On formally bounding information leakage in deterministic programs

Michele Boreale

DiSIA - Università di Firenze

IFIP WG 2.2, TUM - Münich
September 15-18, 2014

Storyline

- Quantitative Information Flow (QIF)
- Computing approximate QIF
- White-box: Abstract Interpretation
- Black-Box: Statistical estimators
- Sampling from 'good' distributions
- Experiments

QIF: motivation

Example (Agat & Sands, SSP'01): cache behaviour

Neither `x` nor `y` are initially cached.

```
if ( h>0 )
    z = x;
else
    z = y;
z = x;
```

QIF: motivation

Example (Agat & Sands, SSP'01): cache behaviour

Neither `x` nor `y` are initially cached.

```
if ( h>0 )
    z = x;
else
    z = y;
z = x;
```

short exec. time implies $\mathtt{h} > 0$; *long* exec. time implies $\mathtt{h} \leq 0$: 1 bit is revealed!

QIF: motivation

Example (Agat & Sands, SSP'01): cache behaviour

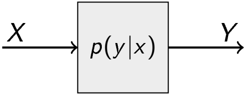
Neither `x` nor `y` are initially cached.

```
if ( h>0 )
    z = x;
else
    z = y;
z = x;
```

short exec. time implies $\mathtt{h} > 0$; *long* exec. time implies $\mathtt{h} \leq 0$: 1 bit is revealed!

Very serious threat, underlies practical attacks to crypto-software (AES, RSA,...).
Type systems can be used to detect potential leaks (Noninterference violations), but in general not to quantify them .
QIF: models and methods to detect and **quantify** leakage.

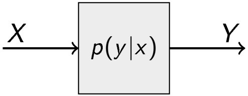
QIF in a nutshell



Programs seen as channels.

- X = input = sensitive information
- Y = observable information
- $p(y|x)$ = conditional probability matrix.

QIF in a nutshell



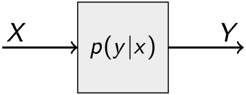
Programs seen as channels.

- X = input = sensitive information
- Y = observable information
- $p(y|x)$ = conditional probability matrix.

Attacker will try a **guess** $\hat{X}$ at X after observing Y. (Min-entropy) leakage is

$$L \overset{\text{def}}{=} \log_2\left(\frac{\Pr[\text{correct guess after observ.}]}{\Pr[\text{correct guess before observ. }]}\right) = \log_2\left(\frac{\Pr[\hat{X} = X]}{\max_x \Pr[X = x]}\right)$$

QIF in a nutshell



$$X \longrightarrow \boxed{p(y|x)} \longrightarrow Y$$

Programs seen as channels.

- X = input = sensitive information
- Y = observable information
- $p(y|x)$ = conditional probability matrix.

Attacker will try a **guess** $\hat{X}$ at X after observing Y. (Min-entropy) leakage is

$$L \stackrel{\text{def}}{=} \log_2\left(\frac{\Pr[\text{correct guess after observ.}]}{\Pr[\text{correct guess before observ. }]}\right) = \log_2\left(\frac{\Pr[\hat{X}=X]}{\max_x \Pr[X=x]}\right)$$

$$\leq \log_2 |ran(Y)|$$

$|ran(Y)|$ = number possible distinct **observables**. Equality can be achieved, e.g. if program is deterministic and X is uniform. So, modulo the $\log_2$

maximum leakage is $|ran(Y)|$ = **program capacity**

An example – easy

```
if ( h>0 )
    z = x;
else
    z = y;
z = x;
```

Assume attacker can detect **sequences** of cache miss (M) or hit (H). Here:

- $X = \mathtt{h}$ chosen at random in $-2^{31}+1..2^{31}$;
- $ran(Y) = \{MM, MH\} \Rightarrow L = 1$ bit

An example – easy

```
if ( h>0 )
    z = x;
else
    z = y;
z = x;
```

Assume attacker can detect **sequences** of cache miss (M) or hit (H). Here:

- $X = \mathtt{h}$ chosen at random in $-2^{31}+1..2^{31}$;
- $ran(Y) = \{MM, MH\} \Rightarrow L = 1$ bit

Indeed, with $\hat{X}$: if $Y = MM$ then any integer in $-2^{31}+1..0$, else any integer in $1..2^{31}$

$$\frac{\Pr[X=\hat{X}]}{\max_x \Pr[X=x]} = \frac{\Pr[X=\hat{X}|Y=MM]\Pr(MM)+\Pr[X=\hat{X}|Y=MH]\Pr(MH)}{\max_x \Pr[X=x]}$$

$$= \frac{(2^{-31})\frac{1}{2}+(2^{-31})\frac{1}{2}}{2^{-32}} = \frac{2^{-31}}{2^{-32}} = 2$$

Another example – less easy

Say `v[]` is contains wages, listed by employees' alphabetical order. Initial ordering of `v[]` is a sensitive info. Same attacker model: $\mathcal{Y} = \{M, H\}^*$.
How many bits can be recovered?

```
public static int BubbleSort(int[] v){
   int n = v.length; int swap;
   for(int i=0; i<n-1; i++){
     for(int j=0; j<n-i-1; j++){
        if(v[j]>v[j+1]){
           swap = v[j];
           v[j] = v[j+1];
           v[j+1] = swap;
               }
              }
            }
           }
```

For large `n`, computing $|ran(Y)|$ can be very complex.

Computing approximate QIF

Why approximate? Just deciding whether $|ran(Y)| > 1$ is NP-hard.
Given a boolean formula `phi(x1,...,xn)`, build the program

```
if   phi(x1,...,xn)
    z = x;
else
    z = y;
z = x;
```

See (Yasuoka & Terauchi, JCS 2011) for more precise results.

Computing approximate QIF

Why approximate? Just deciding whether $|ran(Y)| > 1$ is NP-hard.
Given a boolean formula `phi(x1,...,xn)`, build the program

```
if   phi(x1,...,xn)
    z = x;
else
    z = y;
z = x;
```

See (Yasuoka & Terauchi, JCS 2011) for more precise results.

We must give up something.

- **White box** analysis: statically derive bounds on program capacity. They will be necessarily loose or vacuous in a number of cases.
- **Black box** analysis: by simulation, derive bounds that hold with high confidence. They may be wrong, but only with negligible probability.

Abstract interpretation for cache side channels (Köpf et al., CAV'12)

Overapproximation: $Traces^\sharp(P) \supseteq Traces(P)$ can be easy to compute.
Then $|ran(Y)| = |Traces(P)| \leq |Traces^\sharp(P)|$.
Basic idea of cache AI from (Ferdinand et al., SAS'06).

```
P:   if .. e .. then .. a .. else .. b ..
```

- Possible final cache states (4-blocks cache, LRU)

$$\begin{array}{rcl} c_1 & = & [a, e, \bot, \bot] \\ c_2 & = & [b, e, \bot, \bot] \end{array}$$

- Abstract state cache $c^\sharp = [\{a, b\}, \{e\}, \bot, \bot]$

Abstract interpretation for cache side channels (Köpf et al., CAV'12)

Overapproximation: $Traces^\sharp(P) \supseteq Traces(P)$ can be easy to compute.
Then $|ran(Y)| = |Traces(P)| \leq |Traces^\sharp(P)|$.
Basic idea of cache AI from (Ferdinand et al., SAS'06).

```
P:   if .. e .. then .. a .. else .. b ..
```

- Possible final cache states (4-blocks cache, LRU)

$$\begin{aligned} c_1 &= [a, e, \bot, \bot] \\ c_2 &= [b, e, \bot, \bot] \end{aligned}$$

- Abstract state cache $c^\sharp = [\{a, b\}, \{e\}, \bot, \bot]$
- Effects of block access on abstract cache

$$\begin{aligned} eff([\{a, b\}, \{e\}, \bot, \bot], e) &= \{H\} \\ eff([\{a, b\}, \{e\}, \bot, \bot], a) &= \{M, H\} \end{aligned}$$

- From abstract semantics and *eff* (easily) compute and count

$$Traces^\sharp(P) = \{H\} \cdot \{M, H\} = \{HM, HH\}$$

Imprecision

Analysis can be imprecise mostly due to variable index lookup `A[i]`

(Köpf et al. CAV'12, USENIX'13) present ways to mitigate this, e.g. *partitioning*: replace

```
A[i];
...
```

with

```
if (i >= 8) {
  A[i];
  ...
} else if (i >= 16) {
  A[i]
  ...
}
```

Abstract interpretation for cache side channels (Köpf et al., USENIX'13)

Idea: compute *overapproximation* of concrete traces, $Taces^\sharp(P) \supseteq Traces(P)$. Then $|ran(Y)| = |Traces(P)| \leq |Traces^\sharp(P)|$.

- Concrete states = cache set $\mathcal{C} = \mathcal{B} \to \{0, ..., k-1, k\}$
- Concrete update when accessing block b

$$next(c, b) = \lambda b' \in \mathcal{B}. \begin{cases} 0 & : b = b' \\ c(b') + 1 & : c(b') < c(b) \\ c(b') & : c(b') > c(b) \end{cases}$$

- Abstract states = abstract cache set $\mathcal{C}^\sharp = \mathcal{B} \to \mathcal{P}(\{0, ..., k-1, k\})$
- Abstract update when accessing block b

$$next(c^\sharp, b) = \lambda b' \in \mathcal{B}. \begin{cases} \{0\} & : b = b' \\ \bigcup_{a \in c^\sharp(b)} \left(c^\sharp(b')_{>a} \cup c^\sharp(b')_{<a} + 1\right) \end{cases}$$

Abstract interpretation for cache side channels (2)

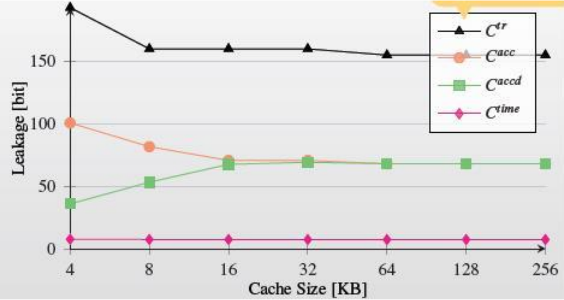
- effect

$$eff\,(c^\sharp, b) = \begin{cases} \{H\} & : c^\sharp(b) \subseteq \{0, ..., k-1\} \\ \{M\} & : c^\sharp(b) = \{k\} \\ \{M, H\} & : \text{otherwise} \end{cases}$$

- P's sequence of block accesses $b_1 b_2 \cdots$ gives rise to $Traces^\sharp(P) = \{H\} \cdot \{H, M\} \cdots$ via $next^\sharp$ and $eff^\sharp$. Easy to compute (fixpoint) and to count.

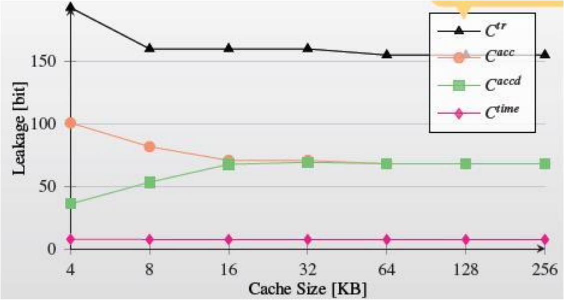
CacheAudit (Köpf et al. USENIX'13)

- A tool based on this approach
- Successful in proving bounds on leakage for AES and Salsa20 software



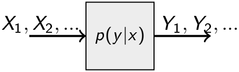
CacheAudit (Köpf et al. USENIX'13)

- A tool based on this approach
- Successful in proving bounds on leakage for AES and Salsa20 software



- Alas, no meaningful upper bounds for e.g. sorting algorithms under trace-based adversaries – yields vacuous bounds $\geq \log(n!)$. This is where loss of precision due to AI shows up
- One reason may be presence of nested iterations with variable indeces

Black-box statistical analysis (Boreale & Paolini, ISC'14)

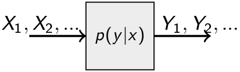


We, the analyst

1. ignore P's code and internal working $p(y|x)$
2. ignore P's input distribution X (to be relaxed later!)
3. obtain a sample of i.i.d. observations $S = Y_1, \ldots, Y_m \quad (m \ll |\mathcal{X}|)$
4. want to estimate $|ran(Y)|$, hence leakage

Analyst needs a function, whatever the program P and the distribution X, given sample yields an estimation of $|ran(Y)|$.

Black-box statistical analysis (Boreale & Paolini, ISC'14)



We, the analyst

1. ignore P's code and internal working $p(y|x)$
2. ignore P's input distribution X (to be relaxed later!)
3. obtain a sample of i.i.d. observations $S = Y_1, \ldots, Y_m \quad (m \ll |\mathcal{X}|)$
4. want to estimate $|ran(Y)|$, hence leakage

Analyst needs a function, whatever the program P and the distribution X, given sample yields an estimation of $|ran(Y)|$.

Estimator

Let $\gamma > 1$ and $1/2 > \delta > 0$. A (γ, δ) estimator is a function $f : \mathcal{Y}^m \to \mathbb{R}^+$ s.t. **for every** X **and** P and $Y = P(X)$

$$\Pr\left(|ran(Y)| \in [f(S)/\gamma, f(S)\cdot\gamma]\right) \geq 1 - \delta$$

Bad news

Negative result 1

There is no such thing as an estimator for $|ran(Y)|$.

Bad news

Negative result 1

There is no such thing as an estimator for $|ran(Y)|$.

Proof intuition. A small fraction ϵ of Y's probability mass could be spread among *a lot* of observables. Before any one of them shows up in the sample, an average of $1/\epsilon$ extractions are necessary. Let $\epsilon \to 0$.

Bad news

Negative result 1

There is no such thing as an estimator for $|ran(Y)|$.
Proof intuition. A small fraction ϵ of Y's probability mass could be spread among *a lot* of observables. Before any one of them shows up in the sample, an average of $1/\epsilon$ extractions are necessary. Let $\epsilon \to 0$.

Negative result 2

If we fix X uniform, $\approx |\mathcal{X}|/\gamma^2$ extractions are still necessary: no significantly better than input enumeration!
Proof intuition. Say $\gamma < \sqrt{2}$ and try to tell these two apart (`x` `n`-bit variable).

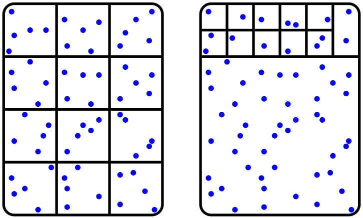
```
P: y = 0;

Q: if x == 42 then y=1 else y=0;
```

Discussion

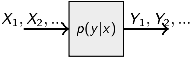
As a function $P : \mathcal{X} \to \mathcal{Y}$, program P partitions the input space $\mathcal{X}$ into equivalence classes (inverse images). The analyst wants to count how many classes are there:

$$|ran(Y)| = |\mathcal{Y}/\sim| \overset{\mathrm{def}}{=} k$$



Another way of looking at the negative results is that, if the analyst has no control over the input distribution, small classes will be very difficult to detect; and there may be *a lot* of them.

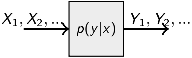
Assume analyst controls input X



Analyst can choose X so that Y is **(nearly) uniform**. Then he

1. generates $S = Y_1, \ldots, Y_m$ i.i.d. and counts how many distinct elements occur in S, say D (e.g. $S = a, b, a, c, a, b \Rightarrow D = 3$)
2. if m is large enough, he expects $D \approx E[D] = k(1 - (1 - 1/k)^m) \overset{\text{def}}{=} g(k)$
3. he lets $g^{-1}(D)$ be his estimation of k.

Assume analyst controls input X



Analyst can choose X so that Y is **(nearly) uniform**. Then he

1. generates $S = Y_1, \ldots, Y_m$ i.i.d. and counts how many distinct elements occur in S, say D (e.g. $S = a, b, a, c, a, b \Rightarrow D = 3$)
2. if m is large enough, he expects $D \approx E[D] = k(1 - (1 - 1/k)^m) \overset{\mathrm{def}}{=} g(k)$
3. he lets $g^{-1}(D)$ be his estimation of k.

Positive result

Let Y be *uniform* and

$$J_t \overset{\mathrm{def}}{=} g^{-1}(D - t)$$

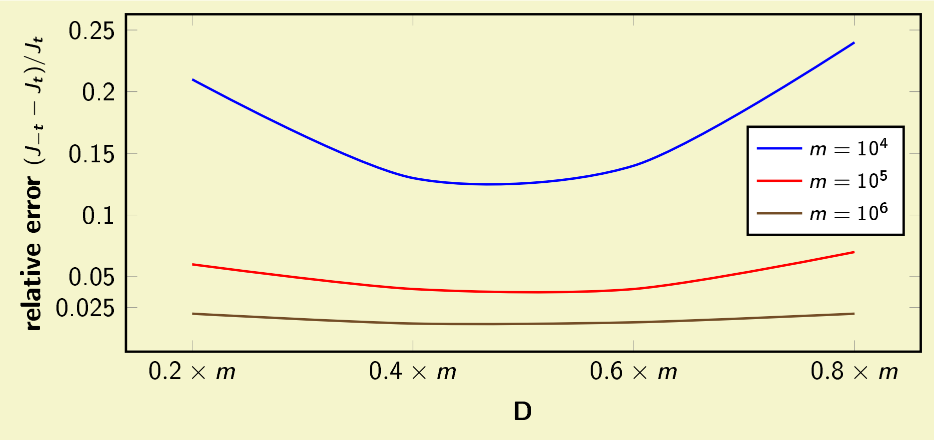
Then

$$\Pr\left(k \in [J_t, J_{-t}]\right) \geq 1 - \delta$$

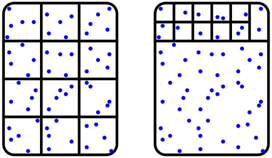
provided $t \geq \sqrt{m \ln(1/\delta)/2}$. (NB: can be extended to *nearly* uniform Y.)

Relative error bounds

We fix $\delta = 0.001$ (99.99% confidence)



How to sample from a good input distribution X?

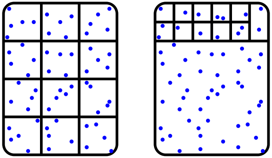


An optimal input distribution assigns the same probability mass, $1/k$, to all classes. E.g. $p^*(x) = \frac{1}{k \cdot |[x]|}$. We consider two sampling algorithms.

1. Markov Chain Monte Carlo. Define a random walk $\{X_t\}_{t \geq 0}$ on the state space $\mathcal{X}$ that "keeps off" the big classes:

1. pick up x according to a fixed proposal distribution $Q(x|X_t)$
2. *accept* x ($X_{t+1} = x$) with probability $\min\{1, |X_t|/|[x]|\}$ else *reject* it ($X_{t+1} = X_t$)

How to sample from a good input distribution X?



An optimal input distribution assigns the same probability mass, $1/k$, to all classes. E.g. $p^*(x) = \frac{1}{k \cdot |[x]|}$. We consider two sampling algorithms.

1. Markov Chain Monte Carlo. Define a random walk $\{X_t\}_{t \geq 0}$ on the state space $\mathcal{X}$ that "keeps off" the big classes:

1. pick up x according to a fixed proposal distribution $Q(x|X_t)$
2. *accept* x $(X_{t+1} = x)$ with probability $\min\{1, |X_t|/|[x]|\}$ else *reject* it $(X_{t+1} = X_t)$

2. Accept-Reject.

1. pick up random x
2. pick up random $u \in [0, 1]$
3. if $u < \frac{\min_{x'} |[x']|}{|[x]|}$ then accept $X = x$ else reject x and goto 1

How to sample from a good input distribution? (2)

Note:

1. both algorithms converge to p^*, but require knowledge of the (relative) size of equivalence classes
2. in practice, $|[x]|$ is approximated in a pre-computation phase
3. MCMC only converges in the limit
4. A-R can be extremely expensive in unbalanced situations and must be tuned

Due to approximations, sampled X is only "good" rather an optimal, and Y not necessarily uniform.
Yet, obtained **lower bounds** on $k = |ran(Y)|$ are still formally valid and often quite good!

Experiment 1: unbalanced classes

```
z=mod(x,2^l);
if mod(z,2^r)==0
    y=z;
else
    y=mod(z,2^r);
return y;
```

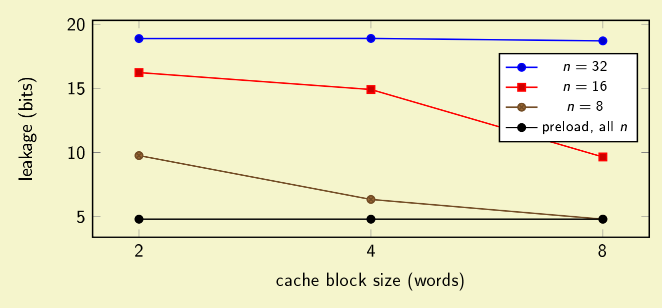
- n-bit input, 2^{r-1} large classes and 2^{l-r} small classes
- $l - r$ large $\Rightarrow$ unbalanced
- confidence $\delta = 0.001$ and $m = 5 \times 10^5$ (note $m \ll 2^n$)
- we report $(\log J_t)/\log k$

n	l	r	k	CMC	MCMC	AR
24	22	22	4.9143×10^6	**0.98**	0.96	**0.98**
	22	20	1.0486×10^6	**0.99**	0.98	**0.99**
	22	2	2.0972×10^6	0.80	0.88	**0.95**
	22	1	2.0972×10^6	0.86	0.93	**0.99**
28	23	23	8.3886×10^6	**0.99**	0.96	**0.99**
	23	20	1.0486×10^6	**0.99**	0.98	**0.99**
	23	2	2.0972×10^6	0.80	0.89	**0.99**
	23	1	4.1943×10^6	0.82	0.90	**0.99**
32	26	26	6.7108×10^7	**0.96**	0.91	**0.96**
	26	23	8.3886×10^6	**0.99**	0.96	**0.99**
	26	2	1.6777×10^7	0.70	0.79	**0.98**
	26	1	3.3554×10^7	0.72	0.80	**0.98**

Experiment 2: sorting algorithms

Recall that (upper) bounds obtained with static analysis (CacheAudit) are vacuous on these algorithms.
We consider trace-based cache leaks in Java implementations of `BubbleSort` and `InsertionSort`. Here

- cache replacement policy: LRU
- varying cache block size (in words): 2,4,8
- preload yes/no
- confidence $\delta = 0.001$ and $n =$ vector length



Conclusion

- Obtaining formal bounds on quantitative information leaks of programs is difficult
- **Static analysis** presupposes access to the source code and depends on precision of underlying abstract domains. Effective at finding good **upper bounds** in specific domains (cache analysis)
- **Statistical estimation** does not require source code, but cannot give in general (tight) upper bounds. If input can be controlled by analyst, effective at finding good **lower bounds**, independently of domain of application
- It looks obvious that the approaches two should be used in conjunction. Further **experiments** and tool validation are called for.

References

- M. Alt, C. Ferdinand, F. Martin, R. Wilhelm. Cache Behavior Prediction by Abstract Interpretation. Proceedings of the Third International Symposium on Static Analysis (SAS), 2006
- M. Boreale, M. Paolini. On formally bounding information leakage by statistical estimation. ISC 2014
- G. Doychev, D. Feld, B. Köpf, L. Mauborgne, J. Reineke. CacheAudit: A Tool for the Static Analysis of Cache Side Channels. USENIX Security 2013
- B. Köpf, L. Mauborgne, M. Ochoa. Automatic Quantification of Cache Side-Channels. CAV 2012
- H. Yasuoka, T. Terauchi. On bounding problems of quantitative information flow. Journal of Computer Security 19(6): 1029-1082, 2011