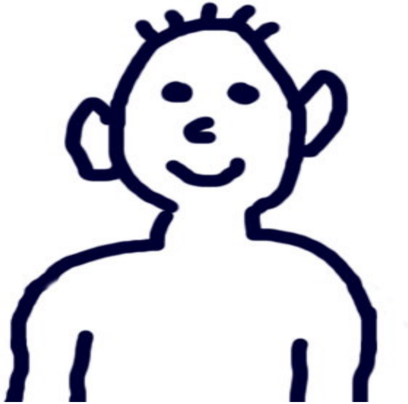
TANA: Precise and Practical Sampling-based Side channel Information Leakage Quantification

Bao Qinkun

What is side-channel attack ?

A side-channel attack is any attack that allows attackers to infer some sensitive information based on non-functional characteristics.



A naïve Ph.D. student

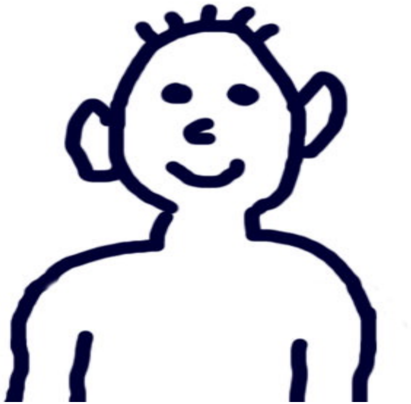
A prestigious professor

Side-channel attack

Secret: whether your advisor is in the office

What you can do:

1. Ask your advisor (Not a good idea)



Attacker

Victim

Side-channel attack

Secret: whether your advisor is in the office

What you can do:

1. Ask your advisor (Not a good idea)

Insight:

Your advisor always drives to work.

The parking lot is shared with you.

Side-channel attack

Secret: whether your advisor is in the office

What you should do:

1. Go to the parking lot
2. Find your advisor's vehicle

Address-based Side Channels

Side-channels in computer systems: timing, power, CPU usage, cache …

```
unsigned char key = input();
// 4 bits information
unsigned char temp = key/2;
if (temp = 0xb)
// branch 0 execution 2s
else
// branch 1 execution 1s
```

Secret-dependent control-flow transfers

```
T[1024];
// Lookup tables with 1024 entries
index = key % 1024;
temp = T[index]
// Secret-dependent memory access
```

Secret-dependent memory accesses

Cache-based: Prime + Probe, Flush + Reload, Flush + Evict
Memory-based: Controlled-channel attack, page-table attack

Side-channel Vulnerability Detections

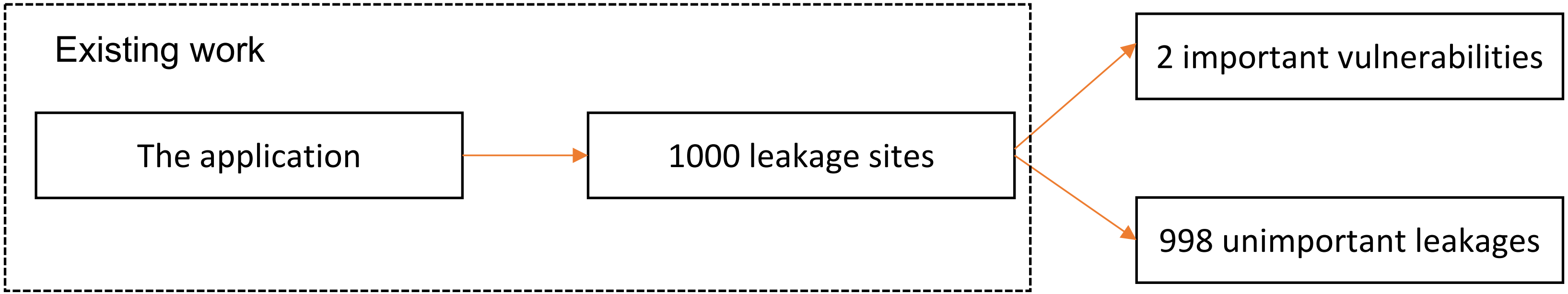
Software countermeasures:

- Find vulnerabilities
  - It's hard. Real cryptosystems are complicated
    - OpenSSL 281K LoC
  - Those vulnerabilities are architecture-dependent
  - Existing works: CacheAudit, CacheD, DATA, CacheS.
- Fix vulnerabilities
  - It's also hard. Side-channel vulnerabilities are inevitable.
  - Many vulnerabilities are unpatched.

Motivation

Existing works:

- False positives
  - e.g., 2248 potential leakages for RSA in OpenSSL, 1510 of them were dismissed
- Report many insensitive leakages
- Only on one leakage site



Our Objective: Identify and quantify address-based side-channels precisely.

An example

```c
unsigned char k1, k2;
...
t1 = T[k1 % 8];        // Leakage 1
if(k1 > 127)           // Leakage 2
   A();
if(k2 + k1 > 127)      // Leakage 3
   B();
```

An attacker:

1. knows which element the code read
2. knows if the code runs A(), B()

Leakage	1	2	3	1,2	1,3	2,3
Leaked (bits)	3	1	1	4	4	2

Table 1. Leakage Summary

Overview

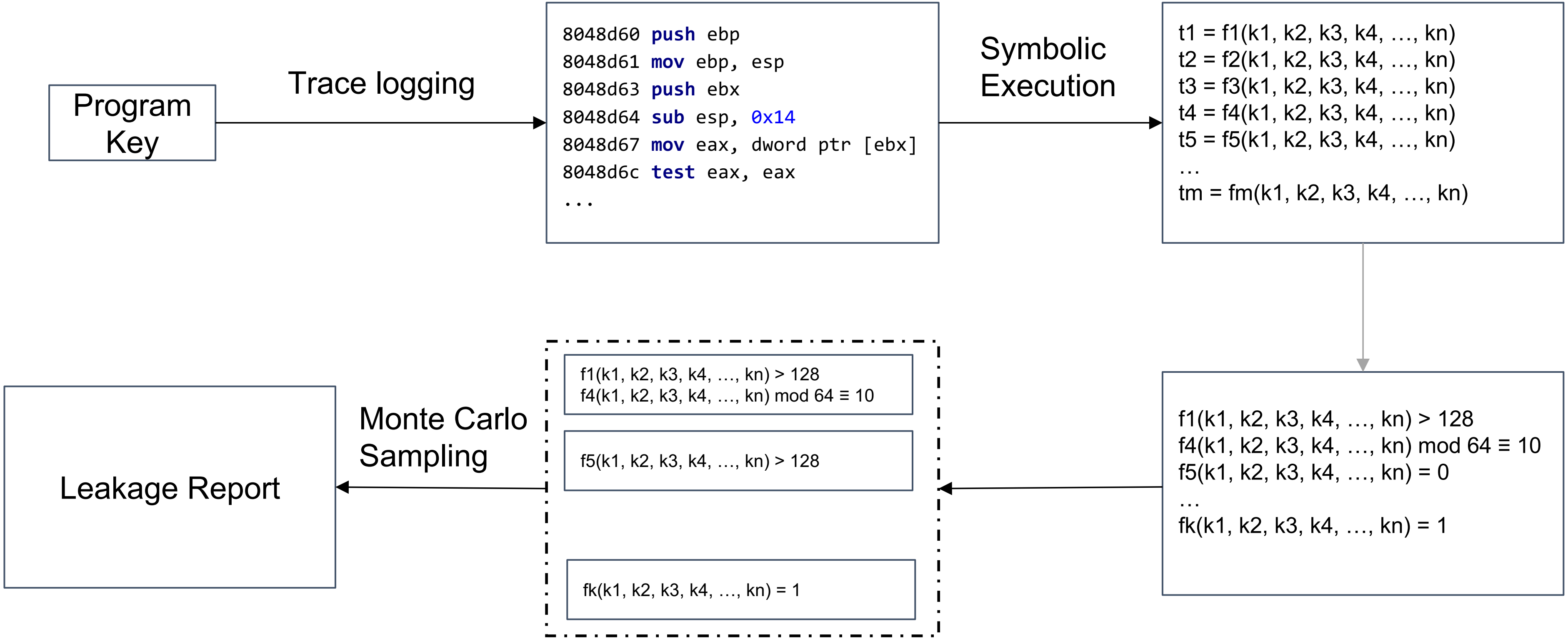


Figure 1. TANA architecture

Challenges

- Information Leakage Definition
  - Existing work: Shannon entropy and mutual information
  - Number of different observations
- Leakage Dependence
  - Real-world applications may have multiple information leakages sites
  - Some leakages are dependent
- Scalability
  - We want to quantify information leakages in real-world applications
  - e.g. RSA implementations OpenSSL have thousands of lines -> 1 million instructions
  - The performance is important

Challenge 1: Information Leakage Definition

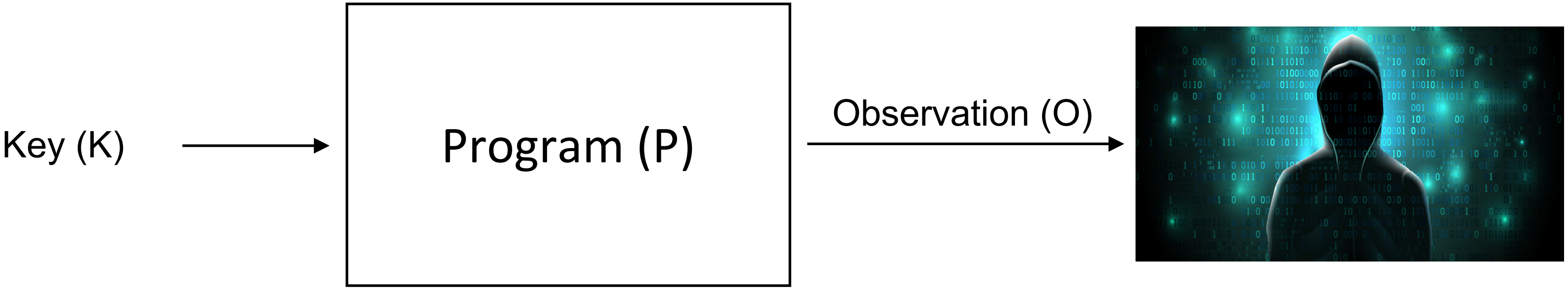


Figure 2. Observation Model

Observations:
1. Secret-dependent control-flow transfers
2. Secret-dependent memory access

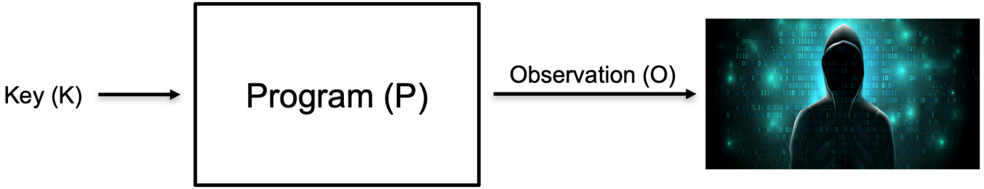
Previous Information Leakage Definition

Mutual Information (MI):

$$\mathrm{MI} = \mathrm{I}(\mathrm{K};\mathrm{O}) = \mathrm{H}(\mathrm{O}) - \mathrm{H}(\mathrm{O}|\mathrm{K}) = \mathrm{H}(\mathrm{O}) = \sum_{\mathrm{o_i} \in \mathrm{O}} \mathrm{P}(\mathrm{o_i})\log_2 \mathrm{P}(\mathrm{o_i})$$

Maximal Leakage (ML):

$$\mathrm{ML} = \log_2 |\mathrm{O}|$$




```
unsigned char key = input();
// key = [0 ... 255]
if(key = 128)
   A(); // branch 1
else if (key < 64)
   B(); // branch 2
else if (key < 128)
   C(); // branch 3
else
   D(); // branch 4
```

Branch	1	2	3	4
Possibility	$\frac{1}{256}$	$\frac{64}{256}$	$\frac{63}{256}$	$\frac{128}{256}$

$$MI = \frac{1}{256}log_2\frac{1}{256} + \frac{64}{256}log_2\frac{64}{256} + \frac{63}{256}log_2\frac{63}{256} + \frac{128}{256}log_2\frac{128}{256} = 1.7bits$$

$$ML = log_2 4 = 2bits$$

Challenge 1: Information Definition

```c
// Dummy password checker
unsigned char key = input();
// key = [0 ... 255]
if(key = 128)
   A(); // branch 1
else if (key < 64)
   B(); // branch 2
else if (key < 128)
   C(); // branch 3
else
   D(); // branch 4
```

Suppose an attacker knows the code run branch 1, then he knows the key equals to 128.

Mutual Information: 1.7 bits
Max Leakage: 2 bits

Problem: Both methods neglect the input key and give an "average" estimate.

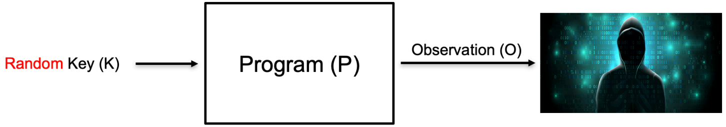


Figure 3. Mutual Information

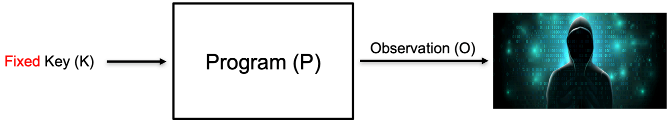


Figure 4. Real Attack

Challenge 1: Information Leakage Definition

Definition. *Given a program P with the input set K, an adversary has the observation o when the input k ∈ K. We denote it as*

$$P(k) = 0$$

The leakage L_{ko} *based on the observation is*

$$L_{ko} = log_2|K| - log_2|K^o|$$

where

$$K_o = \{k' \mid k' \in K \text{ and } P(k') = o\}$$

```
// Dummy password checker
unsigned char key = input();
// key = [0 ... 255]
if(key = 128)
   A(); // branch 1
else if (key < 64)
   B(); // branch 2
else if (key < 128)
   C(); // branch 3
else
   D(); // branch 4
```

Example:

$|K| = 256 \quad |K_1| = 1 \quad |K_2| = 64$

$L_{ko1} = log_2 256 - log_2 1 = 8\ bits$

$L_{ko2} = log_2 256 - log_2 64 = 2\ bits$

Basic idea: counting the number of observations that have the same memory-access pattern.

Challenge 2: Leakage Dependence

- Suppose one program has two leakage sites A and B.
  - Independent Leakage
  - Mutual Exclusive Leakage
  - Dependent Leakage

Leakage A:

$$\begin{cases} 0 \le k_1 \\ k_1 \le 255 \\ 3k_1 + k_2 > 127 \end{cases}$$

Leakage B:

$$\begin{cases} 0 \le k_2 \\ k_2 \le 255 \\ k_1 - 3k_2 > 0 \end{cases}$$

Leakage A and B :

$$\begin{cases} 0 \le k_1 \\ k_1 \le 255 \\ 3k_1 + k_2 > 127 \\ k_1 - 3k_2 > 0 \end{cases}$$

```
unsigned char k1, k2, t1, t2;
t1 = k1 + 2*k2;
t2 = 2*k1 - k2;
if(t2 + t1 > 127)  // Leakage A
    A();
if(t2 - t1 > 0)    // Leakage B
    B();
```

if $L_{(k1k2)A} + L_{(k1+k2)B} = L_{(k1+k2)AB}$,
then A and B are independent

Challenge 3: Scalability

We want to quantify information leakages from real-world applications.

The performance suffers from the following aspects:

- Symbolic Execution
  - IR explosion
- Monte Carlo Sampling
  - #P problem

Challenge 3: Scalability (Symbolic Execution)

- Why symbolic execution (SE) is slow?
  - Path Explosion ✔
  - Constraint Solving ✔
  - Intermediate Languages (IR) ?

Why symbolic execution uses IR?

- Support many architecture platform.
  - X86, ARM, MIPS
- Easy to implement
  - X86, more than 1000 instructions
  - Side effects

test eax, eax →

```
STR  R_EAX:32,       , V_00:32
STR  0:1,            , R_CF:1
AND  V_00:32,  ff:8, V_01:8
SHR  V_01:8,    7:8, V_02:8
SHR  V_01:8,    6:8, V_03:8
XOR  V_02:8, V_03:8, V_04:8
SHR  V_01:8,    5:8, V_05:8
SHR  V_01:8,    4:8, V_06:8
XOR  V_05:8, V_06:8, V_07:8
XOR  V_04:8, V_07:8, V_08:8
SHR  V_01:8,    3:8, V_09:8
SHR  V_01:8,    2:8, V_10:8
XOR  V_09:8, V_10:8, V_11:8
SHR  V_01:8,    1:8, V_12:8
XOR  V_12:8, V_01:8, V_13:8
XOR  V_11:8, V_13:8, V_14:8
XOR  V_08:8, V_14:8, V_15:8
AND  V_15:8,    1:1, V_16:1
NOT  V_16:1,         , R_PF:1
STR     0:1,         , R_AF:1
EQ  V_00:32,   0:32, R_ZF:1
SHR V_00:32,  1f:32, V_17:32
AND    1:32,V_17:32, V_18:32
EQ     1:32,V_18:32, R_SF:1
STR     0:1,         , R_OF:1
```

QSYM: A Practical Concolic Execution Engine Tailored for Hybrid Fuzzing. Insu Yun, Sangho Lee, Meng Xu, Yeongjin Jang, and Taesoo Kim.In Proceedings of the 27th USENIX Security Symposium (Security 2018),Baltimore, MD, August 2018.

Solution: instruction-level symbolic execution

Challenge 3: Scalability (Monte Carlo Sampling)

- The problem is #P-Hard.
- The number of satisfying assignments might be exponentially small.
  - e.g., $F(K) = \begin{cases} k_1 = 1 \\ 120 < k_2 < 123 \\ k_3 = 3 \\ k_4 = 4 \\ k_5 = 5 \end{cases}$ $\quad k_1, k_2, k_3, k_4, k_5 \in [0, 255]$

Total search space: 2^{40}
Only two satisfying assignments

Solution: Markov Chain Monte Carlo Sampling (MCMC)
Chernoff bound: $N \geq \Theta(\frac{log\delta}{p\varepsilon^2})$ to get an (δ, ε) approximation scheme

Implementation

- 15k LoC of C++11
- Trace Collection
    - Intel Pin Tool
- Symbolic Execution
    - arithmetic, bitwise, logical, control-transfer
    - Not support: AVX, floating number, AES-NI
- Information Leakage Computation

Evaluation (Not finished)

- Real-world cryptosystems:
  - OpenSSL 0.9.7 1.1.1
  - mbedTLS 2.5 2.15
  - Libjpeg
- Performance
  - CacheD 5 hours
  - 30s found 8 secret-dependent memory access for the DES in mbedTLS 2.5

Summary

Identify and quantify address-based side-channels leakages precisely

- A trace-based method that models each leakage site with math formulas.
- We quantify the information leakages based on the number of secrets that satisfy the constrain.
- Most of leakages found by recent works are hard to be exploited. (Need to be confirmed)