

Automatic Discovery and Quantification of Information Leaks

Michael Backes
Saarland University and MPI-SWS
backes@cs.uni-sb.de

Boris Köpf
MPI-SWS
bkoepf@mpi-sws.org

Andrey Rybalchenko
MPI-SWS
rybal@mpi-sws.org

Abstract

Information-flow analysis is a powerful technique for reasoning about the sensitive information exposed by a program during its execution. We present the first automatic method for information-flow analysis that discovers what information is leaked and computes its comprehensive quantitative interpretation. The leaked information is characterized by an equivalence relation on secret artifacts, and is represented by a logical assertion over the corresponding program variables. Our measurement procedure computes the number of discovered equivalence classes and their sizes. This provides a basis for computing a set of quantitative properties, which includes all established information-theoretic measures in quantitative information-flow. Our method exploits an inherent connection between formal models of qualitative information-flow and program verification techniques. We provide an implementation of our method that builds upon existing tools for program verification and information-theoretic analysis. Our experimental evaluation indicates the practical applicability of the presented method.

1. Introduction

Information-flow analysis keeps track of sensitive information that is processed by a program during its execution. One of the main goals of the analysis is to check whether any sensitive information is exposed to the environment. When information is leaked, the analysis needs to qualitatively and quantitatively assess the extent of the leak.

The existing approaches to information-flow analysis provide a variety of techniques for dealing with the disclosure of information, see [35]. Several approaches deal with the qualitative aspect of information-flow analysis [1], [5], [13], [18], [34], which is usually formalized by an equivalence relation over secret artifacts manipulated by the program. Security guarantees correspond to the (im)possibility of distinguishing between secret artifacts by observing program behaviors. Existing quantitative approaches characterize the magnitude of information leaks, e.g. in terms of the number of secret bits that are revealed [9], [17], or in terms of the rate at which information can be transmitted through the leak [26].

Unfortunately, the applicability of the existing information-flow analyses suffers from several limitations. The qualitative approaches assume that the equivalence relation is supplied manually; however, such relations are notoriously difficult to find due to the complexity of reasoning about how the program treats its secrets. On the quantitative side, the estimates computed by the existing approaches mostly deal with the number of leaked bits, e.g. [10], [29], which is not sufficient for establishing comprehensive security guarantees. For example, a security analysis might require a measure for the number of attempts that are needed to identify a secret value, bounds on the throughput of the program if it is used as an unwanted communication channel, or a combination of several such measures.

In this paper, we present the first automatic method for information-flow analysis that addresses these challenges. Our method delivers a complete analysis that automatically discovers the leaked information, determines its information-theoretic characteristics, and computes a comprehensive set of quantitative properties.

The leaked information is computed in the form of an equivalence relation and is represented by a logical assertion over program variables. The equivalence relation computed by our method is precise, i.e., describes only the information that is leaked, and can be used on its own, e.g., for declassification policies [4]. Our method goes beyond this qualitative characterization, and uses the assertion as an input to a measurement procedure that computes the number of discovered equivalence classes and their sizes. We demonstrate how these data provide a basis for computing a set of quantitative properties, which is comprehensive in the sense that it includes all information-theoretic measures that are commonly considered in the literature on quantitative information flow, i.e., Shannon entropy, guessing entropy, min-entropy, and channel capacity.

Our method exploits an inherent connection between qualitative information-flow and program verification techniques. The desired equivalence relation can be viewed as a precondition for safe execution of the program under consideration augmented with a ‘shadow’ copy and an assertion checking the information leakage. The assertion fails whenever the shadow copy of the program exhibits a behavior that witnesses the ability to distinguish equivalent

artifacts. Our method iteratively constructs the equivalence relation. It starts with a coarse initial candidate that claims no leakage of sensitive information, and incrementally weakens the claim by refining the candidate relation, until there is no witness of further leaks. The key to the automatic construction is the successive exclusion of counterexamples witnessing inadequacy of the current equivalence.

We identify the characteristics of the equivalence relation that provide the common basis for computing various entropy measures, as required by the quantitative analysis. These characteristics consist of the number of equivalence classes and their sizes, and can be further refined by probability distributions inside the equivalence classes. We show how, given these characteristics, one can compute the average uncertainty about the secret in bits (Shannon entropy), the average number of guesses that are needed to identify secrets (conditional and minimal guessing entropy, respectively), and the maximal rate at which information can be transmitted using the program as a communication channel (channel capacity). Finally, we present a procedure for computing these characteristics for a given equivalence relation.

The presentation of our method is aligned with the existing body of research on program verification and symbolic reasoning. It suggests a basis for an automatic tool that can be built by utilizing existing software model checkers, quantifier elimination algorithms and solution counting techniques. We use these components in a black-box fashion, hence our tool will immediately benefit from the development of the state-of-the-art in the respective areas.

We have implemented the presented method and have successfully applied it to analyze a series of example programs: a password checker, an electronic purse, a sum query, and an electronic auction. For each program, we determined the equivalence relations representing the leaked information and computed the sizes of the equivalence classes together with different information-theoretic interpretations.

In summary, our main contribution is the first automatic method for information-flow analysis that discovers what information is leaked and computes its comprehensive quantitative interpretation.

Outline. The paper is structured as follows. We present related work in the remainder of this section. In Section 2, we illustrate how our method applies to an example program. We give the basic definitions in Section 3. In Section 4, we present our method in abstract terms, before we outline its implementation in Section 5. We present experimental results in Section 6.

Related work. For an overview of language-based approaches to information-flow security, refer to [33]; for an overview on declassification, see [35].

The use of equivalence relations to characterize partial information flow was proposed in [13] and further explored in [5], [18], [41]. Several approaches use equivalence relations to specify downgrading assertions within information flow type systems [4], [34]. Our method can be used to synthesize such assertions. The idea that secure information flow can be verified by analyzing pairs of program runs can be found in [5], [16], [23], [38], [39].

Early approaches for quantifying information flow focus on the capacity of covert channels between processes in multi-user systems [20], [30], [40] rather than on information flow in programs. The first approach to connect information theory to program analysis is [17].

A type system for statically deriving quantitative bounds on the information that a program leaks is presented in [9], [10]. The analysis is based on Shannon entropy and an observer that can see, but not influence, the public inputs to the program. Our method accommodates a variety of information measures and captures attackers that can interact with the program by providing inputs.

The information leakage of loops can be characterized in terms of the loop's output and the number of iterations [27]. In our model, the information that is revealed by the number of loop iterations can be captured by augmenting loops with observable counters. For given upper bounds on the number of iterations, our method can be used to automatically determine this information.

Information-theoretic bounds in terms of the number of program executions are presented in [24]. The algorithms for computing these bounds for a concrete system rely on an enumeration of the entire input space, and it is not yet clear how the analysis scales to larger systems.

The model in [12] captures an attacker's belief about a secret, which may also be wrong. Reasoning about beliefs is out of the scope of entropy-based measures, such as the ones used in this paper. One advantage of entropy-based measures is the direct connection to equivalence relations, which makes them amenable to automated reasoning techniques. To the best of our knowledge, our method is the first static, quantitative analysis that has been implemented.

An automatic dynamic quantitative information flow analysis method is presented in [29]. The method enables one to derive tight bounds on the information flow in individual program runs, but does not yield bounds on the maximal information that a program can leak, which is important for security analysis.

2. Illustrative example

To illustrate our method and the kind of results that it provides, we show how it applies to an example program.

We consider an electronic sealed-bid auction in which bidders want their bids to remain confidential and the winner is publicly announced. The announcement of the winner

reveals partial information about the individual bids, e.g., about their ordering.

This kind of electronic auction can be implemented as a program P that takes as input n secret bids h_1, \dots, h_n and outputs the winner of the auction in a variable l , i.e., upon termination, $l = i$ such that $h_i = \max\{h_1, \dots, h_n\}$.

```

int l=0;
for (int i=0; i<n; i++){
  if (h[i]>h[l])
    l=i;
}

```

In the first step, we deduce an equivalence relation R on the set of possible secret inputs to express what an attacker can learn about the input by observing the program's output: two inputs are in the same equivalence class whenever the program produces the same result on both inputs. By observing the output of the program, the attacker can then only deduce the secret input up to its R -equivalence class. We cast such equivalence relations R as formulas over pairs of secret inputs.

Checking that a program leaks no more secret information than what is specified by R can be cast as a reachability problem on two independent instances of the program, and it can be solved using off-the-shelf model-checkers, such as BLAST [21], SLAM [3], SATABS [11], and ARMC [31]. If the check fails (i.e., if the program leaks more information), the model checker produces a counterexample: it returns two program paths π and η along which two R -related inputs produce different outputs.

Guided by this counterexample, we refine the relation R to R' , such that R' distinguishes between all secret inputs that lead to different observable outputs along the paths π and η . We iterate this refinement process until we have found a relation R for which the check fails; this R is a logical characterization of the maximal information that the program can leak.

For our auction program and $n = 3$, this iterative refinement process yields the relation

$$\begin{aligned}
R \equiv & (h_1 < h_3 \wedge h_2 < h_3 \wedge \bar{h}_1 < \bar{h}_3 \wedge \bar{h}_2 < \bar{h}_3) \\
& \vee (\bar{h}_1 < \bar{h}_3 \wedge \bar{h}_3 \leq \bar{h}_2 \wedge h_1 < h_2 \wedge h_3 \leq h_2) \\
& \vee (\bar{h}_3 \leq \bar{h}_1 \wedge \bar{h}_1 < \bar{h}_2 \wedge h_1 < h_2 \wedge h_3 \leq h_2) \\
& \vee (h_2 < h_3 \wedge h_3 \leq h_1 \wedge \bar{h}_2 \leq \bar{h}_1 \wedge \bar{h}_3 \leq \bar{h}_1) \\
& \vee (h_3 \leq h_2 \wedge h_2 \leq h_1 \wedge \bar{h}_2 \leq \bar{h}_1 \wedge \bar{h}_3 \leq \bar{h}_1) ,
\end{aligned}$$

which represents a set of pairs $((h_1, h_2, h_3), (\bar{h}_1, \bar{h}_2, \bar{h}_3))$ of triples of input variables, i.e., a binary relation. Here \bar{h}_1, \bar{h}_2 and \bar{h}_3 denote the secret bids that are input to the second instance of the program. Our specific implementation uses the model-checker ARMC, which does not cover arrays. To analyze the auction program, we unfold the loop and replace each array element $h[i]$ by a variable h_i . Note that this

is a limitation of our implementation rather than one of our method.

In the second step, we determine the R -equivalence classes. To this end, we pick an arbitrary vector of bids (say, $(0, 0, 0)$) and use it to instantiate the variables $\bar{h}_1, \bar{h}_2, \bar{h}_3$ of R . In this way, we obtain a formula B_1 over the variables h_1, h_2, h_3 that represents all the bids that are R -equivalent to $(0, 0, 0)$, i.e., the R -equivalence class of $(0, 0, 0)$. Then we pick a representative of another equivalence class, i.e. a model of $\neg B_1$ and repeat the procedure. We proceed in this way until we have enumerated all equivalence classes, i.e., until $B_1 \vee \dots \vee B_r \equiv \top$. The Omega-calculator [32] is a tool for manipulating formulas in Presburger Arithmetic (i.e., linear arithmetic with quantifiers) using various operations, such as computing models, relational composition, and set difference, and we use it to implement the enumeration of the equivalence classes. For our auction example, we obtain the following equivalence classes:

$$\begin{aligned}
B_1 & \equiv h_2 \leq h_1 \wedge h_3 \leq h_1 , \\
B_2 & \equiv h_1 < h_3 \wedge h_2 < h_3 , \\
B_3 & \equiv (h_1 < h_3 \wedge h_3 \leq h_2) \vee (h_3 \leq h_1 \wedge h_1 < h_2) ,
\end{aligned}$$

where the clauses of B_3 are exclusive.

In the third step, we use LATTÉ (Lattice point Enumeration) [25], a tool for computing the number of integer solutions of linear arithmetic constraints, for counting the size of each of the equivalence classes. For this, we have to add additional constraints to bound the size of the input domain. For our auction program, we choose as inputs positive integers of 32 bits, i.e. $0 \leq h_1, h_2, h_3 \leq 2^{32} - 1$.

Then LATTÉ determines the following sizes:

$$\begin{aligned}
|B_1| & = 26409387513978151235418587136 , \\
|B_2| & = 26409387495531407161709035520 , \\
|B_3| & = 26409387504754779196416327680 .
\end{aligned}$$

The result shows that the three equivalence classes B_1, B_2, B_3 are of almost equal size; the difference is due to the asymmetry with which our program determines the winner of an auction with two or more equal bids. If the input values are independently and uniformly chosen (modeled by a random variable \mathcal{U}), the attacker's initial uncertainty about the auction is $H(\mathcal{U}) = 3 \cdot 32 = 96$ bits. Observing the output of the program reduces this uncertainty to

$$H(\mathcal{U}|\mathcal{V}_R) = \frac{1}{2^{96}} \sum_{i=1}^3 |B_i| \log |B_i| \approx 94.42 ,$$

which corresponds to an information leakage (a reduction in uncertainty) of $96 - 94.2 = 1.58$ bits.

Existing quantitative approaches will return (an approximation of) this number as a result. Our approach additionally offers quantities that go beyond the number of bits that are leaked. We can, for example, answer questions about the

average number of guesses required to correctly determine the secret inputs after observing the output ($1.3204 \cdot 10^{28}$), the average number of guesses required to determine the weakest secrets ($1.3204 \cdot 10^{28}$, as the equivalence classes in our examples are almost of equal size), or simply the number of possible bid combinations that lead to a given output ($|B_1|$, $|B_2|$, and $|B_3|$). As we will show, all of these measures can be easily derived from the sizes $|B_1|$, $|B_2|$, $|B_3|$ of the R -equivalence classes computed by our approach.

3. Preliminaries

In this section, we give the necessary definitions for dealing with programs and information-flow analysis.

3.1. Programs and computation

We model a program P as a transition system (S, \mathcal{T}, I, F) that consists of

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

Our exposition does not assume any further state structure; however, for the sake of concreteness we point out that usually a program state represents a valuation of program variables in scope, and a program transition corresponds to a program statement as written in a programming language.

A program *computation* σ is a sequence of program states s_1, \dots, s_n that starts at an initial state, ends at a final state, and relates each pair of consecutive states by some program transition. Formally, we require that $s_1 \in I$, $s_n \in F$, and for each $1 \leq i < n$ there exists a transition $\tau \in \mathcal{T}$ such that $(s_i, s_{i+1}) \in \rho_\tau$.

A program *path* π is a (non-empty) sequence of program transitions, i.e., $\pi \in \mathcal{T}^+$. We write $\pi \cdot \tau$ to denote a path obtained by extending π using a transition τ . Given two transition relations ρ_1 and ρ_2 , we define their *relational composition* $\rho_1 \circ \rho_2$ as usual:

$$\rho_1 \circ \rho_2 \equiv \{(s, s') \mid \exists s'' \in S : (s, s'') \in \rho_1 \wedge (s'', s') \in \rho_2\}.$$

Given a path $\pi = \tau_1 \cdot \dots \cdot \tau_n$, a *path relation* ρ_π consists of the relational composition of transition relations along the path, i.e.,

$$\rho_\pi \equiv \rho_{\tau_1} \circ \dots \circ \rho_{\tau_n}.$$

A program path π is *feasible* if the corresponding path relation is not empty, i.e., $\rho_\pi \neq \emptyset$.

We assume that initial and final states are pairs consisting of *low* and *high* components, i.e., $I = I_{hi} \times I_{lo}$ and $F = F_{hi} \times F_{lo}$. We assume an observer that knows the program, i.e., the corresponding transition system, and can

see the low components of the initial and final states of a given computation. That is, the observer cannot see the high components of the initial and final states, and it cannot see any intermediate states of the computation. The later condition explains why we assume the high/low structure only on the initial and final states.

3.2. Qualitative information flow

We use an equivalence relation R over I_{hi} , i.e., $R \subseteq I_{hi} \times I_{hi}$, to characterize the information that is leaked to an observer. R represents the observer knowledge in terms of equivalence classes. After observing a program computation the observer only knows that the high component of the input state belongs to the set $[s_{hi}]_R$. If R is the identity relation, i.e.,

$$=_{hi} \equiv \{(s_{hi}, s_{hi}) \mid s_{hi} \in I_{hi}\},$$

then the observer knows the value s_{hi} , since the equivalence class $[s_{hi}]_{=_{hi}}$ is a singleton set and hence uniquely determines s_{hi} . In contrast, the largest equivalence relation All_{hi} that does not distinguish between any states, i.e., $All_{hi} = I_{hi} \times I_{hi}$, captures that the observer knows nothing about I_{hi} , since we have $[s_{hi}]_{All_{hi}} = I_{hi}$. An equivalence relation R such that $=_{hi} \subset R \subset All_{hi}$ represents a partial knowledge about the high component of the input.

The information that a program leaks partially depends on the low component of the initial states. We call such a low component of an initial state an *experiment*, and assume that it can be chosen by the attacker. Our goal is to characterize the secret information that a program leaks when it is run on a given set of experiments $E \subseteq I_{lo}$. Given a program P and a set of experiments E , there is an information leak with respect to an equivalence relation R if there is a pair of computations induced by paths π and η that start from initial states with R -equivalent high components and equal low components in E , and lead to final states with different low components:

$$\begin{aligned} Leak_P(R, E, \pi, \eta) &\equiv \\ \exists s, t \in I \exists s', t' \in F : (s, s') \in \rho_\pi \wedge (t, t') \in \rho_\eta \wedge \\ s_{lo} = t_{lo} \wedge (s_{hi}, t_{hi}) \in R \wedge s_{lo} \in E \wedge s'_{lo} \neq t'_{lo}. \end{aligned}$$

The relation R over-approximates the maximal information that is leaked when the program P is run on the experiments E , written as $Confine_P(R, E)$, if there is no witness of further leaks:

$$\begin{aligned} Confine_P(R, E) &\equiv \\ \forall \pi, \eta \in \mathcal{T}^+ : \neg Leak_P(R, E, \pi, \eta). \end{aligned}$$

The largest equivalence relation R with $Confine_P(R, E)$ is the most precise characterization of the leaked information, denoted by \approx_E .

$$\approx_E \equiv \bigcup \{R \mid Confine_P(R, E)\}.$$

Example 1. If a program P satisfies $\text{Confine}_P(\text{All}_{hi}, I_{lo})$, a low observer does not learn any information about the high inputs even if he runs the program on all low inputs. This property is called *non-interference*.

The set of experiments E characterizes the set of low input values for which the information leakage of a program is characterized. Different instantiations of E correspond to different attack scenarios. In general, $\approx_E \subseteq \approx_{E'}$ whenever $E' \subseteq E$, i.e., more experiments allow for a more precise characterization of the secret and hence leak more information.

Example 2. Consider a password checker P that receives as high input a password and as low input a password candidate. The relation $\approx_{\{x\}}$ captures the information that P leaks when it is run on the password candidate x . The relation $\approx_{\{I_{lo}\}}$ captures the information that P leaks in an exhaustive password search.

$\text{Confine}_P(R, E)$ does not capture information leaks exposed due to non-termination of P . Sufficient preconditions for the termination of P can be automatically generated [14] and used to exclude non-terminating inputs. From now on, we will assume that P terminates on all initial states $s \in I$.

3.3. Quantitative information flow

In the following, we use information theory to give quantitative characterizations of equivalence relations R with $\text{Confine}_P(R, E)$. These characterizations have the advantage of being compact and human-readable. Moreover, they yield concise interpretations, e.g., in terms of the expected guessing effort that is required to determine the secret given the available information.

We first illustrate in detail how R can be characterized using the guessing entropy as a measure. After this, we give examples of alternative information measures.

Guessing entropy. Let A be a finite set and $p: A \rightarrow \mathbb{R}$ a probability distribution. For a random variable $X: A \rightarrow X$, we define $p_X: X \rightarrow \mathbb{R}$ as $p_X(x) = \sum_{a \in X^{-1}(x)} p(a)$, which is often denoted by $p(X = x)$ in the literature.

The guessing entropy of the random variable X is the average number of questions of the kind “does $X = x$ hold” that must be asked to guess X ’s value correctly [28]. If we assume p to be public, the optimal procedure is to try each of the possible values in order of their decreasing probabilities. Without loss of generality, let X be indexed such that $p_X(x_i) \geq p_X(x_j)$, whenever $i \leq j$. Then the *guessing entropy* $G(X)$ of X is defined as

$$G(X) = \sum_{1 \leq i \leq |X|} i p_X(x_i).$$

Given another random variable $\mathcal{Y}: A \rightarrow Y$, one denotes by $G(X|\mathcal{Y} = y)$ the guessing entropy of X given $\mathcal{Y} = y$, that

is, with respect to the distribution $p_{X|\mathcal{Y}=y}$. The *conditional guessing entropy* $G(X|\mathcal{Y})$ is defined as the expected value of $G(X|\mathcal{Y} = y)$ over all $y \in Y$, namely,

$$G(X|\mathcal{Y}) = \sum_{y \in Y} p_Y(y) G(X|\mathcal{Y} = y).$$

This quantity represents the expected number of guesses needed to determine X when the value of \mathcal{Y} is already known.

Guessing entropy and programs. We assume a given probability distribution $p: I_{hi} \rightarrow \mathbb{R}$ and an equivalence relation $R \subseteq I_{hi} \times I_{hi}$. We use two random variables to quantify the information that corresponds to R . The first is the random variable \mathcal{U} that models the choice of a secret in I_{hi} according to p (i.e., $\mathcal{U} = id_{I_{hi}}$). The second is the random variable \mathcal{V}_R that maps each secret to its R -equivalence class: $\mathcal{V}_R: I_{hi} \rightarrow I_{hi}/R$, where $\mathcal{V}_R(s_{hi}) = [s_{hi}]_R$.

Consider now a program P that satisfies $\text{Confine}_P(R, E)$. Then $G(\mathcal{U})$ is the expected number of guesses that an attacker must perform to determine the secret input, prior to observing the output of P . The value of $G(\mathcal{U}|\mathcal{V}_R = [s_{hi}]_R)$ is the adversary’s remaining guessing effort after learning the R -equivalence class of s_{hi} . Hence, $G(\mathcal{U}|\mathcal{V}_R)$ is a lower bound on the expected number of guesses that an attacker must perform for recovering the secret input after having run P on all experiments from E .

Alternative information measures. A number of alternative information measures, e.g., see [7], [24], [37], can be connected to programs along the same lines as the guessing entropy.

The *minimal guessing entropy* [24] $\hat{G}(\mathcal{U}|\mathcal{V}_R)$ is defined as the expected guessing effort for the weakest secrets in I_{hi} , i.e. the secrets that are easiest to guess after observing the output of P . Formally, one defines

$$\hat{G}(\mathcal{U}|\mathcal{V}_R) = \min\{G(\mathcal{U}|\mathcal{V}_R = [s_{hi}]_R) \mid s_{hi} \in I_{hi}\}.$$

The *conditional Shannon entropy* $H(\mathcal{U}|\mathcal{V}_R)$ captures the attacker’s uncertainty about the secret input of the program in terms of bits, i.e., in terms of a lower bound on the shortest representation of the secret [2], [36]. Formally, one defines $H(\mathcal{U}|\mathcal{V}_R)$ as the expected value of $H(\mathcal{U}|\mathcal{V}_R = [s_{hi}]_R)$ over all $s_{hi} \in I_{hi}$, where $H(\mathcal{U}|\mathcal{V}_R = [s_{hi}]_R)$ is the Shannon entropy of \mathcal{U} with respect to the distribution $p_{\mathcal{U}|\mathcal{V}_R=[s_{hi}]_R}$.

The *channel capacity*

$$C_R = \max_p (H(\mathcal{U}) - H(\mathcal{U}|\mathcal{V}_R))$$

is an upper bound on the rate at which information can be transmitted through the program by variation of its secret inputs [2], [36]. Here, p ranges over all probability distributions on I_{hi} .

The *conditional min-entropy* $H_\infty(\mathcal{U}|\mathcal{V}_R)$ captures the uncertainty about the secret input in terms of the probability

for guessing the secret in one try after observing the output of the program. There are different definitions for the conditional min-entropy in the literature; the one given in [37] is easily cast in terms of \mathcal{U} and \mathcal{V}_R .

Which measure is appropriate depends on the given attack scenario. For example, the channel capacity is appropriate for assessing the damage of intentional communication, e.g., by a Trojan horse, while the minimal guessing entropy can be used to assess the impact of unintentional information release.

4. Leak discovery and quantification

In this section, we present our method, called DISQUANT (Discovery and QUANTification), for the automatic discovery of leaked information and its comprehensive quantitative interpretation. DISQUANT takes as input a program P and a set of experiments E . It produces the characterization of the leaking information in terms of the equivalence relation \approx_E and performs its information-theoretic, quantitative interpretation.

DISQUANT consists of two procedures DISCO and QUANT that perform the qualitative and quantitative analysis, respectively. We proceed with a detailed description of each procedure below, before we provide a correctness statement for DISQUANT and discuss the scope of our method in Sections 4.3 and 4.4, respectively.

Our presentation does not rely on any particular way of representing programs and equivalence relations over program states. In Section 5, we exploit this generality to provide an implementation of DISQUANT using existing tools for program verification and information-theoretic analysis.

4.1. Discovery of information leaks

Given a program P and a set of experiments E , our goal is to synthesize \approx_E , i.e. the largest equivalence relation R such that $Confine_P(R, E)$ holds. The procedure Disco shown in Figure 1 computes this equivalence relation.

The computation is performed in an incremental fashion. Our procedure Disco stores the current candidate for the equivalence relation in the variable R . Initially, R contains the coarsest equivalence relation, i.e. one that claims that no information is leaked, see line 1 in Figure 1.

During the execution of Disco, it is checked whether R adequately represents the leaking information, see line 2. If R is inadequate, which is witnessed by a pair of paths, say π and η , then the candidate R is refined. The refinement step eliminates the discovered inadequacy using the relation $Refine_E(\pi, \eta)$, see line 3. The refinement step guarantees that the information leak witnessed by the pair of paths π and η is captured by the refined relation, i.e., after executing line 3 the predicate $Leak_P(R, E, \pi, \eta)$ no longer holds.

```

procedure Disco( $P, E$ )
input
   $P$  : program
   $E$  : set of experiments
vars
   $R$  : equivalence relation
output
   $\approx_E$  : characterization of leaking information
begin
1   $R := I_{hi} \times I_{hi}$ 
2  while exists  $\pi, \eta \in \mathcal{T}^+ : Leak_P(R, E, \pi, \eta)$  do
3     $R := R \cap Refine_E(\pi, \eta)$ 
4  done
5   $R := R \cup =_{I_{hi}}$ 
6  return  $R$ 
end.

```

Figure 1. Procedure Disco for computing a logical representation of the leaked information during a set of experiments.

The procedure DISCO generates a symmetric and transitive relation. For nondeterministic programs P , this relation is not necessarily reflexive, because there can be two computations that start from the same initial state and produce different low outputs. We choose a conservative approach and assume that such non-reflexive inputs are leaked. This is achieved by adding the identity relation in Line 5, which yields an equivalence relation.

The search for leaks in Line 2 and the refinement step in Line 3 are complex tasks for which we can employ existing techniques.

Detecting leaks. Line 2 identifies pairs of paths that witness information leaks with respect to the current candidate equivalence relation R . We discover such paths automatically by analyzing pairs of program runs, e.g., see [5], [39].

An equivalence relation R does not yet adequately capture the leaked information if and only if there is a pair of R -related initial high states from which this error state is reachable with low input from E . This reachability problem can be solved, for example, using software model checkers such as BLAST [21], SLAM [3], or SATABS [11]. The witness paths π and η can be reconstructed by inspecting the counterexample produced by the model checker.

Refining equivalence. The refinement of the candidate relation R is determined by the paths π and η . This step partitions R by adding new equivalence classes, see the relational intersection in line 3. To compute this refinement, we distinguish between all high input states that lead to different low-observable outputs along (π, η) . Formally, we use a relation $Refine_E(\pi, \eta)$ such that

$$\begin{aligned} \text{Refine}_E(\pi, \eta) \equiv \\ \{(s_{hi}, t_{hi}) \mid \forall s, t \in I \forall s', t' \in F : (s, s') \in \rho_\pi \wedge (t, t') \in \rho_\eta \wedge \\ s_{lo} = t_{lo} \wedge s_{lo} \in E \rightarrow s'_{lo} = t'_{lo}\} . \end{aligned}$$

The relation $\text{Refine}_E(\pi, \eta)$ can be obtained by applying existing quantifier elimination procedures, e.g., the Fourier-Motzkin algorithm for linear arithmetic.

Correctness of Disco. The procedure Disco in Figure 1 is a formalization of our counterexample-guided computation of leaking information, based on the building blocks presented above. The correctness of Disco is formalized in the following proposition.

Proposition 1. *Let P be a program and let E be a set of experiments. If Disco terminates on input (P, E) and returns R , then*

$$R = \approx_E ,$$

that is, R is the largest equivalence relation such that $\text{Confine}_p(R, E)$ holds.

Proof: $\text{Confine}_p(R, E)$ follows directly from the termination condition of Disco. R is the largest relation with this property, as it is the conjunction of weakest preconditions for the equality of the low outputs. It remains to show that R is an equivalence relation, i.e., reflexive, symmetric and transitive. For symmetry, assume that there is a $(s, t) \in R$ such that $(t, s) \notin R$. Then there is a conjunct $\text{Refine}_E(\pi, \eta)$ in R that is not satisfied by (t, s) , i.e., there is an experiment in which t and s lead to different low outputs along (π, η) . Then s and t also lead to different low outputs along (η, π) . As $\text{Confine}_p(R, E)$ is satisfied, this contradicts the assumption $(s, t) \in R$. For transitivity, assume that there are $(s, t), (t, u) \in R$ such that $(s, u) \notin R$. Then there exists a pair of paths (π, η) and $s', u' \in F$ such that $(s, s') \in \rho_\pi$, $(u, u') \in \rho_\eta$ and $s'_{lo} \neq u'_{lo}$. Choose $t' \in F$ with $(t, t') \in \rho_\eta$. Such a t' exists because we assume that P terminates on all $t \in I$. As $(s, t), (t, u) \in R$, we have $s'_{lo} = t'_{lo}$ and $t'_{lo} = u'_{lo}$, which contradicts the assumption $s'_{lo} \neq u'_{lo}$. Reflexivity holds because the identity relation is added to R before it is returned. \square

4.2. Quantification of information leaks

For computing the information-theoretic characteristics presented in Section 3.2, the procedure QUANT determines the number r and the sizes n_1, \dots, n_r of the equivalence classes of an equivalence relation R . See Figure 2 for its description. QUANT proceeds by iteratively computing representative elements of the equivalence classes of the relation R , identifying the corresponding equivalence classes and determining their sizes.

```

procedure QUANT( $R$ )

```

```

input

```

```

 $R$  : equivalence relation

```

```

vars

```

```

 $Q$  : auxiliary set of high initial states

```

```

output

```

```

 $\{n_1, \dots, n_r\}$  : sizes of the  $R$ -equivalence classes

```

```

begin

```

```

1    $i := 1$ 

```

```

2    $Q := I_{hi}$ 

```

```

3   while  $Q \neq \emptyset$  do

```

```

4      $s_i := \text{select in } Q$ 

```

```

5      $n_i := \text{Count}([s_i]_R)$ 

```

```

6      $Q := Q \setminus [s_i]_R$ 

```

```

7      $i := i + 1$ 

```

```

8   done

```

```

9   return  $\{n_1, \dots, n_{i-1}\}$ 

```

```

end.

```

Figure 2. Procedure QUANT for computing the information-theoretic characteristics of a given equivalence relation R .

Our iteration manipulates a set of high initial states Q , which is initialized to the full set in line 2. In the first iteration, we choose an arbitrary $s_1 \in Q$ and determine the size of the R -equivalence class $[s_1]_R$ of s_1 . In the i -th iteration, we find s_i such that $[s_i]_R \neq [s_j]_R$ for all $j < i$, see line 4. If such an s_i exists, we determine the size of $[s_i]_R$ in line 5 and proceed with the next iteration after excluding the equivalence class of s_i from Q . Otherwise, we report the sizes of the equivalence classes.

Logical operations. Our procedure performs a number of logical operations on the set Q . We assume that the set Q is given as a logical assertion over the high variables h and that the equivalence relation R is given by an assertion over h and their copies \bar{h} . The while condition in line 3 is performed by a logical satisfiability check. Line 4 requires finding a satisfying assignment to the assertion $Q(h)$. We determine the members of the equivalence class $[s_i]_R$ by the assertion $R(h, \bar{h}) \wedge s_i = \bar{h}$ whose free variable h represents the high states related to s_i . Line 6 amounts to logical conjunction and negation.

Counting elements. Our method requires an algorithm *Count* that, given a set A , returns the number of elements in A , i.e., $\text{Count}(A) = |A|$. If A is represented as a formula ϕ , this number corresponds to the number of models for ϕ .

For example, if S is represented in linear arithmetic, this task can be solved as follows. Suppose ϕ is in disjunctive normal form, i.e., $\phi = \phi_1 \vee \phi_2 \vee \dots \vee \phi_n$, where the clauses ϕ_i are conjunctions of linear inequalities. Then the number of satisfying assignments of each clause corresponds to the number of integer solutions of the corresponding system

of inequalities. We can apply Barvinok’s algorithm [6] for computing the number of integer points in convex polyhedra. The Lattice Point Enumeration Tool (LATE) [25] provides an implementation of this algorithm.

By summing over the number of solutions for every clause and removing possible duplicates, we obtain the number of models for ϕ .

Correctness of QUANT. The algorithm QUANT in Figure 2 is a formalization of the procedure sketched above. Its correctness is implied by the following proposition.

Proposition 2. *Let R be an equivalence relation on I_{hi} with $I_{hi}/R = \{B_1, \dots, B_r\}$. If QUANT terminates on input R , it returns the set $\{|B_1|, \dots, |B_r|\}$.*

Proof: Q is a predicate representing a set of high initial states. It is initialized with I_{hi} , which represents all possible high initial states. The assignment $Q := Q \setminus [s_i]_R$ removes $[s_i]_R$ from this set. As the next representative element is chosen from this reduced state space, we have $[s_i]_R \neq [s_j]_R$ for $i \neq j$. The algorithm terminates if $Q = \emptyset$, which implies that all equivalence classes have been found. If *Count* is correct, the assertion follows. \square

Information-theoretic interpretation. With a uniform probability distribution on I_{hi} , the output of QUANT can be given various information-theoretic interpretations. We consider the attacker’s guessing effort for deducing the secret input to the program from the observable output (conditional and minimal guessing entropy), the attacker’s remaining uncertainty about the secret in terms of bits (Shannon entropy), and the maximal rate at which information can be transmitted using the program as a communication channel (channel capacity). A formula for the min-entropy in terms of $|I_{hi}|$ and r can be found in [37].

Proposition 3. *Let $R \subseteq I_{hi} \times I_{hi}$ be an equivalence relation with $I_{hi}/R = \{B_1, \dots, B_r\}$, let $n = |I_{hi}|$, and let \mathcal{U} and \mathcal{V}_R be defined as in Section 3.3. Then*

- 1) $G(\mathcal{U}|\mathcal{V}_R) = \frac{1}{2n} \sum_{i=1}^r |B_i|^2 + \frac{1}{2}$,
- 2) $\hat{G}(\mathcal{U}|\mathcal{V}_R) = \min\{(|B_i| + 1)/2 \mid i \in \{1, \dots, r\}\}$,
- 3) $H(\mathcal{U}|\mathcal{V}_R) = \frac{1}{n} \sum_{i=1}^r |B_i| \log_2 |B_i|$,
- 4) $C_R = \log_2 r$, if P is deterministic.

Proof: Assertions 1 and 3 are due to [24]. For 2, a simple calculation shows that, on the average, $(k+1)/2$ attempts are needed for guessing one of k equally likely alternatives. For 4, observe that $H(\mathcal{U}) - H(\mathcal{U}|\mathcal{V}_R) = H(\mathcal{V}_R) - H(\mathcal{V}_R|\mathcal{U}) = H(\mathcal{V}_R)$. The last step follows because \mathcal{V}_R is determined by \mathcal{U} . $H(\mathcal{V}_R)$ reaches its maximum of $\log r$ when \mathcal{V}_R is uniformly distributed. For deterministic programs P , this uniform distribution can be achieved by a suitable choice of $p_{\mathcal{U}}$. \square

4.3. Correctness of DISQUANT

The following theorem states the correctness of DISQUANT.

Theorem 1 (Correctness of DISQUANT). *Let P be a program and E be a set of experiments. If DISQUANT(P, E) terminates, then it outputs the sizes $\{n_1, \dots, n_r\}$ of the \approx_E -equivalence classes of I_{hi} .*

Theorem 1 implies that DISQUANT correctly determines the sizes of the \approx_E -equivalence classes. Together with Proposition 3, this gives rise to push-button technology for computing a variety of information-theoretic measures beyond those offered by existing approaches.

4.4. Scope of our method

We briefly discuss the scope of our method. In particular, we identify the languages for which DISCO can be easily implemented, discuss the impact of nontermination, and present initial ideas for scaling-up.

Soundness and language features. In principle, DISCO can be implemented for any programming language for which model-checkers are available, including those with arrays [22] and heap structures [8]. When implementing DISCO using a model-checker that is sound but not complete, spurious leaks may be detected. In this case, the equivalence relation computed by DISCO will be finer than \approx_E , which corresponds to an over-approximation of the maximal information leakage, and can be used for certifying the security of a program.

Similarly, QUANT can in principle be implemented for any logical theory for which the number of models of an assertion can be counted, however, we are not aware of practical solutions for this task that go beyond Boolean propositional logic and Presburger Arithmetic.

In Section 5 we present an implementation of DISQUANT for a subset of C with expressions in linear arithmetic.

(Non)termination of DISQUANT. The algorithm DISCO is initialized with the coarsest equivalence relation on the set of secret inputs, which claims absence of leaks in the analyzed program. This equivalence relation is refined during the execution of DISCO. A longer execution time of DISCO corresponds to a finer equivalence relation, which corresponds to more information leakage. DISCO can be interrupted at any time, and will return an equivalence relation that is coarser than \approx_E , i.e., an under-approximation of the maximal information that the analyzed program can leak. If this under-approximation already violates a given security policy, the *insecurity* of the analyzed program is certified.

For QUANT, a longer execution time corresponds to a larger number of equivalence classes, which corresponds to more

information leakage. QUANT can be interrupted at any time. In the worst-case, all equivalence classes that have not been determined at this point are singleton sets. Together with the sizes of the equivalence classes that have already been enumerated and the size of the set of secret inputs, this can be used for computing an over-approximation of the maximal information that is leaked. In this way, the output of an interrupted execution of QUANT can be used for certifying the security of the analyzed program.

Scaling up. The computation of DISQUANT relies on the enumeration of the leaks of a program. The number of leaks can be quadratic in the number of program paths. Hence, when applied in a vanilla fashion, our method works well on programs with few paths, e.g. programs without branching statements inside of loops. For programs with branching statements inside of loops, the number of leaks may be large. For example, in the electronic auction program given in Section 2, the number of paths is exponential in the number of bids n , which does not scale.

A number of speed-up techniques, such as predicate abstraction [19] and widening [15], can be used to analyze multiple program paths in Line 2 of Disco. Applying such techniques will improve the efficiency of our method, however, these speed-up techniques may lead to incompleteness, i.e., their application may report spurious leaks. In this case, the equivalence relation computed by Disco will be finer than \approx_E , which corresponds to an over-approximation of the maximal information leakage.

Our method applies self-composition, which doubles the program size. It is possible to avoid duplication of some parts of the program by making related statements share the same control-flow constructions, as proposed in [38].

5. Implementing DISQUANT

In this section, we outline the implementation of a push-button tool for the quantitative information flow analysis of C programs, based on the algorithm DISQUANT presented in this paper. We show how all of the building blocks on which DISQUANT is based can be instantiated and combined. In Section 6, we report on experimental results obtained by using our tool to analyze a number of example programs.

5.1. Language

We use a subset of C as the programming language. In particular, we allow programs to contain all control flow statements and assignments to scalar variables. As the expression language, we use linear arithmetic, i.e. integer variables with addition and comparison operators. We use propositional logic with propositions in linear arithmetic to represent the equivalence relations R and the set E .

As a convention, we assume that the high and low input states of a program (i.e. I_{hi} and I_{lo}) are stored in variables named h_1, \dots, h_n and l_1, \dots, l_m , respectively. We sometimes use single input variables h and l to comprise all of the high and low input to a program, respectively, and we write $P(h, l)$ to emphasize that variables h and l occur in P .

5.2. Implementing Disco

The building blocks for Disco are methods for checking *Confine*, for computing *Refine*, and for detecting leaks. We show how all three can be implemented using the software model checker ARMC and the Omega calculator, which is a tool for manipulating formulas in linear arithmetic with quantifiers.

Implementing *Confine*. We cast the check for $Confine_P(R, E)$ as a reachability problem, which we solve using the model checker ARMC. As a preprocessing step, we create a modified copy \bar{P} of P , where we replace every program variable x that occurs in P by a fresh variable \bar{x} . This ensures that P and \bar{P} have disjoint variable sets. The existence of an R -leak corresponds to the reachability of the **error** state in the following program.

```

if ( $l = \bar{l} \wedge l \in E \wedge (h, \bar{h}) \in R$ )
   $P(h, l)$ 
   $\bar{P}(\bar{h}, \bar{l})$ 
if  $l \neq \bar{l}$ 
  error
return

```

We apply ARMC to this reachability problem. If **error** is reachable, ARMC outputs a path to **error**. As the variable sets of both copies of P are disjoint, the leak (π, η) can be reconstructed from this path. This directly leads to an implementation of $Leak_P(E, R)$. As we will explain below, this direct computation of $Leak_P(E, R)$ can be avoided due to extra information provided by ARMC.

Implementing *Refine*. ARMC not only returns the path to **error**, but also a formula in linear arithmetic that characterizes all initial states from which the error state is reachable along the counterexample path. This formula characterizes sets of pairs $((h, l), (\bar{h}, \bar{l}))$ of initial states and can thus be interpreted as a binary relation \bar{R} over I . We project out the low variables from \bar{R} , i.e. we define $\bar{R}_E \subseteq I_{hi} \times I_{hi}$ as

$$\bar{R}_E \equiv \{(h, \bar{h}) \mid \exists l \in E : ((h, l), (\bar{h}, \bar{l})) \in \bar{R}\} .$$

\bar{R}_E characterizes all pairs of high inputs from which the error state can be reached with an experiment from E . The complement of \bar{R}_E characterizes all pairs of high initial states from which the error state is not reachable along (π, η) with an experiment from E , i.e., it corresponds to $Refine_E(\pi, \eta)$:

$$Refine_E(\pi, \eta) \equiv I_{hi} \times I_{hi} \setminus \bar{R}_E .$$

In our implementation, E is represented as an assertion in linear arithmetic, and we perform the projection and set difference using the Omega calculator.

5.3. Implementing QUANT

The building blocks for the algorithm QUANT presented in Section 4.1 are operations on sets and relations, such as finding models, determining equivalence classes, and counting the number of elements (the function *Count*). First, we show how to implement the relational operations using the Omega calculator. Subsequently, we show how to count the number of elements in a set using the Lattice Point Enumeration Tool (LATTÉ).

Implementing the relational operations. The example command of the Omega calculator implements the operation of picking an element from Q , see line 4 in Figure 2. The computation of an equivalence class in line 5 can be implemented using relational composition:

$$[s]_R \equiv \{s\} \circ R = \{t \in I_{hi} \mid (s, t) \in R\} .$$

The loop body in Figure 2 (except for the call to *Count* in line 5) thus maps to the following input of the Omega calculator:

```
B:= S.R;
Q:= Q-B;
S:= example Q;
```

Here R and Q represent R and Q , respectively. S corresponds to $\{s_i\}$, B corresponds to $[s_i]_R$, and “.” denotes the relational composition. The result of successively applying these operations to $Q = I_{hi}$ and the relation R given by Disco is the set

$$I_{hi}/R = \{B_1, \dots, B_n\}$$

of R -equivalence classes, each represented as a linear arithmetic proposition in disjunctive normal form.

Implementing Count. An R -equivalence class B is represented as a conjunction $\phi \equiv c_1 \wedge \dots \wedge c_m$ of atomic propositions $c_i \equiv \sum_{j=1}^n a_{ij}h_j \leq b_i$, with $i \in \{1, \dots, m\}$ and can be interpreted as a system of linear inequalities

$$A h \leq b ,$$

where $A = (a_{ij})$, $h = (h_1, \dots, h_n)$, and $b = (b_1, \dots, b_m)$. It is not difficult to see that the number of satisfying assignments of ϕ (i.e., the size of B) corresponds to the number of integer solutions to this system of inequalities. For counting the number of solutions of two disjuncts $\phi = \phi_1 \vee \phi_2$, we compute

$$\text{Count}(\phi) \equiv \text{Count}(\phi_1) + \text{Count}(\phi_2) - \text{Count}(\phi_1 \wedge \phi_2) .$$

This easily generalizes to the DNF formulas output by the Omega calculator.

For efficiency reasons, ARMC assumes that program variables range over rational numbers. As a consequence, the equivalence classes computed by the Omega calculator may be of unbounded size. In practice, however, the values of integers variables will be bounded by the range of their respective types. To incorporate such bounds, we extend A by additional constraints of the form

$$I_n h \leq b_u \text{ and } -I_n h \leq b_l .$$

Here, I_n denotes the identity matrix of dimension n and b_u and b_l are vectors of upper and lower bounds, respectively. We note that such a bounding step does not compromise the soundness of our approach, however it might sacrifice completeness, e.g., if the input leading to a leak lies outside of the bounds.

Applying LATTÉ to the following system of inequalities

$$\begin{pmatrix} A \\ I_n \\ -I_n \end{pmatrix} h \leq \begin{pmatrix} b \\ b_u \\ b_l \end{pmatrix}$$

yields the number of elements of the equivalence class B represented by ϕ . For all of our examples, the running time of LATTÉ was largely independent of the bounds b_u and b_l .

6. Experimental results

In this section, we report on experimental results obtained by applying our technique to determine the information that is revealed by a program. As examples, we analyze programs for checking passwords, debiting from an electronic purse, and computing sum queries.

6.1. Password checker

Consider a password checker that receives a secret password h and a candidate password l , and outputs whether the candidate password was correct, i.e. whether $l = h$.

```
if (l==h)
  l=1;
else
  l=0;
```

A password check necessarily reveals partial information about the password. We use our approach to characterize this information for two different experiments. As shown in Section 5.2, the set of experiments E determines how the low variables are eliminated from the equivalence relation computed by ARMC. In Section 5.2, this elimination is performed in each refinement step. For a uniform presentation of both experiments, we postpone this elimination until after the computation of the complete relation.

Then the relation R computed by ARMC is

$$\begin{aligned} R \equiv & (\bar{h} = l \wedge l - h \leq -1) \vee (\bar{h} = l \wedge l - h \geq 1) \\ & \vee (h = l \wedge \bar{h} - l \leq -1) \vee (h = l \wedge l - \bar{h} \leq -1) \end{aligned}$$

We consider two experiments, where the first corresponds to a single password guess and the second corresponds to an exhaustive search of the password space.

Single password guess. The experiment $E = \{x\}$ corresponds to the guess of a single password x . The relation $\approx_{\{x\}}$ captures the information that is leaked in this guess and is obtained from R as described in Section 5.2. For $x = 0$, the equivalence classes computed by QUANT are

$$\begin{aligned} B_1 &\equiv h = 0 \\ B_2 &\equiv h \leq -1 \vee h \geq 1, \end{aligned}$$

which reflects that a correct guess reveals the password while all incorrect (nonzero, in this case) passwords cannot be distinguished. We obtain $|B_1| = 1$ and $|B_2| = 4294967295$ if the passwords are nonnegative 32-bit integers. For uniformly distributed passwords, the attacker’s uncertainty about the password hence drops from 32 bits to

$$\frac{1}{2^{32}} \sum_{i=1}^2 |B_i| \log |B_i| = 31.999999992$$

after a single guess.

Exhaustive search. The experiment $E = I_{lo}$ corresponds to exhaustively running P on all password candidates. A representation \approx_E of the leaked information is obtained from ARMC’s output as described in Section 5.2. We obtain

$$\approx_{I_{lo}} \equiv h = \bar{h},$$

which is the identity relation on I_{hi} .

This result confirms the fact that the attacker can determine every password by exhaustive search. His remaining uncertainty about the password will then be 0. Note that we derived this interpretation from the output of Disco (i.e., without applying QUANT). QUANT enumerates all $\approx_{I_{lo}}$ -equivalence classes, which is infeasible for large password spaces. The overall running time of the analysis was less than 2 seconds.

6.2. Electronic purse

Consider a program that receives as input the balance h of a bank account and debits a fixed amount l from this account until the balance is insufficient for this transaction, i.e. until $h < l$.

```
lo=0;
while(h>=l){
  h=h-l;
  lo=lo+1;
}
```

Upon termination, the program outputs the number of times l has been successfully subtracted from h in the variable lo .

This number reveals partial information about the initial balance of the account. We use our approach to automatically quantify this information.

The number of loop iterations depends on the account balance h . Without any restriction on the range of the input values, the number of program paths (and leaks) is infinite, and Disco will not terminate. To avoid this, we bound the maximal account balance by 20 (i.e. $h < 20$). We consider a single experiment where $l = 5$. With these parameters, Disco computes the following equivalence relation on $\{0, \dots, 19\}$

$$\begin{aligned} \approx_{\{5\}} &\equiv 10 \leq h \wedge h \leq 14 \wedge 10 \leq \bar{h} \wedge \bar{h} \leq 14 \\ &\vee 5 \leq h \wedge h \leq 9 \wedge 5 \leq \bar{h} \wedge \bar{h} \leq 9 \\ &\vee 0 \leq h \wedge h \leq 4 \wedge 0 \leq \bar{h} \wedge \bar{h} \leq 4 \\ &\vee 15 \leq h \wedge h \leq 19 \wedge 15 \leq \bar{h} \wedge \bar{h} \leq 19, \end{aligned}$$

Given $\approx_{\{5\}}$, QUANT computes the equivalence classes

$$\begin{aligned} B_1 &\equiv 0 \leq h \leq 4 \\ B_2 &\equiv 5 \leq h \leq 9 \\ B_3 &\equiv 10 \leq h \leq 14 \\ B_4 &\equiv 15 \leq h \leq 19, \end{aligned}$$

from which we obtain $|B_1| = |B_2| = |B_3| = |B_4| = 5$. This result confirms the intuition that our program leaks the result of integer division of h by l .

For this example, the structure of the equivalence classes is simple and can be directly interpreted. We also give an information-theoretic interpretation in terms of guessing. If the value of h is chosen from a uniform distribution (modeled by a random variable \mathcal{U}), the number of guesses to correctly determine the purse balance is $G(\mathcal{U}) = 10.5$. Using Proposition 3, the expected number of guesses decreases to

$$G(\mathcal{U} | \mathcal{V}_{\approx_{\{5\}}}) = \frac{1}{2 \cdot 20} \sum_{i=1}^4 |B_i|^2 + \frac{1}{2} = 3$$

by observing the low output of the program.

The overall running time for analyzing this example is dominated by the model checker’s running time of 24 seconds. The running times for computing the equivalence classes and determining their size are each below one second.

6.3. Sum query

Consider a program that receives as input n secret integers and computes and outputs their sum. We use our approach to characterize the information that is revealed by this program. This result corresponds to the information that a sum query reveals about a database record. For our example, we choose $n = 3$ and represent the input by variables h_1, h_2, h_3 , i.e., we analyze the program

```
l=h1;
```

$$l=1+h_2;$$

$$l=1+h_3;$$

The equivalence relation synthesized by Disco is

$$R \equiv \bar{h}_3 = h_1 + h_2 + h_3 - \bar{h}_1 - \bar{h}_2 .$$

For determining the sizes and the number of the \approx -equivalence classes, we choose $0 \leq h_i < 10$ for $i \in \{1, 2, 3\}$. QUANT computes equivalence classes of the form

$$B_i \equiv h_1 + h_2 + h_3 = i - 1$$

for $i \in \{1, \dots, 28\}$ with respective sizes $|B_1|, \dots, |B_{28}|$ of 1, 3, 6, 10, 15, 21, 28, 36, 45, 55, 63, 69, 73, 75, 75, 73, 69, 63, 55, 45, 36, 28, 21, 15, 10, 6, 3, 1.

For independently chosen and uniformly distributed input values (modeled by a random variable \mathcal{U}) the expected number of guesses to correctly determine the input is

$$G(\mathcal{U}) = 500.5 .$$

The average number of guesses is reduced to

$$G(\mathcal{U}|\mathcal{V}_{\approx}) = \frac{1}{2 \cdot 10^3} \sum_{i=1}^{28} |B_i|^2 + \frac{1}{2} = 28.126$$

by observing the output of the analyzed program.

An analysis with the minimal guessing entropy shows

$$\hat{G}(\mathcal{U}|\mathcal{V}_{\approx}) = 1 ,$$

which additionally reveals that there are secrets that are very easy to guess, a fact that is not revealed by any average-case measure. This illustrates the benefit of combining multiple information measures in one analysis.

7. Conclusion

We presented the first automatic method for information-flow analysis that discovers what information is leaked and computes its comprehensive quantitative interpretation.

References

- [1] T. Amtoft, S. Bandhakavi, and A. Banerjee. A logic for information flow in object-oriented programs. In *Proc. Symposium on Principles of Programming Languages (POPL '06)*, pages 91–102. ACM Press, 2006.
- [2] R. B. Ash. *Information Theory*. Dover Publications Inc., 1990.
- [3] T. Ball, R. Majumdar, T. Millstein, and S. Rajamani. Automatic predicate abstraction of C programs. In *Proc. ACM Conference On Programming Language Design and Implementation (PLDI '01)*, volume 36 of *ACM SIGPLAN Notices*, 2001.
- [4] A. Banerjee, D. A. Naumann, and S. Rosenberg. Expressive declassification policies and modular static enforcement. In *Proc. IEEE Symposium on Security and Privacy (S&P '08)*, pages 339–353. IEEE Computer Society, 2008.
- [5] G. Barthe, P. D’Argenio, and T. Rezk. Secure Information Flow by Self-Composition. In *Proc. IEEE Computer Security Foundations Workshop (CSFW '04)*, pages 100–114. IEEE Computer Society, 2004.
- [6] A. Barvinok. A Polynomial Time Algorithm for Counting Integral Points in Polyhedra when the Dimension is Fixed. *Mathematics of Operations Research*, 19:189–202, 1994.
- [7] C. Cachin. *Entropy Measures and Unconditional Security in Cryptography*. PhD thesis, ETH Zürich, 1997.
- [8] C. Calcagno, D. Distefano, P. W. O’Hearn, and H. Yang. Compositional shape analysis by means of bi-abduction. In *Proc. Symposium on Principles of Programming Languages (POPL '09)*, pages 289–300. ACM Press, 2009.
- [9] D. Clark, S. Hunt, and P. Malacaria. Quantitative Information Flow, Relations and Polymorphic Types. *J. Log. Comput.*, 18(2):181–199, 2005.
- [10] D. Clark, S. Hunt, and P. Malacaria. A static analysis for quantifying information flow in a simple imperative language. *Journal of Computer Security*, 15(3):321–371, 2007.
- [11] E. M. Clarke, D. Kroening, N. Sharygina, and K. Yorav. Satabs: SAT-based predicate abstraction for ANSI-C. In *Proc. Intl. Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS' 05)*, volume 3440 of *LNCS*, pages 570–574. Springer, 2005.
- [12] M. R. Clarkson, A. C. Myers, and F. B. Schneider. Belief in Information Flow. In *Proc. IEEE Computer Security Foundations Workshop (CSFW '05)*, pages 31–45. IEEE Computer Society, 2005.
- [13] E. Cohen. Information Transmission in Sequential Programs. In *Foundations of Secure Computation*, pages 297–335. Academic Press, 1978.
- [14] B. Cook, A. Podelski, and A. Rybalchenko. Termination proofs for systems code. In *Proc. ACM Conference on Programming Language Design and Implementation (PLDI '06)*, pages 415–426. ACM Press, 2006.
- [15] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. Symposium on Principles of Programming Languages (POPL '77)*, pages 238–252, 1977.
- [16] A. Darvas, R. Hähnle, and D. Sands. A Theorem Proving Approach to Analysis of Secure Information Flow. In *Proc. International Conference on Security in Pervasive Computing*, LNCS 3450, pages 193 – 209. Springer, 2005.
- [17] D. E. Denning. *Cryptography and Data Security*. Addison-Wesley, 1982.
- [18] R. Giacobazzi and I. Mastroeni. Abstract Non-Interference: Parameterizing Non-Interference by Abstract Interpretation. In *Proc. ACM Symposium on Principles of Programming Languages (POPL '04)*, pages 186–197. ACM, 2004.

- [19] S. Graf and H. Saïdi. Construction of Abstract State Graphs with PVS. In *Proc. Intl. Conference on Computer Aided Verification (CAV '97)*, volume 1254 of *LNCS*, pages 72–83. Springer, 1997.
- [20] J. W. Gray. Toward a Mathematical Foundation for Information Flow Security. *Journal of Computer Security*, 1(3-4):255–294, 1992.
- [21] T. A. Henzinger, R. Jhala, R. Majumdar, and K. L. McMillan. Abstractions from proofs. In *Proc. ACM Symposium on Principles of Programming Languages (POPL '04)*, pages 232–244. ACM Press, 2004.
- [22] R. Jhala and K. L. McMillan. Array abstractions from proofs. In *Proc. Intl. Conference on Computer Aided Verification (CAV '07)*, volume 4590 of *LNCS*, pages 193–206. Springer, 2007.
- [23] B. Köpf and D. Basin. Timing-Sensitive Information Flow Analysis for Synchronous Systems. In *Proc. European Symposium on Research in Computer Security (ESORICS '06)*, *LNCS* 4189, pages 243–262. Springer, 2006.
- [24] B. Köpf and D. Basin. An Information-Theoretic Model for Adaptive Side-Channel Attacks. In *Proc. ACM Conference on Computer and Communications Security (CCS '07)*, pages 286–296. ACM, 2007.
- [25] J. A. D. Loera, D. Haws, R. Hemmecke, P. Huggins, J. Tauzer, and R. Yoshida. LattE. <http://www.math.ucdavis.edu/latte/>. [Online; accessed 08-Nov-2008].
- [26] G. Lowe. Quantifying Information Flow. In *Proc. IEEE Computer Security Foundations Workshop (CSFW '02)*, pages 18–31. IEEE Computer Society, 2002.
- [27] P. Malacaria. Assessing security threats of looping constructs. In *Proc. Symposium on Principles of Programming Languages (POPL '07)*, pages 225–235. ACM Press, 2007.
- [28] J. L. Massey. Guessing and Entropy. In *Proc. IEEE International Symposium on Information Theory (ISIT '94)*, page 204. IEEE Computer Society, 1994.
- [29] S. McCamant and M. D. Ernst. Quantitative information flow as network flow capacity. In *Proc. Conf. on Programming Language Design and Implementation (PLDI '08)*, pages 193–205, 2008.
- [30] J. K. Millen. Covert Channel Capacity. In *Proc. IEEE Symposium on Security and Privacy (S&P '87)*, pages 60–66. IEEE Computer Society, 1987.
- [31] A. Podelski and A. Rybalchenko. ARMC: the logical choice for software model checking with abstraction refinement. In *Proc. Intl. Symposium on Practical Aspects of Declarative Languages (PADL '07)*. Springer, 2007.
- [32] E. Rosser, W. Kelly, W. Pugh, D. Wonnacott, T. Shpeisman, and V. Maslov. The Omega Project. <http://www.cs.umd.edu/projects/omega/>. [Online; accessed 05-Nov-2008].
- [33] A. Sabelfeld and A. C. Myers. Language-based Information-Flow Security. *IEEE J. Selected Areas in Communication*, 21(1):5–19, 2003.
- [34] A. Sabelfeld and A. C. Myers. A model for delimited information release. In *Proc. Intl. Symposium on Software Security (ISSS '03)*, *LNCS* 3233, pages 174–191. Springer, 2004.
- [35] A. Sabelfeld and D. Sands. Dimensions and Principles of Declassification. In *Proc. IEEE Workshop on Computer Security Foundations (CSFW '05)*, pages 255–269. IEEE Computer Society, 2005.
- [36] C. E. Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27:379–423 and 623–656, July and October 1948.
- [37] G. Smith. On the foundations of quantitative information flow. In *Proc. Intl. Conference of Foundations of Software Science and Computation Structures (FoSSaCS '09)*, *LNCS* 5504, pages 288–302. Springer, 2009.
- [38] T. Terauchi and A. Aiken. Secure information flow as a safety problem. In *Proc. Intl. Symposium on Static Analysis (SAS '05)*, *LNCS* 3672, pages 352–367. Springer, 2005.
- [39] H. Unno, N. Kobayashi, and A. Yonezawa. Combining Type-Based Analysis and Model Checking for Finding Counterexamples Against Non-interference. In *Proc. Workshop on Programming Languages and Analysis for Security (PLAS '06)*, pages 17–26. ACM Press, 2006.
- [40] J. T. Wittbold and D. M. Johnson. Information Flow in Nondeterministic Systems. In *Proc. IEEE Symposium on Security and Privacy (S&P '90)*, pages 144–161. IEEE Computer Society, 1990.
- [41] S. Zdancewic and A. C. Myers. Robust declassification. In *Proc. IEEE Computer Security Foundations Workshop (CSFW '01)*, pages 15–23. IEEE Computer Society, 2001.