Quantitative Information Flow Analysis
Hauptseminar WS'2010/11

Sergey Grebenshchikov

June 30, 2011

Contents

Introduction

The goal of *quantitative information flow* (QIF) analysis is to quantify the amount of sensitive information revealed (*leaked*) by a program. This problem is relevant for any program that handles sensitive or secret data - anonymization protocols, electronic voting, payment or banking systems, appointment organizers providing busy/free data and networked multiplayer games, among others. In a nutshell, we seek an answer to the question:

> "How much information can an attacker gain about sensitive data from a program's observable behavior?"

In the following, we will present several approaches to this question.

1 Preliminaries

1.1 Secret information and information leaks

In order to describe the secret information a program leaks, we need to define what we mean by "secret information". One approach commonly taken (e.g in [4], [7]) is to consider program inputs consisting of a *high* (secret) and a *low* (public) part, and assume the low input and the entire source code (i.e. transition system) and output of a program to be public. In this presentation we assume the *entire* input as secret and the entire output and source code as public.

Information-theoretic entropy

In order to quantify the information leaked by a program, we need to introduce a quantitative measure of information. *Information-theoretic entropy* (also referred to as *Shannon entropy*) [6] is such a measure.

Let $(\Omega_X, \Pr)$ be a discrete probability space and Ω a measurable space. A *random variable* X is a map $\Omega_X \to \Omega$. If the probability distribution of X is $p_X : \Omega \to [0,1]$, we write $X \sim p_X$. For $x \in \Omega$ we use the suggestive notation $\Pr[X = x]$ for $p_X(x)$. We denote the fact that a random variable Y has the same probability distribution as a random variable X by writing $Y \simeq X$. A random variable can be obtained by applying a function to another random variable. We write $Y = f(X)$ for $Y = f \circ X$. We define the *binary entropy* $\mathrm{H}[X]$ of X as follows:

$$\mathrm{H}[X] := -\sum_{x \in \Omega} \Pr[X = x] \log_2 \Pr[X = x] \tag{1}$$

If X is the identity function $\mathrm{id}_{\Omega_X} : \Omega_X \to \Omega_X$ and is uniformly distributed on Ω_X, i.e. $\forall x \in \Omega_X : \Pr[X = x] = \frac{1}{|\Omega_X|}$, we can derive a simple expression for $\mathrm{H}[X]$:

$$\begin{aligned}\mathrm{H}[X] &= -\sum_{x \in \Omega_X} \Pr[X = x] \log_2 \Pr[X = x] \\ &= \frac{1}{|\Omega_X|} \sum_{x \in \Omega_X} \log_2 |\Omega_X| \\ &= \log_2 |\Omega_X|\end{aligned}$$

Entropy can be interpreted as describing *uncertainty* of the value of a random variable. In the special case of binary entropy, $\mathrm{H}[X]$ is the average number of bits needed to determine the value of X. The above result for a uniform distribution matches our intuition: to identify an element out of a set of n equally probable elements, we need $\log_2 n$ bits.

Given two random variables $X : \Omega_X \to \Omega$ and $Y : \Omega_Y \to \Omega'$, we define the *conditional entropy* $\mathrm{H}[X|Y = y]$ of X given the value y of Y as follows:

$$\mathrm{H}[X|Y = y] := -\sum_{x \in \Omega} \Pr[X = x|Y = y] \log_2 \Pr[X = x|Y = y] \tag{2}$$

The average conditional entropy of X given Y is the average of $\mathrm{H}[X|Y = y]$ over $y \in \Omega'$:

$$\mathrm{H}[X|Y] := \sum_{y \in \Omega'} \Pr[Y = y]\, \mathrm{H}[X|Y = y] \tag{3}$$

Applying the definition of conditional probability, we can derive the following equation for $\mathrm{H}[U|V]$:

$$\mathrm{H}[U|V] = \mathrm{H}[U, V]\, \mathrm{H}[V]$$

The average conditional entropy $\mathrm{H}[X|Y]$ describes the uncertainty of the value of X after observing the value of Y.

Example 1 (Conditional entropy)**.** Let $\Omega_X := \{1,2,3,4\}$. Consider the two random variables X and Y:

$$X : \Omega_X \to \Omega_X \quad Y : \Omega_X \to \{0,1\}$$
$$X = \mathrm{id}_{\Omega_X} \qquad Y = \begin{cases} 0 & X \leq 2 \\ 1 & \text{otherwise.} \end{cases}$$

Assume a uniform probability $\Pr[x] = \frac{1}{|\Omega_X|} = \frac{1}{4}$ for all elementary events $x \in \Omega_X$. The entropy of X is $\mathrm{H}[X] = \log_2 |\Omega_X| = 2$. Intuitively, observing the result of Y gives us a binary distinction about the value of X, so for $X|Y$ we expect an entropy reduced by one bit. For the elementary and conditional probabilities we obtain:

$$\Pr[Y = 0] = \Pr[Y = 1] = \frac{1}{2}$$
$$\Pr[X = x | Y = 0] = \begin{cases} \frac{1}{2} & x \leq 2 \\ 0 & \text{otherwise.} \end{cases}$$

Plugging the probabilities into the formulas for conditional entropy, we get:

$$\begin{aligned} \mathrm{H}[X|Y=0] &= -\sum_{x=1}^{4} \Pr[X = x, Y = 0] \log_2 \Pr[X = x, Y = 0] \\ &= -\sum_{x=1}^{2} \frac{1}{4} \log_2 \frac{1}{4} = 1 \\ \mathrm{H}[X|Y=1] &= -\sum_{x=3}^{4} \Pr[X = x, Y = 1] \log_2 \Pr[X = x, Y = 1] \\ &= -\sum_{x=3}^{4} \frac{1}{4} \log_2 \frac{1}{4} = 1 \end{aligned}$$

For the average conditional entropy we obtain, as expected:

$$\begin{aligned} \mathrm{H}[X|Y] &= \sum_{y \in \{0,1\}} \Pr[Y = y] \, \mathrm{H}[X|Y = y] \\ &= \frac{1}{2} \cdot 1 + \frac{1}{2} \cdot 1 = 1 \end{aligned}$$

The remaining uncertainty of X is 1 bit.

1.2 Programs and computations

We consider programs $\mathcal{P}$ of the form $\mathcal{P} = (\mathsf{S}, \mathsf{I}, \mathsf{F}, \mathcal{T})$, where

- S is a set of program states
- $\mathsf{I} \subseteq \mathsf{S}$ is a set of *initial* states
- $\mathsf{F} \subseteq \mathsf{S}$ is a set of *final* states
- $\mathcal{T}$ is a set of *transitions*, such that for each $\tau \in \mathcal{T}$ a *transition relation* $\rho_\tau \subseteq \mathsf{S} \times \mathsf{S}$ is given.

We define the *program transition relation* ρ of $\mathcal{P}$ as the union of its transition relations:

$$\rho := \bigcup_{\tau \in \mathcal{T}} \rho_\tau$$

Using ρ, we can define the *input-output relation* ρ_{IO} of $\mathcal{P}$ as the transitive closure of ρ, restricted to initial and final states:

$$\rho_{\mathsf{IO}} \coloneqq \rho^* \cap (\mathsf{I} \times \mathsf{F})$$

We require that ρ_{IO} is a total function $\mathsf{I} \to \mathsf{F}$, i.e.:

$$\forall s \in \mathsf{I} : \exists_1 s' \in \mathsf{F} : (s, s') \in \rho_{\mathsf{IO}}$$

A final state $s' \in \mathsf{F}$ is *reachable* from an initial state $s \in \mathsf{I}$ iff $(s, s') \in \rho_{\mathsf{IO}}$. We write $\mathsf{F}_{\mathsf{reach}}$ for the set of reachable final states:

$$\mathsf{F}_{\mathsf{reach}} \coloneqq \{s' \in \mathsf{F} \mid \exists s \in \mathsf{I} : s' \text{ reachable from } s\} \subseteq \mathsf{F}$$

For a final state $s' \in \mathsf{F}$, the *preimage* $\mathcal{P}^{-1}(s')$ is the set of all initial states s from which s' is reachable:

$$\mathcal{P}^{-1}(s') \coloneqq \{s \in \mathsf{I} \mid (s, s') \in \rho_{\mathsf{IO}}\}$$

An execution step $s \xrightarrow{\rho} s'$ of $\mathcal{P}$ can be described using the *post operator*:

$$\mathsf{Post} : 2^{\mathsf{S}\times\mathsf{S}} \times \mathsf{S} \to 2^{\mathsf{S}}$$
$$\mathsf{Post}_\rho(s \in \mathsf{S}) \coloneqq \{s' \in \mathsf{S} \mid (s, s') \in \rho\}$$

The definition of Post naturally extends to sets of states:

$$\mathsf{Post} : 2^{\mathsf{S}\times\mathsf{S}} \times (2^{\mathsf{S}} \cup \mathsf{S}) \to 2^{\mathsf{S}}$$
$$\mathsf{Post}_\rho(X \subseteq \mathsf{S}) \coloneqq \{s' \in \mathsf{S} \mid \exists s \in \mathsf{S} : (s, s') \in \rho\}$$

The set of reachable final states $\mathsf{F}_{\mathsf{reach}}$ can be computed by iterating Post:

$$\begin{aligned}\mathsf{F}_{\mathsf{reach}} &= \mathsf{F} \cap \left(\bigcup_{k=0}^{\infty} \mathsf{Post}_\rho^k(\mathsf{I})\right) \\ &= \mathsf{F} \cap \mathsf{lfp}(\mathsf{Post}_\rho, \mathsf{I})\end{aligned}$$

Computing the above fixpoint explicitly can be prohibitively expensive. This is due to the large number of states of almost any practical program. We can, however, employ approximative methods to remedy this problem.

Over-approximation of $\mathsf{F}_{\mathsf{reach}}$

Abstract interpretation overcomes the limitations of explicit reachability computation by approximating the Post operator. This approximation is achieved by *abstracting* sets of states. For a set X of states, an *abstraction* of X is a superset of X. We compute abstractions using an *abstraction function* $\alpha : 2^{\mathsf{S}} \to 2^{\mathsf{S}}$, a monotone (w.r.t set inclusion) function that maps sets of states to abstractions. Instead of iterating Post_ρ, we iterate the composition $\mathsf{Post}_\rho{}^\sharp$ of Post_ρ with α:

$$\begin{aligned}\mathsf{Post}_\rho{}^\sharp &\coloneqq \alpha \circ \mathsf{Post}_\rho \\ \mathsf{F}_{\mathsf{reach}}{}^\sharp &\coloneqq \mathsf{F} \cap \mathsf{lfp}\left(\mathsf{Post}_\rho{}^\sharp, \alpha(\mathsf{I})\right)\end{aligned}$$

By the monotonicity of α, the resulting set $\mathsf{F}_{\mathsf{reach}}{}^\sharp$ is an over-approximation of $\mathsf{F}_{\mathsf{reach}}$, i.e.:

$$\mathsf{F}_{\mathsf{reach}} \subseteq \mathsf{F}_{\mathsf{reach}}{}^\sharp$$

The abstraction function α can be chosen such that the fixpoint can be computed in a finite number of steps.

Under-approximation of $\mathsf{F}_{\mathsf{reach}}$

Abstract interpretation does not yet provide practical approaches to under-approximation. Instead, we symbolically execute a program for a selected set of initial states $\{s_1, \ldots, s_n\} \subseteq \mathsf{I}$. We assume a symbolic execution of $\mathcal{P}$ on a initial state s_i yields a tuple $(\pi_i, s_i') \in \mathcal{T}^+ \times \mathsf{F}$, where π_i is a path $s_i \to s_i'$ and s_i' is the final state reached. We can derive an under-approximation ${\mathsf{F}_{\mathsf{reach}}}^\flat$ of $\mathsf{F}_{\mathsf{reach}}$ by collecting the final states encountered along the paths:

$$ {\mathsf{F}_{\mathsf{reach}}}^\flat := \bigcup_{i=1}^{n} \{s' \mid \exists s \in \mathsf{S} : (s, s') \in (\rho_{\pi_i} \cap (\mathsf{I} \times \mathsf{F}))\} $$

We know each final state in ${\mathsf{F}_{\mathsf{reach}}}^\flat$ to be reachable in $\mathcal{P}$. We obtain the inclusion:

$$ {\mathsf{F}_{\mathsf{reach}}}^\flat \subseteq \mathsf{F}_{\mathsf{reach}} $$

We will use the methods from this section for the approximative QIF algorithms in section 3.

Example 2 (Programs and computations)**.** Consider the program given in Fig. 2. The set of states is given by the set of valuations of program variables and the

```
0:  byte f(byte x) { // 0 <= x <= 255;
1:        byte ret=0;
2:        if(x <= 127)
3:           ret=x;
4:        else
5:           ret=0;
6:        return ret;
    }
```

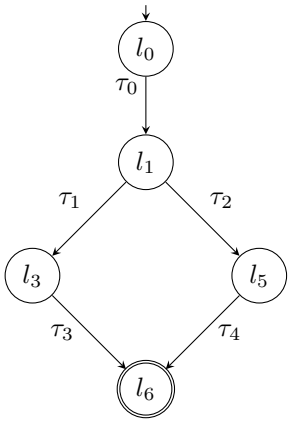
Figure 1: Program for Example 2

program counter:

$$ \mathsf{S} = \{\langle \mathsf{ret} = u, x = v, \mathsf{pc} = w \rangle \mid u, v \in \{0, .., 255\}, w \in \{0, .., 6\}\} $$

The set I of initial states is given by

$$ \mathsf{I} = \{s \in \mathsf{S} \mid s.\mathsf{pc} = 0 \wedge s.\mathsf{ret} = 0\} $$



$$
\begin{aligned}
\rho_{\tau_0} &\equiv \mathsf{pc} = 0 \wedge 0 \leq x \leq 255 \wedge x' = x \wedge \mathsf{ret}' = 0 \wedge \mathsf{pc}' = 1 \\
\rho_{\tau_1} &\equiv \mathsf{pc} = 1 \wedge x \leq 127 \wedge x' = x \wedge \mathsf{ret}' = x \wedge \mathsf{pc}' = 3 \\
\rho_{\tau_2} &\equiv \mathsf{pc} = 1 \wedge 128 \leq x \wedge x' = x \wedge \mathsf{ret}' = 0 \wedge \mathsf{pc}' = 5 \\
\rho_{\tau_3} &\equiv \mathsf{pc} = 3 \wedge x' = x \wedge \mathsf{ret}' = \mathsf{ret} \wedge \mathsf{pc}' = 6 \\
\rho_{\tau_4} &\equiv \mathsf{pc} = 5 \wedge x' = x \wedge \mathsf{ret}' = \mathsf{ret} \wedge \mathsf{pc}' = 6 \\
\rho_{\mathsf{lo}} &= (\mathsf{pc} = 0 \wedge 0 \leq x \leq 127 \wedge x' = x \wedge \mathsf{ret}' = x \wedge \mathsf{pc}' = 6) \vee \\
&\quad\ (\mathsf{pc} = 0 \wedge 128 \leq x \leq 255 \wedge x' = x \wedge \mathsf{ret}' = 0 \wedge \mathsf{pc}' = 6)
\end{aligned}
$$

Figure 2: Transition system for Example 2

The set F of final states is given by

$$\mathsf{F} = \{s \in \mathsf{S} \mid s.\mathsf{pc} = 6\}$$

The set of $\mathcal{T}$ of transitions is given by $\mathcal{T} = \{\tau_1, \tau_2, \tau_3, \tau_4\}$ and is shown in Fig. 2 with the respective transition relations. The program returns values less than or equal to 127 unchanged and returns 0 otherwise, thus the set $\mathsf{F}_{\mathsf{reach}}$ of reachable final states is given by:

$$\begin{aligned}\mathsf{F}_{\mathsf{reach}} = &\{s \in \mathsf{F} \mid 0 \leq s.x \leq 127 \wedge s.\mathsf{ret} = x \wedge s.\mathsf{pc} = 6\} \\ &\cup \{s \in \mathsf{F} \mid 128 \leq s.x \leq 255 \wedge s.\mathsf{ret} = 0 \wedge s.\mathsf{pc} = 6\}\end{aligned}$$

2 Exact static analysis

In this section, we present the conceptual framework and algorithm for the exact static QIF analysis method presented in [1]. This presentation differs from [1] in that we assume no high/low structure on the program states.

2.1 Qualitative information flow

We use *set partitions* to characterize partial knowledge about the underlying set. We call the elements of a partition *blocks*. A partition P of a set X models that each $x \in X$ is known up to its enclosing subset of X in P. We say that a partition P is *more precise* than P' (in symbols: $P \sqsubseteq P'$) if each partition block of P is contained in a partition block of P':

$$P \sqsubseteq P' :\Longleftrightarrow \forall A \in P : \exists B \in P' : A \subseteq B$$

Let $\mathcal{P} = (\mathsf{S}, \mathsf{I}, \mathsf{F}, \mathcal{T})$ be a program. We require that $\mathcal{P}$ implements a total function, i.e. from each initial state we can reach exactly one final state and, as a consequence, $\mathsf{I} = \bigcup_{s' \in \mathsf{F}_{\mathsf{reach}}} \mathcal{P}^{-1}(s')$. The preimage operator thus induces a partition Π on the set of initial states I:

$$\Pi := \left\{\mathcal{P}^{-1}(s') \mid s' \in \mathsf{F}_{\mathsf{reach}}\right\}$$

We can qualitatively characterize the information a program reveals about its input in terms of the partition Π. For example, $\Pi = \{\mathsf{I}\}$ models a program that does not reveal any information about its input. On the other hand, $\Pi = \{\{s\} \mid s \in \mathsf{I}\}$ models a program that reveals its input completely.

2.2 Refinement for leak discovery

In order to construct the preimage partition Π, we compute an equivalence relation $\approx \subseteq \mathsf{I} \times \mathsf{I}$ such that Π is equal to the set of equivalence classes of $\approx$:

$$\Pi = \mathsf{I} / \approx = \{[s]_{\approx} \mid s \in \mathsf{I}\}$$

Given a program $\mathcal{P}$ and an equivalence relation $R \subseteq \mathsf{I} \times \mathsf{I}$, there is an *information leak with respect to* R if there is a pair of paths π_1, π_2 corresponding to computations that lead from R-equivalent initial states $s_1 \sim_R s_2$ to inequal final states $s_1' \neq s_2'$:

$$\begin{aligned}&\mathsf{Leak}_{\mathcal{P}}(R, \pi_1, \pi_2) \equiv \\ &\quad \exists s_1, s_2 \in \mathsf{I} : \exists s_1', s_2' \in \mathsf{F}_{\mathsf{reach}} : \\ &\qquad (s_1, s_1') \in \rho_{\pi_1} \wedge (s_2, s_2') \in \rho_{\pi_2} \wedge s_1 \sim_R s_2 \wedge s_1' \neq s_2'\end{aligned}$$

A refinement algorithm

We construct $\approx$ by successively refining a candidate relation R. R is initialized to the coarsest equivalence relation $R_0 = \mathsf{I} \times \mathsf{I}$ – i.e. $\mathsf{I}/R_0 = \{\mathsf{I}\}$ – and then successively refined until no more leaks can be discovered. This final relation is the largest relation over I such that no leaks in the above sense exist. We refine a relation R with leak paths π_1, π_2 by distinguishing between initial states that lead to different output states along π_1, π_2. This is realized by splitting the equivalence classes of the initial states by removing the corresponding tuples from the equivalence relation. The refinement conjunct $\mathsf{Refine}_{\mathcal{P}}$ implementing this is shown in (4). The leak discovery and refinement procedure is formalized in Algorithm 1.

$$\mathsf{Refine}_{\mathcal{P}}(\pi_1, \pi_2) \equiv \{(s_1, s_2) \in \mathsf{I} \times \mathsf{I} \mid \forall s_1', s_2' \in \mathsf{F}_{\mathsf{reach}} : ((s_1, s_1') \in \rho_{\pi_1} \wedge (s_2, s_2') \in \rho_{\pi_2}) \rightarrow s_1' = s_2'\} \tag{4}$$

Algorithm 1 DISCO – Information leak discovery

procedure DISCO($\mathcal{P}$):
  $R \leftarrow \mathsf{I} \times \mathsf{I}$
  while exists $\pi_1, \pi_2 \in \mathcal{T}^+ : \mathsf{Leak}_{\mathcal{P}}(R, \pi_1, \pi_2)$ **do**
    $R \leftarrow R \cap \mathsf{Refine}_{\mathcal{P}}(\pi_1, \pi_2)$
  $\approx \leftarrow R \cup \mathsf{id}_{\mathsf{I}}$
  return $\approx$

Theorem 1. *The set of equivalence classes of the relation $R =$ DISCO($\mathcal{P}$) computed by* DISCO *is the preimage partition* Π*:*

$$\Pi = I/R$$

Proof. "$\subseteq$" Let $X = \mathcal{P}^{-1}(s') \in \Pi$ for some $s' \in \mathsf{F}_{\mathsf{reach}}$. Let $s_1, s_2 \in X$. Since we assume $\mathcal{P}$ implements a total function (i.e. $\{\mathcal{P}^{-1}(s') \mid s' \in \mathsf{F}_{\mathsf{reach}}\}$ is a partition), no other final state $t' \in \mathsf{F}_{\mathsf{reach}}, t' \neq s'$ is reachable from s_1 or s_2. It follows that $\forall \pi_1, \pi_2 \in \mathcal{T}^+ : (s_1, s_1') \in \rho_{\pi_1} \wedge (s_2, s_2') \in \rho_{\pi_2} \rightarrow s_1' = s_2' = s'$. Since R is the largest relation such that this property holds for all pairs of equivalent states, we obtain $s_1 \sim_R s_2$. Since s_1, s_2 were chosen arbitrarily from X, there exists an $Y \in \mathsf{I}/R$ such that $X \subseteq Y$. Assume there exists a state $t \in Y \setminus X$. Then $\forall \pi_1, \pi_2 \in \mathcal{T}^+ : (s_1, s_1') \in \rho_{\pi_1}, (t, t') \in \rho_{\pi_2} \rightarrow s_1' = t'$. By the above argument, however, $t' = s_1' = s'$ and thus $t \in \mathcal{P}^{-1}(s') = X$. Thus $X = Y \in \mathsf{I}/R$.

"$\supseteq$" Let $X \in I/R$, $s_1, s_2 \in X$, i.e. $s_1 \sim_R s_2$. By construction of R, we have

$$\forall \pi_1, \pi_2 \in \mathcal{T}^+ : \forall s_1', s_2' \in \mathsf{F}_{\mathsf{reach}} : (s_1, s_1') \in \rho_{\pi_1} \wedge (s_2, s_2') \in \rho_{\pi_2} \rightarrow s_1' = s_2'$$

Since $\mathcal{P}$ implements a total function, there exists one state $s' \in \mathsf{F}_{\mathsf{reach}}$, such that $\forall \pi_1, \pi_2 \in \mathcal{T}^+ : (s_1, s_1') \in \rho_{\pi_1} \wedge (s_2, s_2') \in \rho_{\pi_2} \rightarrow s_1' = s_2' = s'$. It follows that $\{s_1, s_2\} \subseteq \mathcal{P}^{-1}(s')$ and thus $\exists Y = \mathcal{P}^{-1}(t') \in \Pi : X \subseteq Y$. As Π is a partition of I and X and Y have a nonempty intersection, $X = Y$. Thus $X = Y = \mathcal{P}^{-1}(s') \in \Pi$.

□

We have shown how to compute the preimage partition Π as the set of equivalence classes of an equivalence relation $\approx \subseteq \mathsf{I} \times \mathsf{I}$. We will now utilize Π to express quantitative information flow properties of the underlying program.

2.3 Quantitative information flow

In order to quantify the information leaked by a program, we model a program as a set of random variables and use Shannon entropy as an information measure. We derive a connection between the preimage partition Π and quantitative information flow in terms of Shannon entropy.

Programs as random variables

Given a program $\mathcal{P} = (\mathsf{S}, \mathsf{I}, \mathsf{F}, \mathcal{T})$, we consider the probability space $(\mathsf{I}, \Pr)$ on the set of program inputs I. In the following, we assume $\Pr$ constant on I, i.e. for all initial states $s \in \mathsf{I} : \Pr[s] = \frac{1}{|\mathsf{I}|}$. We model the behaviour of a program using the random variables U and V. The variable $U := \mathrm{id}_{\mathsf{I}} : \mathsf{I} \to \mathsf{I}$ models the choice of program input, while $V := \rho_{\mathsf{IO}}(U) : \mathsf{I} \to \mathsf{F}$ models a computation of $\mathcal{P}$. For $s \in \mathsf{I}$ and $s' \in \mathsf{F}$, we can derive the following probabilities:

$$\begin{aligned}
\Pr[V = s'] &= \Pr[\rho_{\mathsf{IO}}(U) = s'] \\
&= \frac{\left|\mathcal{P}^{-1}(s')\right|}{|\mathsf{I}|} \\
\Pr[U = s, V = s'] &= \begin{cases} \frac{1}{|\mathsf{I}|} & s \in \mathcal{P}^{-1}(s') \\ 0 & \text{otherwise.} \end{cases} \\
\Pr[U = s | V = s'] &= \frac{\Pr[U = s, V = s']}{\Pr[V = s']} \\
&= \begin{cases} \frac{1}{|\mathcal{P}^{-1}(s')|} & s \in \mathcal{P}^{-1}(s') \\ 0 & \text{otherwise.} \end{cases}
\end{aligned}$$

Applying the concept of entropy outlined above, $\mathrm{H}[U]$ describes the uncertainty of the program input. We assume that all input events have the same probability, thus $\mathrm{H}[U] = \log_2 |\mathsf{I}|$. The conditional entropy $\mathrm{H}[U|V = s']$ describes the uncertainty about the program input, *given a specific program output*. Using the probabilities derived above, we can express $\mathrm{H}[U|V = s']$ in terms of the size of the preimage set $\mathcal{P}^{-1}(s')$:

$$\begin{aligned}
\mathrm{H}[U|V = s'] &= -\sum_{s \in \mathsf{I}} \underbrace{\Pr[U = s|V = s']}_{=0 \text{ for } s \notin \mathcal{P}^{-1}(s')} \log_2 \Pr[U = s|V = s'] \\
&= \sum_{s \in \mathcal{P}^{-1}(s')} \frac{1}{|\mathcal{P}^{-1}(s')|} \log_2 \left|\mathcal{P}^{-1}(s')\right| \\
&= \left|\mathcal{P}^{-1}(s')\right| \cdot \frac{1}{|\mathcal{P}^{-1}(s')|} \log_2 \left|\mathcal{P}^{-1}(s')\right| \\
&= \log_2 \left|\mathcal{P}^{-1}(s')\right|
\end{aligned}$$

Since we are interested in how much information the program reveals *in general*, we consider the average conditional input entropy $\mathrm{H}[U|V]$ over all final states. Using the expression for $\mathrm{H}[U|V = s']$ derived above and the block sizes of the preimage

partition Π of I we obtain:

$$\begin{aligned}
\mathrm{H}[U|V] &= \sum_{s' \in \mathsf{F}} \underbrace{\Pr[V = s']}_{=0 \text{ for } s' \notin \mathsf{F}_{\mathsf{reach}}} \mathrm{H}[U|V = s'] \\
&= \sum_{s' \in \mathsf{F}_{\mathsf{reach}}} \frac{\left|\mathcal{P}^{-1}(s')\right|}{|\mathsf{I}|} \mathrm{H}[U|V = s'] \\
&= \sum_{s' \in \mathsf{F}_{\mathsf{reach}}} \frac{\left|\mathcal{P}^{-1}(s')\right|}{|\mathsf{I}|} \log_2 \left|\mathcal{P}^{-1}(s')\right| \\
&= \frac{1}{|\mathsf{I}|} \sum_{B \in \Pi} |B| \log_2 |B|
\end{aligned}$$

The last equation provides the connection between the quantitative (entropy) and qualitative (preimage partition) viewpoints. Since V is fully determined by U, the joint entropy $\mathrm{H}[U,V]$ of U and V equals the entropy $\mathrm{H}[U]$ of U, thus by the equation derived above, we also obtain

$$\mathrm{H}[U|V] = \mathrm{H}[U,V] - \mathrm{H}[V] = \mathrm{H}[U] - \mathrm{H}[V]$$

The information *leakage* $\mathcal{L}$ of a program is the *reduction in uncertainty* of the value of the input U gained by observing an output V:

$$\mathcal{L} := \mathrm{H}[U] - \mathrm{H}[U|V] = \mathrm{H}[V]$$

Using the expression for $\mathrm{H}[U|V]$ derived above, we can express $\mathcal{L}$ in terms of the block sizes of the preimage partition:

$$\mathcal{L} = \log_2 |\mathsf{I}| - \frac{1}{|\mathsf{I}|} \sum_{B \in \Pi} |B| \log_2 |B|$$

Example 3 (Information leakage)**.** Again, consider the program from Example 2 given in Fig. 2. Since the program maps values $x \leq 127$ to themselves and values $x \geq 128$ to 0, the preimage partition Π of $\mathcal{P}$ is given by

$$\Pi = \{\{s\} \mid s \in I \wedge 0 \leq s.x \leq 127\} \cup \{\{s \mid s \in I \wedge 128 \leq s.x \leq 255\}\}$$

Using this and $|\mathsf{I}| = 256$, we can compute the information leakage $\mathcal{L}$ of $\mathcal{P}$:

$$\begin{aligned}
\mathrm{H}[U|V] &= \frac{1}{|\mathsf{I}|} \cdot \sum_{B \in \Pi} |B| \cdot \log_2 |B| \\
&= \frac{1}{256} \cdot (128 \cdot (1 \cdot \log_2 1) + (127 \cdot \log_2 127)) \\
&= \frac{1}{256} \cdot (127 \cdot \log_2 127) \\
&\approx 3.47 \text{ bit} \\
\mathcal{L} &= \log_2 |\mathsf{I}| - \mathrm{H}[U|V] \approx 8 - 3.47 = 4.53 \text{ bit}
\end{aligned}$$

3 Approximation and randomization

In this section we utilize the formulation of information flow as reduction in Shannon entropy derived above and present an approximative static QIF analysis method from [3].

3.1 Approximation of information leakage

Computing the output entropy $\mathrm{H}[V]$ exactly requires computing the block sizes of the preimage partition Π. In practice, this is often infeasible, even for finite-state programs, due to the large number of program states. We will show that we can approximate the block sizes of Π. Using this approximation, we can derive bounds on $\mathrm{H}[V]$.

Approximation of the block sizes of the preimage partition Π

In order to approximate preimage partition block sizes of a program $\mathcal{P}$, we also need to approximate the input-output relation ρ_{IO}. In order to do this, we construct an *augmented program* $\overline{\mathcal{P}} = \left(\overline{\mathsf{S}}, \overline{\mathsf{I}}, \overline{\mathsf{F}}, \overline{\mathcal{T}}\right)$, such that reachable final states in $\mathcal{P}$ keep track of the corresponding initial states.

- $\overline{\mathsf{S}} := \mathsf{S} \times \mathsf{S}$
- $\overline{\mathsf{I}} := \mathsf{I} \times \mathsf{I}$
- $\overline{\mathsf{F}} := \mathsf{S} \times \mathsf{F}$
- $\overline{\mathcal{T}} := \{\overline{\tau} \mid \tau \in \mathcal{T}\}$, where $\rho_{\overline{\tau}} := \{((s'', s), (s'', s')) \mid (s, s') \in \rho_\tau \wedge s'' \in \mathsf{S}\}$

The program relation $\overline{\rho}$ and reachability can be defined analogously:

$$\overline{\rho} := \bigcup_{\overline{\tau} \in \overline{\mathcal{T}}} \rho_{\overline{\tau}} \qquad \begin{aligned} \overline{\mathsf{F}}_{\mathsf{reach}} &:= \left\{(s, s') \in \overline{\mathsf{F}} \mid s \in \mathsf{I}, s' \in \mathsf{F}, ((s, s), (s, s')) \in \overline{\rho}^*\right\} \\ &= \rho_{\mathsf{IO}} \end{aligned}$$

We can now apply the approximation methods described above to the augmented program. Using abstract interpretation, we compute the over-approximation $\overline{\mathsf{F}}_{\mathsf{reach}}{}^\sharp$ of $\overline{\mathsf{F}}_{\mathsf{reach}}$:

$$\overline{\mathsf{F}}_{\mathsf{reach}}{}^\sharp := \mathsf{lfp}\left(\alpha \circ \mathsf{Post}_{\overline{\rho}}, \alpha(\overline{\mathsf{I}})\right)$$

In order to compute the under-approximation $\overline{\mathsf{F}}_{\mathsf{reach}}{}^\flat$, we symbolically execute a given set of program paths $\{\pi_1, \ldots, \pi_n\}$ of $\overline{\mathcal{P}}$. We obtain the inclusions:

$$\overline{\mathsf{F}}_{\mathsf{reach}}{}^\flat \subseteq \overline{\mathsf{F}}_{\mathsf{reach}} \subseteq \overline{\mathsf{F}}_{\mathsf{reach}}{}^\sharp$$

This can be used to approximate the block sizes of the preimage partition Π of $\mathcal{P}$. Let $\overline{\mathcal{P}}^{\flat -1}(s'), \overline{\mathcal{P}}^{\sharp -1}(s')$ denote the sets of projections of each tuple – in $\overline{\mathsf{F}}_{\mathsf{reach}}{}^\flat$ and $\overline{\mathsf{F}}_{\mathsf{reach}}{}^\sharp$, respectively – to the first component:

$$\begin{aligned} \overline{\mathcal{P}}^{\flat -1}(s') &:= \left\{s \mid (s, s') \in \overline{\mathsf{F}}_{\mathsf{reach}}{}^\flat\right\} \subseteq \mathsf{I} \\ \overline{\mathcal{P}}^{\sharp -1}(s') &:= \left\{s \mid (s, s') \in \overline{\mathsf{F}}_{\mathsf{reach}}{}^\sharp\right\} \subseteq \mathsf{I} \end{aligned}$$

We thus have:

$$\overline{\mathcal{P}}^{\flat -1}(s') \subseteq \mathcal{P}^{-1}(s') \subseteq \overline{\mathcal{P}}^{\sharp -1}(s')$$

and followingly:

$$\left|\overline{\mathcal{P}}^{\flat -1}(s')\right| \leq \left|\mathcal{P}^{-1}(s')\right| \leq \left|\overline{\mathcal{P}}^{\sharp -1}(s')\right|$$

Approximation of input entropy

Using the approximations of $\mathcal{P}^{-1}$ obtained above, we can derive upper and lower bounds for the conditional input entropy $\mathrm{H}[U|V]$.

Lemma 1. *For $s' \in \mathsf{F}_{reach}$ we define:*

$$p_V{}^{\flat}(s') := \frac{1}{|I|} \max\left\{\left|\overline{\mathcal{P}}^{\flat -1}(s')\right|, 1\right\}$$
$$p_V{}^{\sharp}(s') := \frac{1}{|I|} \left|\overline{\mathcal{P}}^{\sharp -1}(s')\right|$$

Then $p_V{}^{\flat}$ and $p_V{}^{\sharp}$ under- and over-approximate p_V, i.e. for all $s' \in \mathsf{F}_{reach}$:

$$p_V{}^{\flat}(s') \leq \Pr[V = s'] \leq p_V{}^{\sharp}(s')$$

Proof. Let $s' \in \mathsf{F}_{\mathsf{reach}}$. Above we have shown that $\Pr[V = s'] = \left|\mathcal{P}^{-1}(s')\right| |\mathsf{I}|^{-1}$ and that $\left|\overline{\mathcal{P}}^{\flat -1}(s')\right|$ and $\left|\overline{\mathcal{P}}^{\sharp -1}(s')\right|$ approximate $\left|\mathcal{P}^{-1}(s')\right|$. Since $s' \in \mathsf{F}_{\mathsf{reach}}$, at least one initial state leads to s', thus $\left|\mathcal{P}^{-1}(s')\right| \geq 1$. The statement follows. □

Using the approximations obtained above, we can give lower and upper bounds on $\mathrm{H}[V]$. We show how to derive the lower bound:

$$\begin{aligned}
\mathrm{H}[V] &= \sum_{s' \in \mathsf{F}_{\mathsf{reach}}} \Pr[V = s'] \left(-\log_2 \Pr[V = s']\right) \\
&\geq \sum_{s' \in \mathsf{F}_{\mathsf{reach}}{}^{\flat}} \Pr[V = s'] \left(-\log_2 \Pr[V = s']\right) && \text{by } \mathsf{F}_{\mathsf{reach}}{}^{\flat} \subseteq \mathsf{F}_{\mathsf{reach}} \\
&\geq \sum_{s' \in \mathsf{F}_{\mathsf{reach}}{}^{\flat}} \Pr[V = s'] \left(-\log_2 p_V{}^{\sharp}(s')\right) && \text{Monotonicity of } -\log_2 \text{ on }]0,1] \\
&\geq \sum_{s' \in \mathsf{F}_{\mathsf{reach}}{}^{\flat}} p_V{}^{\flat}(s') \left(-\log_2 p_V{}^{\sharp}(s')\right)
\end{aligned}$$

The upper bound follows similarly and we obtain:

$$\begin{aligned}
&- \sum_{s' \in \mathsf{F}_{\mathsf{reach}}{}^{\flat}} p_V{}^{\flat}(s') \log_2 p_V{}^{\sharp}(s') \\
&\qquad \leq \mathrm{H}[V] \leq \\
&\qquad - \sum_{s' \in \mathsf{F}_{\mathsf{reach}}{}^{\sharp}} p_V{}^{\sharp}(s') \log_2 p_V{}^{\flat}(s')
\end{aligned}$$

Above we have shown:

$$\mathrm{H}[U|V] = \mathrm{H}[U] - \mathrm{H}[V]$$

Assuming U is uniformly distributed on I, we have $\mathrm{H}[U] = \log_2 |\mathsf{I}|$. Using the bounds on $\mathrm{H}[V]$ we can bound the value of $\mathrm{H}[U|V]$:

$$\begin{aligned}
&\log_2 |\mathsf{I}| + \sum_{s' \in \mathsf{F}_{\mathsf{reach}}{}^{\sharp}} p_V{}^{\sharp}(s') \log_2 p_V{}^{\flat}(s') \\
&\qquad \leq \mathrm{H}[U|V] \leq \\
&\qquad \log_2 |\mathsf{I}| + \sum_{s' \in \mathsf{F}_{\mathsf{reach}}{}^{\flat}} p_V{}^{\flat}(s') \log_2 p_V{}^{\sharp}(s')
\end{aligned}$$

3.2 Randomized approximation

While the approximation methods above are effective in more cases than exact computation, they require the *enumeration* of the over-approximation $\mathsf{F}_{\mathsf{reach}}{}^{\sharp}$ of the set $\mathsf{F}_{\mathsf{reach}}$ of reachable final states. This is frequently impractical due to the size of that set. We will show that we can avoid this enumeration.

Estimating entropy

Our goal is to construct an sampling-based algorithm that can compute upper and lower bounds for the conditional input entropy $\mathrm{H}[U|V]$. Since $\mathrm{H}[U|V] = \log_2|I| - \mathrm{H}[V]$, it is sufficient to approximate the entropy $\mathrm{H}[V]$ of V. Thus, as a first step, we show that we can construct an estimator for the entropy of a random variable.

Lemma 2. *Let $|\Omega| = m$, $X : \Omega_X \to \Omega, X \sim p_X$ a random variable, $0 < \delta \in \mathbb{R}$, random variables $X_1 \simeq \cdots \simeq X_n \simeq X$ independent instances (i.e.* samples*) of X. Then*

$$\Pr\left[\left|-\frac{1}{n}\sum_{i=1}^{n}\log_2\left(\Pr[X = X_i]\right) - H[X]\right| < \delta\right] > 1 - \frac{\log_2 m^2}{n\delta^2}$$

Proof. We define random variables $Y, Y_i : \Omega_X \to \mathbb{R}, Y \simeq Y_i$ for $i = 1 \ldots n$ by

$$Y_i = -\log_2\left(p_X(X_i)\right)$$

Each Y_i is an unbiased estimator for the entropy $\mathrm{H}[X]$ of X:

$$\mathbb{E}[Y_i] = \sum_{y\in\mathbb{R}} \Pr[Y_i = y]\cdot y = -\sum_{x\in\Omega}\Pr[X = x]\cdot\log_2\left(p_X(x)\right) = \mathrm{H}[X]$$

Consider a random variable Z that is the average of the Y_i:

$$Z := \frac{1}{n}\sum_{i=1}^{n} Y_i$$

By linearity of the expectation value, $\mathbb{E}[Z] = \mathrm{H}[X]$. For the variance $\mathrm{Var}[Z]$ of Z we obtain:

$$\mathrm{Var}[Z] = \frac{1}{n}\mathrm{Var}[Y]$$

In [2] it is shown that $\mathrm{Var}[Y] \le (\log_2 m)^2$, thus

$$\mathrm{Var}[Z] \le \frac{\left(\log_2(m)\right)^2}{n}$$

Let $0 < \delta \in \mathbb{R}$. We can apply Chebyshev's inequality:

$$\Pr\left[\,|Z - \mathbb{E}[Z]| \ge \delta\right] \le \frac{\mathrm{Var}[Z]}{\delta^2} \le \frac{(\log_2 m)^2}{n\delta^2}$$

Using $\mathbb{E}[Z] = \mathrm{H}[X]$ and unfolding the definitions of Z and Y_i, we obtain:

$$\Pr\left[\left|-\frac{1}{n}\sum_{i=1}^{n}\log_2\left(p_X(X_i)\right) - \mathrm{H}[X]\right| < \delta\right] > 1 - \frac{(\log_2 m)^2}{n\delta^2}$$

□

We have shown that, given the probability distribution of a random variable X, we can use a sampling estimator Z to bound the entropy $\mathrm{H}[X]$ of X. The confidence of $p = 1 - \frac{(\log_2 m)^2}{n\delta^2}$ for $\mathrm{H}[X] \in]Z - \delta, Z + \delta[$ allows us to judge the quality of the estimation. Conversely, for a desired confidence $p \in [0, 1[$ and precision $\delta \in]0, \infty[$ we can give the minimum number $n = \left\lceil \frac{(\log_2 m)^2}{(1-p)\delta^2} \right\rceil$ of required samples.

A randomized algorithm

In order to bound the output entropy $\mathrm{H}[V]$ for output samples $X_i \simeq V$ using the estimator Z derived above, we would need to know the exact probability distribution of V. This still requires the exact computation of preimage sizes. Our goal is to avoid this. Instead, we can apply the approximations ${p_V}^\sharp$ and ${p_V}^\flat$ of the probability distribution p_V of V from the previous section. In terms of the estimators Y_i and Z we derived above, we construct the following over- and under-approximations:

$$\begin{aligned} Y_i^\flat &:= -\log_2\left({p_V}^\sharp(X_i)\right) && \leq Y_i \leq && Y_i^\sharp := -\log_2\left({p_V}^\flat(X_i)\right) \\ Z^\flat &:= \frac{1}{n}\sum_{i=1}^{n} Y_i^\flat && \leq Z \leq && Z^\sharp := \frac{1}{n}\sum_{i=1}^{n} Y_i^\sharp \end{aligned}$$

We obtain the following bounds on $\mathrm{H}[V]$:

$$Z^\flat - \delta \leq Z - \delta < \mathrm{H}[V] < Z + \delta \leq Z^\sharp + \delta$$

Bounding $\mathrm{H}[U|V]$ using the bounds on $\mathrm{H}[V]$ we obtain:

$$\begin{aligned} \mathrm{H}^\flat := \log_2|\mathsf{I}| - Z^\sharp \leq & \mathrm{H}[U|V] + \delta \\ \mathrm{H}[U|V] - \delta \leq & \log_2|\mathsf{I}| - Z^\flat =: \mathrm{H}^\sharp \end{aligned}$$

By Lemma 2 these bounds are valid with a probability $p > 1 - \frac{(\log_2|\mathsf{I}|)^2}{n\delta^2}$. We can simplify the approximations:

$$\begin{aligned} \mathrm{H}^\flat = \log_2|\mathsf{I}| - Z^\sharp &= \log_2|\mathsf{I}| - \frac{1}{n}\sum_{i=1}^{n} -\log_2\left({p_V}^\flat(X_i)\right) \\ &= \log_2|\mathsf{I}| + \frac{1}{n}\sum_{i=1}^{n}\log_2\left(\frac{\left|\overline{\mathcal{P}}^{\flat -1}(X_i)\right|}{|\mathsf{I}|}\right) \\ &= \frac{1}{n}\sum_{i=1}^{n}\log_2\left(\min\left\{1, \left|\overline{\mathcal{P}}^{\flat -1}(X_i)\right|\right\}\right) \\ \mathrm{H}^\sharp = \log_2|\mathsf{I}| - Z^\flat &= \log_2|\mathsf{I}| - \frac{1}{n}\sum_{i=1}^{n} -\log_2\left({p_V}^\sharp(X_i)\right) \\ &= \log_2|\mathsf{I}| + \frac{1}{n}\sum_{i=1}^{n}\log_2\left(\frac{\left|\overline{\mathcal{P}}^{\sharp -1}(X_i)\right|}{|\mathsf{I}|}\right) \\ &= \frac{1}{n}\sum_{i=1}^{n}\log_2\left(\left|\overline{\mathcal{P}}^{\sharp -1}(X_i)\right|\right) \end{aligned}$$

The estimation procedure derived above is formalized in Algorithm 2.

Algorithm 2 RANT – *R*andomized qu*ant*ification

procedure RANT($\mathcal{P}, p, \delta$):
  $(\mathrm{H}^\flat, \mathrm{H}^\sharp, n) \leftarrow (0, 0, \left\lceil \frac{(\log_2|\mathsf{I}|)^2}{(1-p)\delta^2} \right\rceil)$
  for $i = 1 \ldots n$ **do**
    $s_i \leftarrow$ pick randomly from I
    $(\pi_i, s') \leftarrow$ execute $\mathcal{P}$ symbolically on s_i
    $\mathrm{H}^\flat \leftarrow \mathrm{H}^\flat + \log_2\left|\rho_{\pi_i}^{-1}(s')\right|$
    $\mathrm{H}^\sharp \leftarrow \mathrm{H}^\sharp + \log_2\left|\overline{\mathcal{P}}^{\sharp -1}(s')\right|$
  return $(\frac{\mathrm{H}^\flat}{n}, \frac{\mathrm{H}^\sharp}{n})$

Theorem 2. *For* $(H^\flat, H^\sharp) = \textsc{Rant}(\mathcal{P}, p, \delta)$ *the following holds:*

$$\Pr\left[H^\flat - \delta \leq H[U|V] \leq H^\sharp + \delta\right] \geq p$$

Proof. Rant computes the over-approximation $\mathrm{H}^\sharp$ and under-approximation $\mathrm{H}^\flat$ of the entropy estimator $H = \log_2 |\mathsf{I}| - Z$ for $n = \left\lceil \frac{(\log_2 |\mathsf{I}|)^2}{(1-p)\delta^2} \right\rceil$ samples. The statement follows from Lemma 2 and the construction of $\mathrm{H}^\sharp$ and $\mathrm{H}^\flat$. □

4 Approximative dynamic analysis

The methods presented in the previous sections are *static* analysis methods – they bound the information leakage of a program for all possible executions. In this section, we present a *dynamic* analysis method described in [5]. The goal is to provide a sound upper bound on the information leakage of a *single execution*. Additionally, the method provides for a way of combining the results of several single-execution analyses to obtain more general bounds.

4.1 Assembly programs and computations

We consider assembly-like programs defined by the grammar shown in Fig. 3. The element reg represents a register identifier, addr: reg represents the usage of the value of a register as a memory address. A register identifier on the left-hand side of an assignment represents the register as a variable, a register identifier on the right-hand side represents its value. The **input** instruction writes a (secret) input value to a register, **output** makes the contents of a register public.

$$\begin{array}{rcl}
r \in \mathsf{reg} & ::= & \mathsf{r}_n : n \in \mathbb{N} \\
c \in \mathsf{cst} & ::= & n \in \mathbb{N} \\
\circ \in \mathsf{binop} & ::= & + \mid \textbf{and} \mid \textbf{xor} \\
i \in \mathsf{instr} & ::= & \mathsf{reg} := \mathsf{cst} \\
& \mid & \mathsf{reg} := \mathsf{reg} \\
& \mid & \mathsf{reg} := \mathsf{reg} \circ \mathsf{reg} : \circ \in \mathsf{binop} \\
& \mid & \mathsf{reg} := \textbf{load}\ (\mathsf{addr} : \mathsf{reg}) \\
& \mid & \textbf{store}\ (\mathsf{addr} : \mathsf{reg})\ \mathsf{reg} \\
& \mid & \textbf{output}\ \mathsf{reg} \\
& \mid & \textbf{input}\ \mathsf{reg} \\
& \mid & \textbf{if}\ \mathsf{reg} \geq 0\ \ \textbf{goto}\ \mathsf{cst} \\
p \in \mathsf{program} & ::= & \epsilon \mid \mathsf{instr}\ \mathsf{program}
\end{array}$$

Figure 3: Syntax for a minimal imperative language

4.2 Dynamic tainting and bit-tracking

Dynamic tainting analysis computes the secrecy status for each piece of data occurring during program execution. A piece of data is considered *tainted* if it contains secret information. The amount of information leaked by the program is bounded by the number of secret bits present in the program's output. Tainting can be computed on various levels of granularity – variables, bytes and individual bits could

be marked as secret. Once a piece of data is marked as secret, any operation on it must be instrumented to compute the secrecy status of the result. In the case of bit-level tainting, this is referred to as bit-tracking. Each register and memory location is associated with a *secrecy bit mask* which specifies which of its bits are marked as secret. Examples of bit-tracking formulas for several bit-level operations are given in [5].

4.3 Flow graphs

In order to model dynamic QIF analysis as a maximum-flow problem, we construct a directed acyclic graph, which we refer to as the *flow graph*. The flow graph has potentially secret pieces of information as its vertices (*nodes*). The edges connecting the nodes have capacities that model how much secret information can flow from one piece of data to another.

The edge capacities are additionally bounded by a bit-tracking analysis. Each vertex is associated with the secrecy bit mask of the piece of data it represents. An edge originating from a vertex v can carry at most as many bits of secret information as are set in the secrecy bit mask of v.

The introduction of secret information through **input** instructions is modeled as an edge from a special *source node* to a new node representing the input data. The revelation of potentially secret information by **output** instructions is modeled as an edge to a special *sink node*. The *information leakage* of a program is bounded by the maximum flow from the source node to the sink node.

The graph is constructed dynamically by keeping track of the source of each piece of data during an execution. In the algorithm InstrFlow presented below, this is implemented by mapping each register and memory location to a vertex of the flow graph and updating this mapping upon any modification of data. An example of a flow graph for a specific program execution is given in Fig. 4.

Combining flow graphs

Flow graphs corresponding to different executions of a program can be merged to obtain bounds that are valid for several executions. This is done by identifying edges corresponding to a particular program location in all graphs and merging the source nodes and destination nodes to obtain two nodes and one edge for each set of original edges and nodes. The capacity of the merged edge is the sum of the capacities of the original edges. Finally, the source and sink nodes of all graphs are merged to obtain new sink and source nodes.

Implicit and explicit flows

An *explicit flow* of secret information is one that occurs by copying or mapping over secret data (for example in the assignment $r_i := r_j \circ r_k$, where either of the operands is secret, secret information explicitly flows from the operands to r_i). An *implicit flow* occurs when secret information is revealed indirectly. In the language defined above, this applies to case distinctions over the contents of registers containing secret data (**if** $r_i \geq 0$ **goto** c).

We over-approximate the secret information flow for the conditional **goto** as an additional 1 bit of flow to any of the following **output** statements. This captures the intuition that the result of the comparison might affect the value of any publicly visible register. We account for implicit flows by adding a chain of "leak nodes", to which implicit flows from comparisons on registers are directed.

On each **output** operation, a new node representing the output information is created and an edge from the current leak node to the output node is added. This

0 : **input** r_1
1 : **input** r_2
2 : **if** $r_2 \geq 0$ **goto** 4
3 : **output** r_2
4 : $r_3 := r_2 + r_1$
5 : **output** r_3

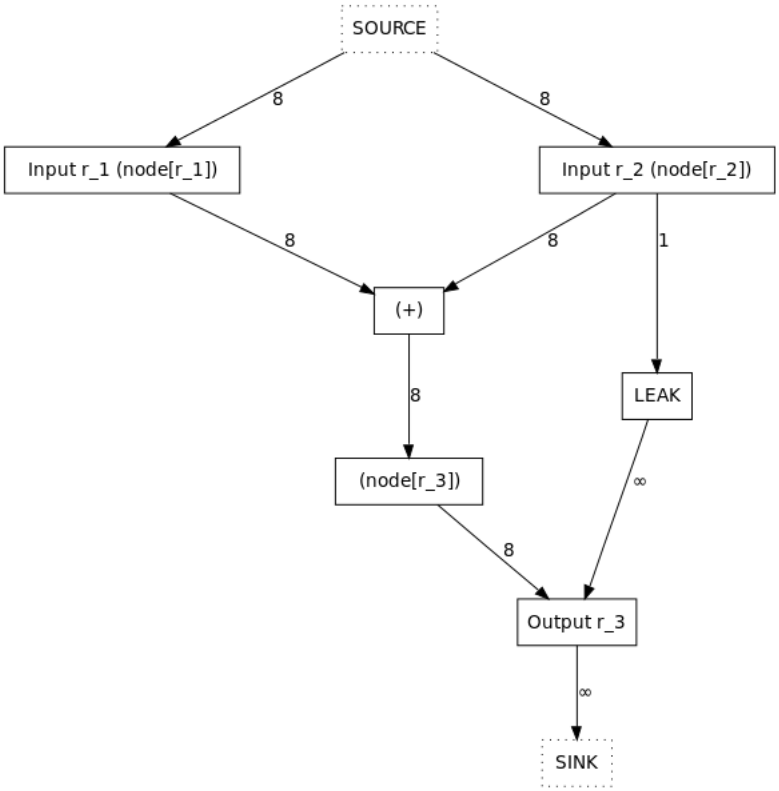


Figure 4: Example program (using 8-bit registers) and the flow graph corresponding to an execution with inputs (-1,0). The upper bound on information leakage, computed as maximum flow, is 9 bits. The labels "node[r_i]" refer to the node map explained in the presentation of the construction algorithm.

models the possibility that a previous comparison on secret data might have an effect on the output value. A new leak node is created and an unbounded edge is added between the old and new leak node to allow implicit flows to propagate to later outputs.

Instrumentation for flow graph construction

The algorithm InstrFlow for incremental construction of the flow graph is presented in Algorithm 3. The program is instrumented such that InstrFlow is applied to every instruction encountered. For each register and memory location, we keep track of the source of the information it contains. The map node associates a register $r_i \in \mathsf{reg}$ or memory location $(\mathsf{addr} : m) \in \mathsf{addr}$ with its node (v, bits). The node contains the vertex $v \in V$ in the flow graph and the secrecy bitmask $\mathsf{bits} \in (\mathbb{N}_0 \bmod 2^{\mathsf{regbits}})$ for its contents.

When a value is to a register, the register's node mapping is set to the node of the rhs value. If a register or memory location is modified, a new node is created. The node n corresponding to the original information is connected to the node n' representing the modified data and the node map is updated accordingly. The capacity c of the new edge $n \to^c n'$ is at most the number $\#(n.\mathsf{bits})$ of set bits in the secret bit mask of its source node. Implicitly, a new edge $n \to^c n'$ represents a new edge $n.v \to^c n'.v$. Binary operations are treated analogously. Source, sink and leak nodes are treated as special cases: edges between leak nodes as well as edges to the

sink node are unbounded, edges from leak nodes to output nodes have a capacity of 1.

Enclosure regions

Additionally to the algorithm InstrFlow presented above, the method in [5] implements *enclosure regions*, specified by source code annotations. An enclosure region denotes the restriction of implicit flows within a part of the source code to explicitly specified "output locations". Implicit flows occurring within an enclosure region can only affect the values of the region's defined output locations. When the enclosure annotations are correct, this method can improve precision while still retaining the soundness of the upper bound on information leakage.

Conclusion

We have presented exact and approximative static methods as well as an approximative dynamic method to quantify information flow properties of programs. While the exact static analysis method [1] yields precise results, it generally cannot be applied in practice due to prohibitive computational cost.

The approximative dynamic analysis method [5] only produces an upper bound on the information leaked by a program for a finite set of program paths, but is computationally cheap and can be applied to pre-compiled binaries.

Randomized approximative static analysis (as presented in [3]) generates upper and lower bounds on information leakage over all paths. Additionally, the method allows a controlled cost-versus-precision trade-off. By providing statistical quality guarantees, it allows the user to judge the tightness and validity of the generated bounds to enable meaningful decisions with respect to an information flow policy.

References

[1] Michael Backes, Boris Köpf, and Andrey Rybalchenko. Automatic Discovery and Quantification of Information Leaks. In *Proc. IEEE Symp. on Security and Privacy (S&P '09)*, pages 141–153. IEEE, 2009.

[2] Tugkan Batu, Sanjoy Dasgupta, Ravi Kumar, and Ronitt Rubinfeld. The complexity of approximating entropy. In *Proc. ACM Symp. on Theory of Computing (STOC '02)*, pages 678–687. ACM, 2002.

[3] Boris Köpf and Andrey Rybalchenko. Approximation and randomization for quantitative information-flow analysis. In *Proceedings of the 23rd IEEE Computer Security Foundations Symposium, CSF 2010, Edinburgh, United Kingdom, July 17-19, 2010*, pages 3–14, 2010.

[4] Pasquale Malacaria. Assessing security threats of looping constructs. In *Proc. Symp. on Principles of Programming Languages (POPL '07)*, pages 225–235. ACM, 2007.

[5] Stephen McCamant and Michael D. Ernst. Quantitative information flow as network flow capacity. In *Proc. ACM Conf. on Programming Language Design and Implementation (PLDI '08)*, pages 193–205. ACM, 2008.

[6] Claude E. Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27:379–423 and 623–656, July and October 1948.

[7] Geoffrey Smith. On the foundations of quantitative information flow. In *Proc. Intl. Conf. of Foundations of Software Science and Computation Structures (FoSSaCS '09)*, LNCS 5504, pages 288–302. Springer, 2009.

Algorithm 3 InstrFlow(i ∈ instr)

vars
  const regbits ∈ ℕ : length of a register in bits
  global V ⊇ {sink, source, public} , E ⊆ V² × ℕ₀ : nodes, edges with capacities
  global leak ∈ V
  global node ⊆ (reg ∪ addr) × V × (ℕ₀ mod 2^{regbits})
end vars
begin
  match i **with**
  | $\boxed{r_i := c}$ →
    node[r_i] := public
  | $\boxed{r_i := r_j}$ →
    node[r_i] := node[r_j]
  | $\boxed{r_i := r_j \circ r_k}$ →
    begin
      m, n := fresh nodes
      n.bits := m.bits := opBits($(\circ), r_j, r_k$)
      c_1 := #(node[r_j].bits)
      c_2 := #(node[r_k].bits)
      c_3 := #(n.bits)
      E := E ∪ {(node[r_j] →^{c_1} n), (node[r_k] →^{c_2} n), (n →^{c_3} m)}
      node[r_i] := m
    end
  | $\boxed{\textbf{input } r_i}$ →
    begin
      n := fresh node with n.bits = $(1^{\text{regbits}})_2$
      node[r_i] := n
      E := E ∪ {(source →$^{\text{regbits}}$ n)}
    end
  | $\boxed{\textbf{output } r_i}$ →
    begin
      n, l := fresh nodes
      c := #(node[r_i].bits)
      E := E ∪ {(node[r_i] →^{c} n), (leak →$^{\infty}$ n), (leak →$^{\infty}$ l), (n →$^{\infty}$ sink)}
      leak := l
    end
  | $\boxed{\textbf{if } r_i \geq 0 \textbf{ goto } loc}$ →
    E := E ∪ {(node[r_i] →^{1} leak)}
  | $\boxed{\textbf{store } (\text{addr} : r_a)\ r_v}$ →
    node[addr : r_a] := node[r_v]
  | $\boxed{r_i := \textbf{load } (\text{addr} : r_a)}$ →
    node[r_i] := node[addr : r_a]

end.