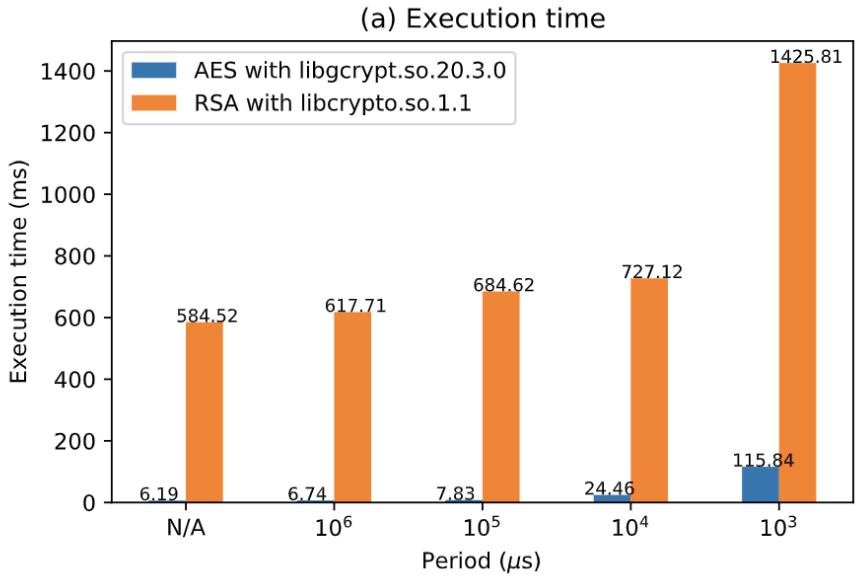

Outside
the target library
(b) outbound
pointers
(a) inbound
pointers
(c) self-referential
pointers
Target library

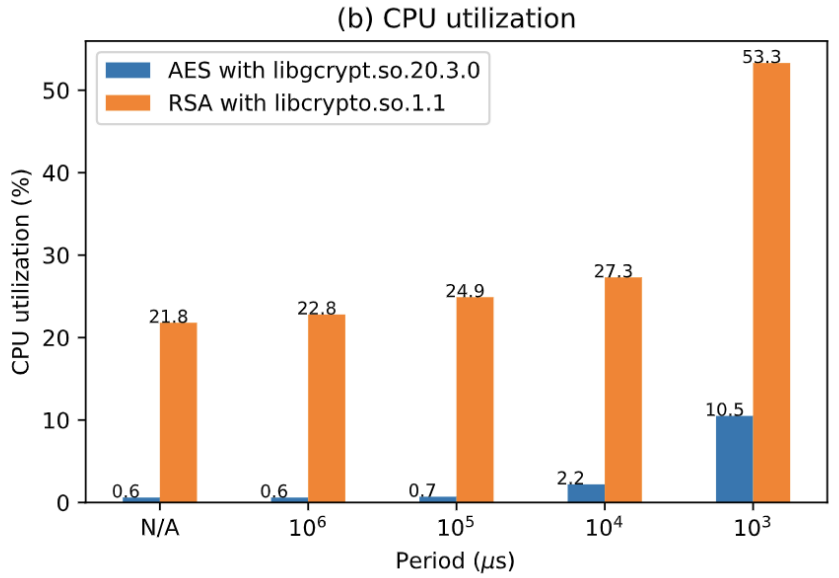
바이너리 변환

- Binary rewriting
  - .ctor in ELF format

- Runtime thread injection
  - Linux **ptrace**

성능 평가


(a) Execution time
AES with libgcrypt.so.20.3.0
RSA with libcrypto.so.1.1
Execution time (ms)
0
200
400
600
800
1000
1200
1400
584.52
617.71
684.62
727.12
1425.81
6.19
6.74
7.83
24.46
115.84
N/A
10^6
10^5
10^4
10^3
Period (μs)


(b) CPU utilization
AES with libgcrypt.so.20.3.0
RSA with libcrypto.so.1.1
CPU utilization (%)
0
10
20
30
40
50
21.8
22.8
24.9
27.3
53.3
0.6
0.6
0.7
2.2
10.5
N/A
10^6
10^5
10^4
10^3
Period (μs)

성능 평가

Secret length (λ)	Period = $4 \cdot 10^4 \mu s$			Period = $2 \cdot 10^4 \mu s$			Period = $10^4 \mu s$		
	T_{L1} (min)	T_{LLC} (min)	Ratio (T_{L1}/T_{LLC})	T_{L1} (min)	T_{LLC} (min)	Ratio (T_{L1}/T_{LLC})	T_{L1} (min)	T_{LLC} (min)	Ratio (T_{L1}/T_{LLC})
4	105	210	0.50	126	252	0.50	220	423	0.52
8	220	420	0.52	264	520	0.51	450	850	0.53
16	550	850	0.65	660	1050	0.63	1210	1820	0.66
32	1320	1790	0.74	1584	2192	0.72	3420	4010	0.85
64	2190	2780	0.79	2820	3336	0.85	5630	6240	0.90

결론

- MTD 전략을 이용한 캐시 부채널 공격 방지 기법
- 라이브러리 개발자에 많은 편의성 제공
- 실행 성능 저하는 해결해야할 문제

Thanks

Q&A