

Using information theory to measure information flow

P. Malacaria (Queen Mary, University of London)

S. Hunt (City University, London)

D. Clark (King's College London)

Overview

- ⇒ The definition of leakage
- ⇒ overview of the analysis for a simple imperative language
- ⇒ rules for expressions
- ⇒ conclusions

The Problem

- ⇒ **problem**: Static analysis of quantification of interference between two variables caused by running a while program.
- ⇒ **application**: Leakage of secrets from confidential to non-confidential variables in a program.
- ⇒ **attack**: Limited. Attacker knows non-confidential inputs/outputs and program text. Timing not considered.

Our Solution

- ⇒ **calculation**: Use information theory to find average number of “bits of information” leaked across all runs of the program.
- ⇒ **requires**: statistical information – probability distribution on inputs (or some information about this).
- ⇒ **focus**: Languages with a ‘simple function semantics’: program viewed as a transformation of inputs to outputs.
- ⇒ **simplification**: Programs terminate.

Entropy and Information

- ⇒ Surprisal of an event: $\log_2 \frac{1}{p}$.
- ⇒ Total information carried by a set of n events: weighted sum of surprisal values.

$$\mathcal{H} = \sum_{i=1}^n p_i \log \frac{1}{p_i}$$

Called *self information* or *entropy* of the set of events.

- ⇒ Greatest when probability distribution is uniform, 0 when single value is certain.

1234 PIN Guessing Program

- ⇒ Program always guesses PIN is 1234.
- ⇒ PIN always is 1234: program does not leak information.
- ⇒ PIN is 1234 half the time and 6528 half the time: program leaks 1 bit.
- ⇒ every possible PIN value is equally likely (uniform probability distribution on the value space): minimal leakage.

Random variables and program points

- ⇒ A random variable (or discrete random element in this case) is a total function $X : D \rightarrow R$. D and R are finite sets, D has a probability distribution.
- ⇒ joint random variable: (X, Y) defined as $\langle X, Y \rangle$
- ⇒ Entropy of a random variable X :

$$\mathcal{H}(X) = \sum_{x \in R} p(x) \log \frac{1}{p(x)}$$

- ⇒ Associate random variables with expressions, particularly program variables, at program points within a program.
- ⇒ Of interest are observations of values of variables at ι (the *entry point*) and the special node ω (the *exit point*).

Conditional Entropy

⇒ $P((X \upharpoonright (Y = y)) = x) = P(X = x|Y = y)$, where

$$P(X = x|Y = y) = \frac{p(x, y)}{p(y)}$$

⇒

$$\mathcal{H}(X|Y) = \sum_y p(y)\mathcal{H}(X \upharpoonright (Y = y))$$

⇒ A key property of conditional information is that $\mathcal{H}(X|Y) \leq \mathcal{H}(X)$, with equality iff X and Y are independent.

Mutual Information

- ⇒ Given two random variables X and Y , the mutual information between X and Y , written $\mathcal{I}(X;Y)$ is defined as follows:

$$\mathcal{I}(X;Y) = \sum_x \sum_y p(x,y) \log \frac{p(x,y)}{p(x)p(y)}$$

- ⇒ routine manipulation of sums and logs yields:

$$\mathcal{I}(X;Y) = \mathcal{H}(X) + \mathcal{H}(Y) - \mathcal{H}(X,Y)$$

This quantity is a direct measure of the amount of information carried by X which can be learned by observing Y (or vice versa).

Conditional Mutual Information

- ⇒ As with entropy one can define conditional mutual information. The mutual information between X and Y given knowledge of Z , written $\mathcal{I}(X; Y|Z)$, may be defined

$$\mathcal{I}(X; Y|Z) = \mathcal{H}(X|Z) + \mathcal{H}(Y|Z) - \mathcal{H}(X, Y|Z)$$

- ⇒ This expression is used in the most general definition of leakage.

The Definition of Leakage into a Variable at Termination

- ⇒ “How much information carried by confidential inputs can be learned by observation of low security outputs, assuming low inputs are known?”
- ⇒ Language is deterministic: variation in outputs \Leftarrow variation in inputs.
- ⇒ Account for information in low security inputs \Rightarrow only source of surprise in low security outputs is confidential inputs.
- ⇒ For program variable(s) X use X^ι and X^ω as corresponding random variables at entry and exit to program.

Leakage

The amount of leakage of confidential information into variable X due to execution of the program:

$$\mathcal{L}(X) = \mathcal{H}(X^\omega | L^\iota)$$

Alternatively: information shared between final value of X and the initial value of H , given that the initial value of L was already known:

$$\mathcal{L}'(X) = \mathcal{I}(H^\iota; X^\omega | L^\iota)$$

See [Gray 91: Toward a mathematical foundation for information flow security]

Gray's channel capacity

The channel capacity from H to L is

$$C \equiv \lim_{n \rightarrow \infty} C_n$$

where C_n is defined as

$$C_n \equiv \max_{\mathbf{H}, \mathbf{L}} \left(\frac{1}{n} \sum_{i=1}^n \mathcal{I} \left(\begin{array}{l} \text{In-Seq-Event}_{H,i}, \\ \text{Out-Seq-Event}_{H,i}; \\ \text{Final-Out-Event}_{L,i} \mid \\ \text{In-Seq-Event}_{L,i}, \\ \text{In-Seq-Event}_{L,i} \end{array} \right) \right)$$

- ⇒ Where a program defines a function from inputs to outputs, \mathcal{L} and \mathcal{L}' are equivalent:
- ⇒ **Proposition:** Let X, Y, Z be random variables such that, $Z = f(X, Y)$, where f is any function. Then $\mathcal{H}(Z|Y) = \mathcal{I}(X; Z|Y)$.
- ⇒ **Corollary:**
 \mathcal{L} and \mathcal{L}' are equivalent for a deterministic language (let $X = H^\nu$, $Y = L^\nu$ where H^ν and L^ν jointly determine the input space, then $Z = L^\omega$ is a function of (X, Y)).
- ⇒ Advantage of \mathcal{L}' is that it is appropriate for a language with a probabilistic semantics.

⇒ A ‘fair coin’

$X := \text{coin};$

⇒

$$\begin{aligned}\mathcal{I}(H^\iota; X^\omega | L^\iota) &\leq \mathcal{I}(H^\iota; X^\omega) \\ &= \mathcal{H}(H^\iota) - \mathcal{H}(H^\iota | X^\omega) \\ &= \mathcal{H}(H^\iota) - \mathcal{H}(H^\iota) = 0\end{aligned}$$

⇒

$$\mathcal{H}(X^\omega | L^\iota) = \mathcal{H}(X^\omega) = \frac{1}{2} \log 2 + \frac{1}{2} \log 2 = 1$$

- ⇒ Unlike the general probabilistic setting it is safe (conservative) to calculate a program's overall input-output leakage by considering its outputs separately:
- ⇒ **Proposition:** Let X be a vector of random variables X_1, \dots, X_n modelling some outputs in a non-probabilistic language. Assuming H^ι and L^ι jointly determine the input space then $\mathcal{L}(X) \leq \mathcal{L}(X_1) + \dots + \mathcal{L}(X_n)$.
- ⇒ **Comment:** The assumption about H^ι and L^ι jointly determine the input space is necessary as the proposition is not true in general.

Analysis for While Language

- ⇒ Assume for vector of confidential variables, H at ι we know bounds on the entropy for H^ι . Default is $0 \leq \mathcal{H}(H) \leq N$ where N is number of bits in vector.
- ⇒ For each non-confidential variable X calculate upper and lower bounds on $\mathcal{L}(X) = \mathcal{H}(X^\omega | L^\iota)$.

Analyse for worst case choice of L^ι

- ⇒ likely little or no information available about low inputs
- ⇒ low inputs may be in *control* of attacker
- ⇒ For random variable X , let $X_\lambda = (X|L = \lambda)$
- ⇒ Analysis calculates bounds on $\mathcal{H}(X_\lambda^\omega)$ given bounds on $\mathcal{H}(H_\lambda^\iota)$
- ⇒ require bounds which hold for all λ
- ⇒ Proposition:

$$(\forall \lambda . a \leq \mathcal{H}(X_\lambda^\omega) \leq b \Rightarrow (a \leq \mathcal{H}(X^\omega|L^\iota) \leq b))$$

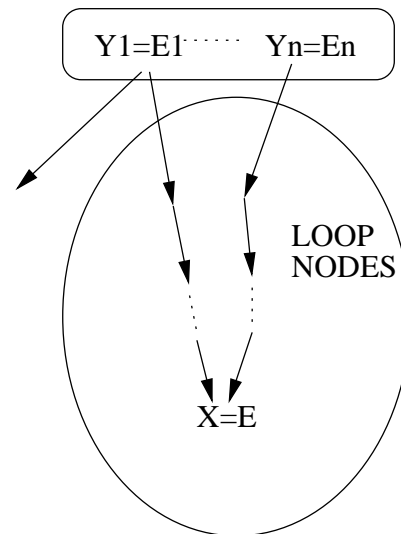
(average is bounded by its smallest and largest terms)

- ⇒ Assuming k -bit variables, the bounds $0 \leq \mathcal{H}(X^\iota | L^\iota = \lambda) \leq k$ are always valid
- ⇒ For all low-security variables $X \in L$, $\mathcal{H}(X^\iota | L^\iota = \lambda) = 0$
- ⇒ In most cases, it will be reasonable to assume that the initial values of H and L are independent, in which case $\mathcal{H}(X^\iota | L^\iota = \lambda) = \mathcal{H}(X^\iota)$ for all high-security variables $X \in H$, so the problem of calculating useful initial assumptions reduces to the problem of finding good estimates of the entropies of the high security inputs

Quantitative analysis

- ⇒ Inference rules for commands, expressions and dependencies
- ⇒ Command rules allow syntax directed analysis: Logical rules, direct flows, four rules for if statements and the “data processing” rule (you can’t get out more entropy from a deterministic system than you put in):
 - ⇒ *Data Processing* Let X_1, \dots, X_n, Z be random variables with a common domain D and such that $Z = f(X_1, \dots, X_n)$, where f is any total function. (Equivalently, in function notation: $Z = f \circ (\lambda d \in D. \langle X_1(d), \dots, X_n(d) \rangle)$.) Then $\mathcal{H}(Z) \leq \mathcal{H}(X_1, \dots, X_n) \leq \mathcal{H}(X_1) + \dots + \mathcal{H}(X_n)$.
- ⇒ Dependency rules needed to handle while loops.
- ⇒ Journal of Computer Security submission.

Intuition for Loops



Analysing expressions

- ⇒ For some expressions can produce bounds on the leakage due to evaluation of expression in terms of the leakage associated with sub-expressions.
- ⇒ Significant success for upper bounds on equality tests when one sub-expression contains very little confidential information.
- ⇒ Can produce some useful bounds for certain arithmetic expression, e.g. $a \leq \mathcal{H}(X) \wedge \mathcal{H}(Y) \leq b \Rightarrow a - b \leq \mathcal{H}(X + Y)$.
- ⇒ Some arithmetic operator analyses are only possible if accompanied by subsidiary analyses, e.g. constant propagation, parity analyses.
- ⇒ Inequalities leak too close to a bit almost every time.

$$[\text{Eq}] \frac{n \vdash [E_1] \geq a \quad n \vdash [E_2] \leq b}{n \vdash [E_1 == E_2] \leq \mathcal{B}(q)} \quad q \leq 0.5, \mathcal{U}_k(q) \leq (a - b), k = \text{bits}(E_1)$$

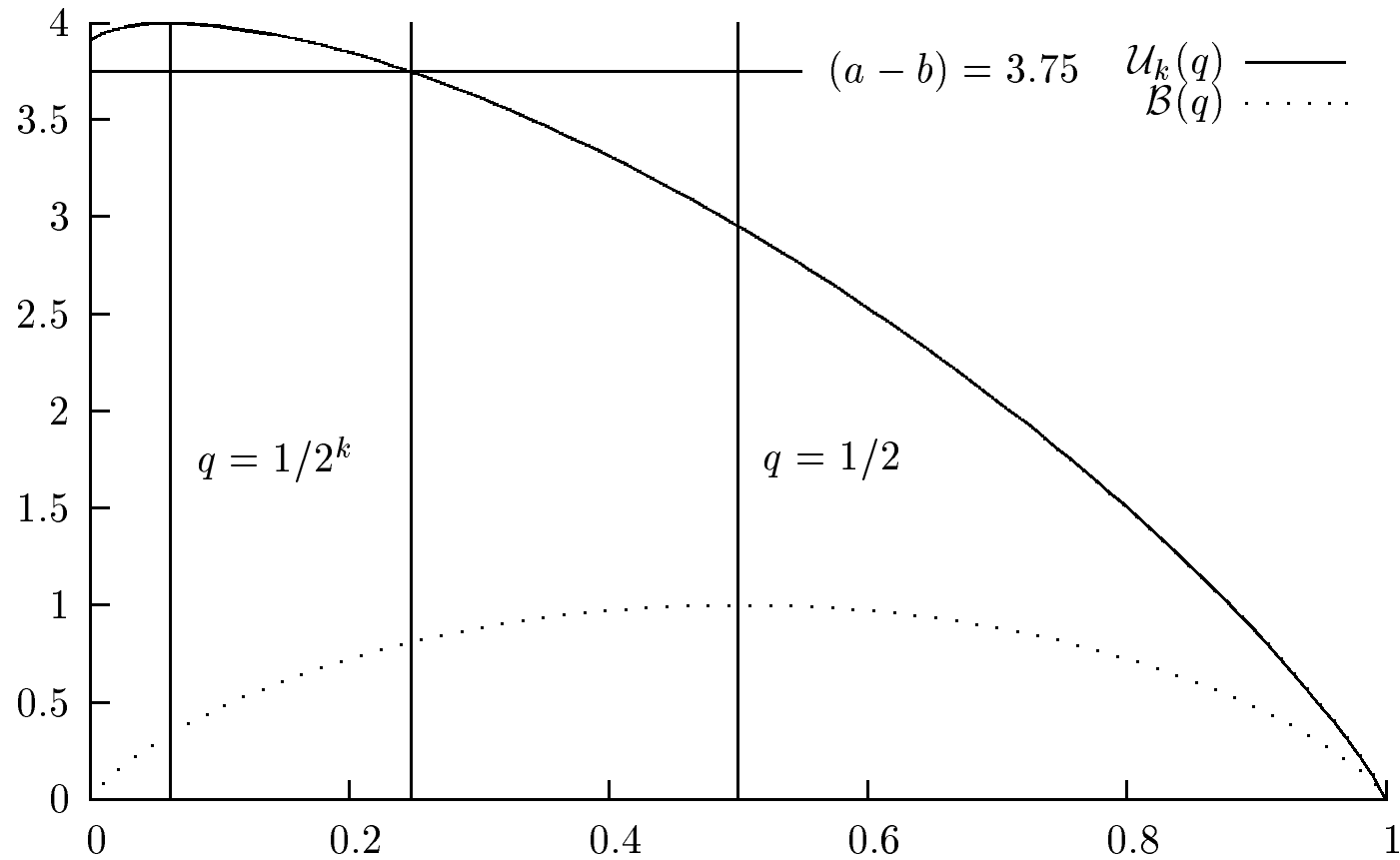
$$[\text{OpMax}] \frac{n \vdash [E_1] \leq b_1 \quad n \vdash [E_2] \leq b_2}{n \vdash [E_1 \odot E_2] \leq b_1 + b_2} \quad [\text{Neg}] \frac{n \vdash [E] \sim a}{n \vdash [-E] \sim a}$$

$$[\text{AddMin}] \frac{n \vdash [E_1] \geq a_1 \quad n \vdash [E_2] \geq a_2 \quad n \vdash [E_1] \leq b_1 \quad n \vdash [E_2] \leq b_2}{n \vdash [E_1 + E_2] \geq \max(a_1, a_2) - \min(b_1, b_2)}$$

$$[\text{ZeroMult}] \frac{}{n \vdash [E_1 * E_2] = 0} \quad E_2 = 0$$

$$[\text{OddMult}] \frac{n \vdash [E_1] \sim a \quad n \vdash [E_2] = 0}{n \vdash [E_1 * E_2] \sim a} \quad E_2 \text{ is odd}$$

Example equality test analysis



Some conclusions

- ⇒ Believe we have the right quantitative definition.
- ⇒ Loops difficult. Can do better via rates of leakage
- ⇒ best (tight bounds) results when no loops involved and all leakage is indirect, ie. via tests.
- ⇒ other work has exploited the connection between equivalence relations and random variables: used this to measure flows in a simple functional language.
- ⇒ extend to working with more information about probability distributions
- ⇒ Michele Boreale applies our deterministic leakage definition to pi-calculus.