Information Leakage Analysis of Database Query Languages

Raju Halder
Indian Institute of Technology
Patna, India
halder@iitp.ac.in

Matteo Zanioli
Università Ca' Foscari
Venezia, Italy
zanioli@unive.it

Agostino Cortesi
Università Ca' Foscari
Venezia, Italy
cortesi@unive.it

ABSTRACT

In this work, we extend language-based information-flow security analysis to the case of database applications embedding query languages. The analysis is performed by (*i*) computing an overapproximation of variables' dependences, in the form of propositional formula, occurred up to each program point, (*ii*) checking the satisfiability on assigning truth values to variables, (*iii*) analyzing the application over a *numerical abstract* domain, and finally, (*iv*) enhancing the analysis using the reduced product of the *propositional formulae domain* and the *numerical abstract domain*.

Categories and Subject Descriptors

F.3.2 [**Semantics of Programming Languages**]: Program Analysis; H.2.0 [**General**]: Security, integrity, and protection; H.2.3 [**Languages**]: Data manipulation languages (DML), Query languages

General Terms

Static Analysis, Abstract Interpretation, Databases, Security

Keywords

Information Flow Analysis, Query Languages

1. INTRODUCTION

Various language-based information flow security models have been proposed, aiming at preventing unauthorized leakage of sensitive data, directly or indirectly, while propagating through an application [7, 8, 12, 14, 17]. Works in this direction have been starting with the pioneering work of Dennings in the 1970s [3].

To ensure end-to-end security, the notion of non-interference was introduced [14]: Given a program P and set of states Σ, the non-interference policy states that $\forall \sigma_1, \sigma_2 \in \Sigma.\ \sigma_1 \equiv_L \sigma_2 \implies [\![P]\!]\sigma_1 \equiv_L [\![P]\!]\sigma_2$, where $[\![.]\!]$ is semantic function and $\equiv_L$ represents low-equivalence relation between states; That is, a variation of confidential data does not cause any variation to public data.

Existing static-analysis models in the literature for the verification of such a property can be classified as type system-based [14, 16], dependence graph-based [6, 7, 9], slicing-based [1, 8], etc. Observably, all these notable works refer only to imperative, object-oriented, functional programming languages [7, 8, 12, 13, 14], while in the information system scenarios most of the data-intensive applications are embedded with SQL commands extracting or manipulating data from back-end databases. Various access control mechanisms are although proved to be very efficient at database level, but in practice confidentiality of sensitive database information can possibly be compromised while propagating through the applications accessing and processing them legitimately. No attention has been given in this direction to address such kind of leakage of database information through data-intensive applications.

In [17, 18], authors used logical formulae to represent variables' dependences in the form

$$\bigwedge_{0\leq i\leq n,\ 0\leq j\leq m} \{y_i \rightarrow z_j\}$$

which means that the values of variable z_j possibly depend on the values of variable y_i. The information leakage analysis on this domain of propositional formulae involves the following steps:

- Construction of propositional formula ψ representing an over-approximation of variables' dependences at each program point.
- Assignment of truth values to each variable considering its sensitivity by a truth-assignment function ξ. If ξ does not satisfy ψ, then there could be some information leakage.
- Analysis of program over a *numerical abstract domain* using the reduced product of the *propositional formulae domain* and the *numerical abstract domain* to make the analysis more accurate by removing possible false positives.

In this paper, we aim to extend the full power of the proposed model in [17, 18] to the case of data-intensive applications embedding SQL statements, in order to identify possible leakage of sensitive database information as well. In particular,

- We define an abstract semantics of programs embedding SQL statements over the domain of *propositional* *formulae*, by considering variables' dependences up to each program point. In this context, we consider two more dependences, namely *database-database* and *program-database* dependences [5].

- As in [17, 18], we use a truth assignment function that assigns each of the database and application variables a truth value based on its sensitivity level[1], and checks the satisfiability of propositional formula in order to identify any possible information leakage.

- Finally, we analyze the application using a *numerical abstract domain* and perform reduced product of the *propositional formulae domain* and the *numerical abstract domain*, to remove possible false positives.

The structure of the paper is as follows: Section 2 illustrates a motivating example. Section 3 recalls some basics on the syntax of programs embedding SQL statements and defines their labeled concrete transition semantics. In section 4, we define an abstract domain of propositional formulae, the abstract labeled transition semantics of the applications and information leakage analysis. Section 5 describes an enhancement of the analysis. Finally, section 6 concludes the work.

2. MOTIVATING EXAMPLE

Consider a "Boarding Manager System" of a flight company that maintains the database *dB*, depicted in Figure 1. Table "Customer" stores personal information about each of the passengers, the total distance already traveled and the travel-points acquired by them (10 points are offered on the journey each 100 km). Assume that values of the attributes 'Address', 'Age', 'DistanceCovered' and 'Points' are private and should not be disclosed to unauthorized people.

Consider the program *P* in Figure 2, where the function `BookFlight()` inserts booking information into "Travel" table and updates the corresponding entries in "Customer" table, when any customer books a flight. A boarding-priority value is assigned to each journey based on the points acquired by the passenger. To distinguish from the database attributes, we prefix $ to the application variables in *P*. Assume that the information in Table "Travel" is public.

Finally, suppose the company decided to upgrade the customers having more than 50 'Points' to the status of 'Board Priority'. This is expressed in P by the activation of the `Upgrade()` function.

custID	custName	Address	Age	DistanceCovered	Points
1	Alberto	Athens	56	650	60
2	Matteo	Venice	68	49	0
3	Francesco	Washington	38	972	90
4	Smith	Paris	42	185	10

(a) Table "Customer"

custID	Source	Destination	FlightID	JourneyDate	BoardPriority
1	A	B	F139	26-04-14	2
2	C	D	F28	16-11-13	0
3	A	B	F139	26-04-14	3
4	A	B	F139	26-04-14	1

(b) Table "Travel"

Figure 1: Database *dB*

[1]We consider only two levels of sensitivity: public and private.

```
Function BookFlight()
1.   $flight=checkAvailability($source, $dest);
2.   if($flight ≠ NULL){
3.       $dist=computeDistance($source, $dest);
4.       UPDATE Customer SET DistanceCovered = DistanceCovered +
         $dist WHERE custID=$id;
5.       UPDATE Customer SET Points = Points + 10 ×
         FLOOR($dist/100) WHERE custID=$id;
6.       ResultSet rs = SELECT Points FROM Customer WHERE
         custID=$id;
7.       while(rs.next()){
8.           $point=rs.next().Points;
9.           $priority=getPriority($point);
10.          INSERT INTO Travel(userID, Source, Destination,
             FlightID, JourneyDate, BoardPriority) VALUES
             ($id,$source,$dest,$flight,$date,$priority);}}
End of Function BookFlight()

         ...
         ...
         ...

Function Upgrade()
15.  ResultSet rs = SELECT custID, DistanceCovered, Points FROM
     Customer WHERE Points>50;
16.  while(rs.next()){
17.      $id=rs.next().custID;
18.      $point=rs.next().Points;
19.      UPDATE Travel SET BoardPriority=BoardPriority +
         ($point-50)/10 WHERE custID=$id;}
End of Function Upgrade()
```

Figure 2: Program *P*

It is clear from the code that the values of 'BoardPriority' in tuples where 'custID' are equal to '1' and '3' will be upgraded from 2 to 3 and from 3 and 7 respectively. Therefore, an attacker can easily deduce the exact values of sensitive attribute 'Points' in Table "Customer", by observing the change that occurred in the public attribute 'BoardPriority' in Table "Travel".

The example above clearly shows that sensitive database information may be leaked through database applications when public attribute values depend, directly or indirectly, on private attribute values or private application variable values in the program. For instance, in the given example, the leakage occurs due to the dependence "Points→ BoardPriority" at program label 19.

3. CONCRETE SEMANTICS

By following [4], the basic functionality of SQL statements can be stated as "Any SQL statement Q first identifies an active data set from the database using a pre-condition that follows first-order logic, and then performs the appropriate operations on the selected data set". Data-intensive applications embedding SQL statements involve two distinct sets of variables: application variables $\mathbb{V}_a$ and database variables $\mathbb{V}_d$. Variables from $\mathbb{V}_d$ appear only in the SQL statements, whereas variables in $\mathbb{V}_a$ may appear in all types of instructions (either SQL or imperative).

3.1 Syntax

In this work, we consider a little variation of the language where programs are written by labeled instructions: each instruction is an ordered set of labeled-actions. The syntax is depicted in Table 1.

Let $in : \mathbb{C} \longmapsto \mathbb{L}$ and $fin : \mathbb{C} \longmapsto \mathbb{L}$ be two functions where $in[\![c]\!]$ and $fin[\![c]\!]$ denote the sets of initial and final labels of statement $c \in \mathbb{C}$ respectively. Let $\mathbb{A}$ be the set of

Constants:			
k	$\in$	$\mathbb{K}$	Set of Constants
k	$::=$	$n \mid s$	
		where $n \in$ Integers and $s \in$ Strings	
Variables:			
v_a	$\in$	$\mathbb{V}_a$	Set of Application Variables
v_a	$::=$	$x \mid y \mid z \mid \ldots$	
v_d	$\in$	$\mathbb{V}_d$	Set of Database Attributes
v_d	$::=$	$a_1 \mid a_2 \mid a_3 \mid \ldots$	
Expressions:			
e	$\in$	$\mathbb{E}$	Set of Arithmetic Expressions
e	$::=$	$c \mid v_d \mid v_a \mid op_u\ e \mid e_1\ op_b\ e_2$	
		where $op_u \in \{+,-\}$ and $op_b \in \{+,-,*,/\}$	
b	$\in$	$\mathbb{B}$	Set of Boolean Expressions
b	$::=$	$\text{true} \mid \text{false} \mid e_1\ op_r\ e_2 \mid \neg b \mid b_1 \oplus b_2$	
		where $op_r \in \{\leq, \geq, ==, >, \neq, \ldots\}$ and $\oplus \in \{\vee, \wedge\}$	
SQL Pre-conditions:			
τ	$\in$	$\mathbb{T}$	Set of Terms
τ	$::=$	$c \mid v_a \mid v_d \mid f_n(\tau_1, \tau_2, ..., \tau_n)$	
		where f_n is an n-ary function.	
a_f	$\in$	$\mathbb{A}_f$	Set of Atomic Formulas
a_f	$::=$	$R_n(\tau_1, \tau_2, ..., \tau_n) \mid \tau_1 == \tau_2$	
		where $R_n(\tau_1, \tau_2, ..., \tau_n) \in \{true, false\}$	
ϕ	$\in$	$\mathbb{W}$	Set of Pre-conditions
ϕ	$::=$	$a_f \mid \neg\phi \mid \phi_1 \oplus \phi_2 \mid \otimes v\ \phi$	
		where $\oplus \in \{\vee, \wedge\}$ and $\otimes \in \{\forall, \exists\}$	
SQL Functions:			
$g(\vec{e})$	$::=$	$\texttt{GROUP BY}(\vec{e}) \mid id$	
		where $\vec{e} = \langle e_1, ..., e_n \mid e_i \in \mathbb{E}\rangle$	
r	$::=$	$\texttt{DISTINCT} \mid \texttt{ALL}$	
s	$::=$	$\texttt{AVG} \mid \texttt{SUM} \mid \texttt{MAX} \mid \texttt{MIN} \mid \texttt{COUNT}$	
$h(e)$	$::=$	$s \circ r(e) \mid \texttt{DISTINCT}(e) \mid id$	
$h(*)$	$::=$	$\texttt{COUNT}(*)$	
		where * represents a list of database attributes denoted by $\vec{v_d}$	
$\vec{h}(\vec{x})$	$::=$	$\langle h_1(x_1), ..., h_n(x_n)\rangle$	
		where $\vec{h} = \langle h_1, ..., h_n\rangle$ and $\vec{x} = \langle x_1, ..., x_n \mid x_i = e \vee x_i = *\rangle$	
$f(\vec{e})$	$::=$	$\texttt{ORDER BY ASC}(\vec{e}) \mid \texttt{ORDER BY DESC}(\vec{e}) \mid id$	
Labeled Commands:			
ℓ	$\in$	$\mathbb{L}$	Set of Labels
Q	$\in$	$\mathbb{Q}$	Set of Labeled SQL Statements
Q	$::=$	SELECT $\mid$ UPDATE $\mid$ INSERT $\mid$ DELETE	
SELECT	$::=$	$\langle {}^{\ell_5}assign(v_a),\ {}^{\ell_4}f(\vec{e'}),\ {}^{\ell_3}r(\vec{h}(\vec{x})),\ {}^{\ell_2}\phi',\ {}^{\ell_1}g(\vec{e}),\ {}^{\ell_0}\phi\rangle$	
UPDATE	$::=$	$\langle {}^{\ell'}\vec{v_d} \overset{upd}{=} \vec{e},\ {}^{\ell}\phi\rangle$	
INSERT	$::=$	$\langle {}^{\ell'}\vec{v_d} \overset{new}{=} \vec{e},\ {}^{\ell}true\rangle$	
DELETE	$::=$	$\langle {}^{\ell'}del(\vec{v_d}),\ {}^{\ell}\phi\rangle$	
c	$\in$	$\mathbb{C}$	Set of Labeled Commands
c	$::=$	${}^{\ell}skip \mid {}^{\ell}v_a = e \mid Q \mid c_1; c_2$	
	$\mid$	$if\ {}^{\ell}b\ then\ c_1\ else\ c_2\ {}^{\ell'}endif$	
	$\mid$	$while\ {}^{\ell}b\ do\ c\ {}^{\ell'}done$	
P	$::=$	c^{ℓ}	Program that ends with label ℓ.

Table 1: Syntax of labeled programs embedding SQL

$$
\begin{aligned}
&\mathscr{A}[\![\text{SELECT}]\!] \overset{def}{=} \{{}^{\ell_5}assign(v_a),\ {}^{\ell_4}f(\vec{e'}),\ {}^{\ell_3}r(\vec{h}(\vec{x})),\ {}^{\ell_2}\phi',\\
&\qquad\qquad\qquad\qquad\qquad\qquad {}^{\ell_1}g(\vec{e}),\ {}^{\ell_0}\phi\}\\
&\mathscr{A}[\![\text{UPDATE}]\!] \overset{def}{=} \{{}^{\ell'}\vec{v_d} \overset{upd}{=} \vec{e},\ {}^{\ell}\phi\}\\
&\mathscr{A}[\![\text{INSERT}]\!] \overset{def}{=} \{{}^{\ell'}\vec{v_d} \overset{new}{=} \vec{e}\}\\
&\mathscr{A}[\![\text{DELETE}]\!] \overset{def}{=} \{{}^{\ell'}del(\vec{v_d}),\ {}^{\ell}\phi\}\\
&\mathscr{A}[\![{}^{\ell}skip]\!] \overset{def}{=} \{{}^{\ell}skip\}\\
&\mathscr{A}[\![{}^{\ell}v_a = e]\!] \overset{def}{=} \{{}^{\ell}v_a = e\}\\
&\mathscr{A}[\![if\ {}^{\ell}b\ then\ c_1\ else\ c_2\ {}^{\ell'}endif]\!] \overset{def}{=} \{{}^{\ell}b,\ {}^{\ell}\neg b,\ {}^{\ell'}endif\} \cup\\
&\qquad\qquad\qquad\qquad\qquad\qquad \mathscr{A}[\![c_1]\!] \cup \mathscr{A}[\![c_2]\!]\\
&\mathscr{A}[\![while\ {}^{\ell}b\ do\ c\ {}^{\ell'}done]\!] \overset{def}{=} \{{}^{\ell}b,\ {}^{\ell}\neg b,\ {}^{\ell'}done\} \cup \mathscr{A}[\![c]\!]\\
&\mathscr{A}[\![c_1;\ c_2]\!] \overset{def}{=} \mathscr{A}[\![c_1]\!] \cup \mathscr{A}[\![c_2]\!]
\end{aligned}
$$

Table 2: Definition of Action Function $\mathscr{A}$

actions, we define a function $\mathscr{A} : \mathbb{C} \longmapsto \wp(\mathbb{A})$ that returns the set of actions involved in program statements. The set of variables appearing in a statement is determined by the function $\mathscr{V} : \mathbb{C} \longmapsto \mathbb{V}$, where $\mathbb{V} = \mathbb{V}_a \cup \mathbb{V}_d$. Functions $\mathscr{A}$ and $\mathscr{V}$ are defined in table 2 and 3 respectively.

3.2 Semantics

Before defining semantics, lets recall from [4] the notion of *environments* correspond to the variables in $\mathbb{V}_a$ and $\mathbb{V}_d$ respectively. The variables take their values from semantic domain $\mathfrak{D}_\mho$, where $\mathfrak{D}_\mho = \{\mathfrak{D} \cup \{\mho\}\}$ and $\mho$ represents the undefined value.

An *application environment* $\rho_a \in \mathfrak{E}_a$ maps a variable $v \in dom(\rho_a) \subseteq \mathbb{V}_a$ to its value $\rho_a(v)$. So, $\mathfrak{E}_a \triangleq \mathbb{V}_a \longmapsto \mathfrak{D}_\mho$.

Consider a database as a set of indexed tables $\{t_i \mid i \in I_x\}$ for a given set of indexes I_x. A *database environment* is defined by a function ρ_d whose domain is I_x, such that for $i \in I_x,\ \rho_d(i) = t_i$.

Given a database environment ρ_d and a table $t \in d$. Assume $attr(t) = \{a_1, a_2, ..., a_k\}$. So, $t \subseteq D_1 \times D_2 \times .... \times D_k$ where a_i is the attribute corresponding to the typed domain D_i. A *table environment* ρ_t for a table t is defined as a function such that for any attribute $a_i \in attr(t)$, $\rho_t(a_i) = \langle \pi_i(l_j) \mid l_j \in t\rangle$, where π is the projection operator and $\pi_i(l_j)$ represents i^{th} element of the l_j-th row. In other words, ρ_t maps a_i to the ordered set of values over the rows of the table t.

A *state* $\sigma \in \Sigma \triangleq \mathbb{L} \times \mathfrak{E}_d \times \mathfrak{E}_a$ is denoted by a tuple $\langle \ell, \rho_d, \rho_a\rangle$ where $\ell \in \mathbb{L}$, $\rho_d \in \mathfrak{E}_d$ and $\rho_a \in \mathfrak{E}_a$ are the label of the statement to be executed, the database environment and the application environment respectively.

The set of *states* of a program P is, thus, defined as $\Sigma[\![P]\!] \triangleq \mathbb{L}[\![P]\!] \times \mathfrak{E}_d[\![P]\!] \times \mathfrak{E}_a[\![P]\!]$, where $\mathbb{L}[\![P]\!]$ is the set of labels in P, and $\mathfrak{E}_d[\![P]\!]$ and $\mathfrak{E}_a[\![P]\!]$ are the sets of database and application environments whose domain is the set of database and application variables in P only.

The *labeled transition relation* $\mathscr{T} : \Sigma \times \mathbb{A} \longmapsto \wp(\Sigma)$ specifies which successor states $\sigma' = \langle \ell', \rho_{d'}, \rho_{a'}\rangle \in \Sigma$ can follow when an action $a \in \mathbb{A}$ executes on state $\sigma = \langle \ell, \rho_d, \rho_a\rangle \in \Sigma$. We denote a labeled transition by $\sigma \xrightarrow{a} \sigma'$ or by $\langle \ell, \rho_d, \rho_a\rangle \xrightarrow{a} \langle \ell', \rho_{d'}, \rho_{a'}\rangle$, or by $\langle \ell, \rho\rangle \xrightarrow{a} \langle \ell', \rho'\rangle$ where ρ and ρ' represent (ρ_d, ρ_a) and $(\rho_{d'}, \rho_{a'})$ respectively.

The *labeled transition semantics* $\mathscr{T}[\![P]\!] \in \wp(\Sigma[\![P]\!] \times \mathscr{A}[\![P]\!] \longmapsto \wp(\Sigma[\![P]\!]))$ of a program P restricts the transition relation to program actions, *i.e.*

$$\mathscr{T}[\![P]\!]\sigma = \{\sigma' \mid \sigma \xrightarrow{a} \sigma' \wedge a \in \mathscr{A}[\![P]\!] \wedge \sigma, \sigma' \in \Sigma[\![P]\!]\}$$

The *labeled transition semantics* of various commands in database applications can easily be defined from the semantic description reported in [4].

Given a program P, let $\mathcal{I}_0 = \{\langle \ell, \rho_d, \rho_a\rangle \mid \ell \in in[\![P]\!] \wedge \langle \ell, \rho_d, \rho_a\rangle \in \Sigma[\![P]\!]\}$ be the set of initial states of P. The *partial trace semantics* [10] of P can be defined as

$$\mathscr{T}[\![P]\!](\mathcal{I}_0) = \text{lfp}^{\subseteq}_{\emptyset}\mathcal{F}(\mathcal{I}_0) = \bigcup_{i \leq \omega} \mathcal{F}^i(\mathcal{I}_0)$$

$$
\begin{aligned}
\text{where } \mathcal{F}(\mathcal{I}) = \quad & \lambda\mathcal{T}.\,\mathcal{I} \cup \Big\{\sigma_0 \xrightarrow{a_0} \ldots \xrightarrow{a_{n-1}} \sigma_n \xrightarrow{a_n} \sigma_{n+1} \mid \\
& \sigma_0 \xrightarrow{a_0} \ldots \xrightarrow{a_{n-1}} \sigma_n \in \mathcal{T} \wedge \sigma_n \xrightarrow{a_n} \sigma_{n+1} \in \mathscr{T}[\![P]\!]\Big\}
\end{aligned}
$$

4. ABSTRACT SEMANTICS

Intuitively, the main reason behind information leakage that possibly occur during program execution, is due to data- or control-dependences among program variables, either directly or indirectly.

As already mentioned, authors in [17, 18] used logical formuae to represent variables' dependences in the form

$$\bigwedge_{0 \leq i \leq n,\ 0 \leq j \leq m} \{y_i \rightarrow z_j\}$$

which means that the values of variable z_j possibly depend on the values of variable y_i. The domain of such logical formula, called *propositional formulae domain*, is treated as an abstract domain.

In case of applications embedding SQL statements, we need to consider two additional dependences, called *database-database* dependence and *program-database* dependence [5]. A *program-database* dependence arises between a database variable and an application variable, where values of the database variable depend on the value of the program variable or vice-versa. A *database-database* dependence arises between two database variables where the values of one depend on the values of the other.

Example 1. *Consider the database of Figure 1 in section 2. Consider the following SELECT query:*

Q_1 = ***SELECT*** *custName, Address, Age* ***INTO*** *v_a* ***FROM*** *Customer* ***WHERE*** *Points >=50*

Note that we use "`INTO` *v_a" in Q_1 to mention that the result of the query is finally assigned to v_a, where v_a is a Record or ResultSet type application variable with fields $\vec{w}$=$\langle w_1, w_2, w_3\rangle$. The type of w_1, w_2, w_3, are same as the return type of 'custName', 'Address', 'Age' respectively.*

Analyzing Q_1 we get the following variable dependences, in the form of propositional formula:

$Points \rightarrow V_a.w_1,\ Points \rightarrow V_a.w_2,\ Points \rightarrow V_a.w_3,\ custName \rightarrow V_a.w_1,\ Address \rightarrow V_a.w_2,\ Age \rightarrow V_a.w_3$

Let us consider another query Q_2:

Q_2 = ***SELECT*** *Points,* *AVG(Age)* ***INTO*** *v_a* ***FROM*** *Customer* ***WHERE*** *Points >=50* ***GROUP BY*** *Points* ***HAVING*** *SUM(DistanceCovered)>100* ***ORDER BY*** *Points*

$\mathcal{V}[\![c]\!]$	$\overset{def}{=}$	$\emptyset$
$\mathcal{V}[\![v]\!]$	$\overset{def}{=}$	$\{v\}$, where $v \in (\mathbb{V}_a \cup \mathbb{V}_d)$
$\mathcal{V}[\![\vec{v}]\!]$	$\overset{def}{=}$	$\bigcup\limits_{v_i \in \vec{v}} \mathcal{V}[\![v_i]\!]$
$\mathcal{V}[\![op_u\ e]\!]$	$\overset{def}{=}$	$\mathcal{V}[\![e]\!]$, where $op_u \in \{+, -\}$
$\mathcal{V}[\![e_1\ op_b\ e_2]\!]$	$\overset{def}{=}$	$\mathcal{V}[\![e_1]\!] \cup \mathcal{V}[\![e_2]\!]$, where $op_b \in \{+, -, *, /\}$
$\mathcal{V}[\![\vec{e}]\!]$	$\overset{def}{=}$	$\bigcup\limits_{e_i \in \vec{e}} \mathcal{V}[\![e_i]\!]$
$\mathcal{V}[\![true]\!]$	$\overset{def}{=}$	$\emptyset$
$\mathcal{V}[\![false]\!]$	$\overset{def}{=}$	$\emptyset$
$\mathcal{V}[\![e_1\ op_r\ e_2]\!]$	$\overset{def}{=}$	$\mathcal{V}[\![e_1]\!] \cup \mathcal{V}[\![e_2]\!]$, where $op_r \in \{\leq, \geq, ==, >, \neq, \dots\}$
$\mathcal{V}[\![\neg b]\!]$	$\overset{def}{=}$	$\mathcal{V}[\![b]\!]$
$\mathcal{V}[\![b_1 \oplus b_2]\!]$	$\overset{def}{=}$	$\mathcal{V}[\![b_1]\!] \cup \mathcal{V}[\![b_2]\!]$, where $\oplus \in \{\vee, \wedge\}$
$\mathcal{V}[\![f_n(\tau_1, \dots, \tau_n)]\!]$	$\overset{def}{=}$	$\mathcal{V}[\![\tau_1]\!] \cup \dots \cup \mathcal{V}[\![\tau_n]\!]$, where f_n is an n-ary function.
$\mathcal{V}[\![R_n(\tau_1, \dots, \tau_n)]\!]$	$\overset{def}{=}$	$\mathcal{V}[\![\tau_1]\!] \cup \dots \cup \mathcal{V}[\![\tau_n]\!]$, where $R_n(\tau_1, \tau_2, ..., \tau_n) \in \{true, false\}$
$\mathcal{V}[\![\tau_1 = \tau_2)]\!]$	$\overset{def}{=}$	$\mathcal{V}[\![\tau_1]\!] \cup \mathcal{V}[\![\tau_2]\!]$
$\mathcal{V}[\![\neg\phi]\!]$	$\overset{def}{=}$	$\mathcal{V}[\![\phi]\!]$
$\mathcal{V}[\![\phi_1 \oplus \phi_2]\!]$	$\overset{def}{=}$	$\mathcal{V}[\![\phi_1]\!] \cup \mathcal{V}[\![\phi_2]\!]$, where $\oplus \in \{\vee, \wedge\}$
$\mathcal{V}[\![\otimes v\ \phi]\!]$	$\overset{def}{=}$	$\{v\} \cup \mathcal{V}[\![\phi]\!]$, where $\otimes \in \{\forall, \exists\}$
$\mathcal{V}[\![\text{SELECT}]\!]$	$\overset{def}{=}$	$\mathcal{V}[\![\vec{v_a}]\!] \cup \mathcal{V}[\![\vec{e'}]\!] \cup \mathcal{V}[\![\vec{x}]\!] \cup \mathcal{V}[\![\phi']\!] \cup \mathcal{V}[\![\vec{e}]\!] \cup \mathcal{V}[\![\phi]\!]$
$\mathcal{V}[\![\text{UPDATE}]\!]$	$\overset{def}{=}$	$\mathcal{V}[\![\vec{v_d}]\!] \cup \mathcal{V}[\![\vec{e}]\!] \cup \mathcal{V}[\![\phi]\!]$
$\mathcal{V}[\![\text{INSERT}]\!]$	$\overset{def}{=}$	$\mathcal{V}[\![\vec{v_d}]\!] \cup \mathcal{V}[\![\vec{e}]\!]$
$\mathcal{V}[\![\text{DELETE}]\!]$	$\overset{def}{=}$	$\mathcal{V}[\![\vec{v_d}]\!] \cup \mathcal{V}[\![\phi]\!]$
$\mathcal{V}[\![{}^{\ell}skip]\!]$	$\overset{def}{=}$	$\emptyset$
$\mathcal{V}[\![{}^{\ell}v_a = e]\!]$	$\overset{def}{=}$	$\{v_a\} \cup \mathcal{V}[\![e]\!]$
$\mathcal{V}[\![if\ {}^{\ell}b\ then\ c_1\ else\ c_2\ {}^{\ell'}endif]\!]$	$\overset{def}{=}$	$\mathcal{V}[\![b]\!] \cup \mathcal{V}[\![c_1]\!] \cup \mathcal{V}[\![c_2]\!]$
$\mathcal{V}[\![while\ {}^{\ell}b\ do\ c\ {}^{\ell'}done]\!]$	$\overset{def}{=}$	$\mathcal{V}[\![b]\!] \cup \mathcal{V}[\![c]\!]$
$\mathcal{V}[\![c_1;\ c_2]\!]$	$\overset{def}{=}$	$\mathcal{V}[\![c_1]\!] \cup \mathcal{V}[\![c_2]\!]$

Table 3: Definition of Variables Function $\mathcal{V}$

Recall from Table 1 that the syntax of SELECT statement is defined as:

$$\langle {}^{\ell_5}assign(v_a),\ {}^{\ell_4}f(\vec{e'}),\ {}^{\ell_3}r(\vec{h}(\vec{x})),\ {}^{\ell_2}\phi',\ {}^{\ell_1}g(\vec{e}),\ {}^{\ell_0}\phi\rangle$$

According the syntax defined above, Q_2 can be formulated as:

Q_2 = `SELECT` $r(\vec{h}(\vec{x}))$ `INTO` $v_a(\vec{w})$ `FROM` *Customer* `WHERE` ϕ `GROUP BY`$(\vec{e})$ `HAVING` ϕ' `ORDER BY ASC`$(\vec{e'})$

where

- ϕ = *Points >=50*
- $\vec{e} = \langle Points\rangle$
- $g(\vec{e})$ = `GROUP BY`$(\langle Points\rangle)$
- ϕ' = (`SUM`$\circ$`ALL`*(DistanceCovered))>100*
- $\vec{h}$ = $\langle$`DISTINCT`, `AVG`$\circ$`ALL`$\rangle$
  $\vec{x} = \langle Points, Age\rangle$
  $\vec{h}(\vec{x})$ = $\langle$`DISTINCT`*(Points)*, `AVG`$\circ$`ALL`*(Age)*$\rangle$
- $\vec{e'} = \langle Points\rangle$
- $f(\vec{e'})$ = `ORDER BY ASC`$(\langle Points\rangle)$
- v_a = *Record or ResultSet type application variable with fields* $\vec{w}=\langle w_1, w_2\rangle$*. The type of* w_1 *and* w_2 *are same as the return type of* `DISTINCT`*(Points) and* `AVG`$\circ$`ALL`*(Age) respectively.*

From the query Q_2, we get the following set of variables dependences:

Points $\rightarrow V_a.w_1$, *Age* $\rightarrow V_a.w_2$, *Points* $\rightarrow V_a.w_2$, *DistanceCovered* $\rightarrow V_a.w_1$, *DistanceCovered* $\rightarrow V_a.w_2$

We now consider some other SQL commands and the corresponding variables' dependences:

Q_3=`UPDATE` *Customer* `SET` *DistanceCovered* = \$*y* + 150 `WHERE` *custID=2*
/ where \$y is an application variable. */*

Variable Dependences in Q_3: custID → DistanceCovered, \$y → DistanceCovered.

Q_4= `INSERT INTO` *Travel(custID,Source,Destination,FlightID, JourneyDate,BoardPriority)* `VALUES` *(5,"D","E","F34", \$y,\$z)*
/ where \$y and \$z are application variables. */*

Variable Dependences in Q_4: \$y → JourneyDate, \$z → BoardPriority.

Q_5 = `DELETE FROM` *Customer* `WHERE` *Age >60*

Variable Dependences in Q_5: Age → custID, Age → custName, Age → Address, Age → Age, Age → DistanceCovered, Age → Points.

The dependences above indicate explicit-flow of information. An example of implicit-flow that may occur in case of our application is, for instance, when manipulation of any public database information is performed under the control statements involving high variables.

4.1 Abstract states and semantics

An abstract state $\sigma^\sharp \in \Sigma^\sharp \equiv \mathbb{L} \times \texttt{Pos}$ is a pair $\langle \ell, \psi \rangle$ where $\psi \in \texttt{Pos}$ represents the set of dependences, in the form of propositional formula, among program variables up to label $\ell \in \mathbb{L}$.

The abstract labeled transition semantics $\mathscr{T}^\sharp[\![c]\!]$ of a statement c is a set of transitions between abstract states $\sigma_1^\sharp$ and $\sigma_2^\sharp$, denoted by $\sigma_1^\sharp \xrightarrow{c} \sigma_2^\sharp$. Table 4 depicts *abstract labeled transition semantics* of various statements in database applications, where the function $\texttt{BV}(c)$ denotes the "defined variables" in statement c. Let $\texttt{SF}(\psi)$ denotes the set of subformulas in ψ, and the operator $\ominus$ is defined by $\psi_1 \ominus \psi_2 = \bigwedge\left(\texttt{SF}(\psi_1) \backslash \texttt{SF}(\psi_2)\right)$.

The abstract semantics of the application is defined by fixpoint computation over the abstract domain [17].

4.2 Assigning Truth Values

Let $\Gamma : \mathbb{V} \rightarrow \{L, H\}$ be a function that assigns to each of the database and application variables in a program P a security class, either public (L) or private (H). We say that program P respects the confidentiality property, if and only if there is no information flow from private to public variables. To verify this property, a corresponding truth assignment function $\bar{\Gamma}$ is used:

$$\bar{\Gamma}(x) = \begin{cases} T & \text{if } \Gamma(x) = H \\ F & \text{if } \Gamma(x) = L \end{cases}$$

If $\bar{\Gamma}$ does not satisfy any propositional formula in ψ of an abstract state, the analysis will report a possible information leakage. For instance, in the motivating example of section 2, the dependence "Points→ BoardPriority" at program label 19 represents a propositional formulae with logical "implication". We get Γ(Points)=H and Γ(BoardPriority)=L. The truth assignment function $\bar{\Gamma}$ does not satisfy the formulae, as $\bar{\Gamma}$(Points)=T and $\bar{\Gamma}$(BoardPriority)=F and $T \rightarrow F$ is false. It is worthwhile to note that we have two different scenarios: (*i*) attackers are able to observe at the beginning and end of the computations and (*ii*) attackers are able to observe the public variables at each step of the computations. In the first case, the satisfiability is checked only on the terminal states, whereas in the later case the analysis is performed on each of the abstract states appeared in the traces that define semantics of the application.

5. ENHANCING THE ANALYSIS

The dependences that we considered so far are syntax-based, and may yield false positives in the analysis. For instance, although y syntactically depends on w in the statement "$y = 4w \bmod 2$", semantically there is no dependence as "$4w \bmod 2$" is always equal to zero.

To improve the accuracy of the analysis, authors in [17] proposed to use some popular *numerical abstract domains*, such as the *domain of intervals*, *polyhedra*, *octagons*, etc [2, 11]. The analysis in such abstract domains collect a sound approximation of the possible values of numerical variables during the execution of a program. Finally, a reduced product is performed between both the *propositional formulae domain* and the *numerical abstract domain*, in order to exclude pointless dependences for variables that have same value during the execution.

We now show two examples where such false-positives occur.

ID	*Name*	*Sal*
1	Alberto	1110
2	Matteo	1638
3	Francesco	2255
4	Smith	1840

(a) Table "*Emp*"

Type	*Rank*	*BASIC*	*HRA*	*DA*
Security	S1	800	20	65
Security	S2	600	20	65
Security	S3	520	15	65
Technical	T1	1000	25	75
Technical	T2	920	25	75
Technical	T3	880	20	75
Technical	T4	840	20	75
Admin	A1	1240	25	80
Admin	A2	1100	25	80

(b) Table "*Job*"

Figure 3: Database *dB*

Consider Table 3 and consider the following query:

Q_6 = `SELECT` *Type* `INTO` v_a `FROM` *Emp, Job* `WHERE` *Sal* = *BASIC*+(*BASIC* $*$ (*DA*/100))+(*BASIC* $*$ (*HRA*/100))

The following PD-dependences exist in Q_6:

ψ_6= $Sal \rightarrow V_a.w_1, BASIC \rightarrow V_a.w_1, DA \rightarrow V_a.w_1, HRA \rightarrow V_a.w_1, Type \rightarrow V_a.w_1$

Assuming '*Sal*', '*BASIC*', '*HRA*', '*DA*' are private and at least one employee in each job-type must exist, we see that although syntactic PD-dependences above indicating the presence of information leakage, but in practice nothing about these secrets is leaked through $V_a.w_1$.

Here is an another example of PD-dependence that is indicating false alarm on leakage: consider the code $\{\$x = 4 * \$w * \ log\ 2;$ `UPDATE` t `SET` $a = a + \$x; \}$. Assuming $\$x$ is private and $\$w$, a are public, we see that the dependence $\$x \rightarrow a$ generates false alarm as because $\$x$ is always equal to 0.

To remove all such false alarms and to increase the accuracy of the analysis, we analyze programs by using semantic-based abstract interpretation framework.

To illustrate, let us consider an abstract domain $\mathscr{N}$ where numerical attributes and numerical application variables are abstracted by the *domain of intervals*[2]. The abstraction yields an abstract query $Q_6^\sharp$ corresponding to Q_6 and an abstract database depicted in Figure 4.

$Q_6^\sharp$ = $\texttt{SELECT}^\sharp$ $Type^\sharp$ $\texttt{INTO}^\sharp$ $v_a^\sharp$ $\texttt{FROM}^\sharp$ $Emp^\sharp$, $Job^\sharp$ $\texttt{WHERE}^\sharp$ $Sal^\sharp =^\sharp BASIC^\sharp + (BASIC^\sharp * (DA^\sharp/[100, 100])) + (BASIC^\sharp * (HRA^\sharp/[100, 100]))$

The right-hand side expression of the condition in $\texttt{WHERE}^\sharp$ is evaluated to abstract values $[936, 1480]$, $[1638, 2000]$, and $[2255, 2542]$ respectively corresponding to the three abstract

[2]For other type of variables, the abstraction function represents identity function.

$\mathscr{T}^\sharp[\![\text{SELECT}]\!]$

$\overset{def}{=}\quad \mathscr{T}^\sharp[\![\langle {}^{\ell_5}assign(v_a),\ {}^{\ell_4}f(\vec{e}),\ {}^{\ell_3}r(\vec{h}(\vec{x})),\ {}^{\ell_2}\phi',\ {}^{\ell_1}g(\vec{e}),\ {}^{\ell_0}\phi\rangle]\!]$

$\overset{def}{=}\quad \{\langle \ell_0, \psi\rangle \xrightarrow{\text{SELECT}} \langle fin[\![\text{SELECT}]\!], \psi'\rangle\}$

where $\psi' = \bigwedge \left\{ y \rightarrow v_a.w_i \mid y \in (\mathscr{V}[\![\phi]\!] \cup \mathscr{V}[\![\vec{e}]\!] \cup \mathscr{V}[\![\phi']\!] \cup \mathscr{V}[\![\vec{e'}]\!]) \wedge v_a.w_i \in v_a.\vec{w} \wedge y \neq v_a.w_i \right\} \wedge$
$\bigwedge \left\{ z_i \rightarrow v_a.w_i \mid z_i \in \mathscr{V}[\![x_i]\!] \wedge\ x_i \in \vec{x} \wedge v_a.w_i \in v_a.\vec{w} \right\} \wedge \left(\psi \ominus \bigwedge\{u \rightarrow v_a.w_i \mid u \in \mathbb{V} \wedge v_a.w_i \in v_a.\vec{w}\} \right)$

$\mathscr{T}^\sharp[\![\text{UPDATE}]\!]$

$\overset{def}{=}\quad \mathscr{T}^\sharp[\![\langle {}^{\ell'}\vec{v_d} \overset{upd}{=} \vec{e},\ {}^{\ell}\phi\rangle]\!]$

$\overset{def}{=}\quad \{\langle \ell, \psi\rangle \xrightarrow{\text{UPDATE}} \langle fin[\![\text{UPDATE}]\!], \psi'\rangle\}$

where $\psi' = \bigwedge \left\{ y \rightarrow z \mid y \in \mathscr{V}[\![\phi]\!] \wedge z \in \vec{v_d} \right\} \wedge$
$\bigwedge \left\{ y_i \rightarrow z_i \mid y_i \in \mathscr{V}[\![e_i]\!] \wedge\ e_i \in \vec{e} \wedge z_i \in \vec{v_d} \right\} \wedge \psi$

$\mathscr{T}^\sharp[\![\text{INSERT}]\!]$

$\overset{def}{=}\quad \mathscr{T}^\sharp[\![\langle {}^{\ell'}\vec{v_d} \overset{new}{=} \vec{e},\ {}^{\ell}true\rangle]\!]$

$\overset{def}{=}\quad \{\langle \ell, \psi\rangle \xrightarrow{\text{INSERT}} \langle fin[\![\text{INSERT}]\!], \psi'\rangle\}$

where $\psi' = \bigwedge \left\{ y_i \rightarrow z_i \mid y_i \in \mathscr{V}[\![e_i]\!] \wedge e_i \in \vec{e} \wedge z_i \in \vec{v_d} \right\} \wedge \psi$

$\mathscr{T}^\sharp[\![\text{DELETE}]\!]$

$\overset{def}{=}\quad \mathscr{T}^\sharp[\![\langle {}^{\ell'}del(\vec{v_d}),\ {}^{\ell}\phi\rangle]\!]$

$\overset{def}{=}\quad \{\langle \ell, \psi\rangle \xrightarrow{\text{DELETE}} \langle fin[\![\text{DELETE}]\!], \psi'\rangle\}$

where $\psi' = \bigwedge \left\{ y \rightarrow z \mid y \in \mathscr{V}[\![\phi]\!] \wedge z \in \vec{v_d} \right\} \wedge \psi$

$\mathscr{T}^\sharp[\![{}^{\ell}skip]\!] \overset{def}{=} \{\langle \ell, \psi\rangle \rightarrow \langle fin[\![{}^{\ell}skip]\!], \psi\rangle\}$

$\mathscr{T}^\sharp[\![{}^{\ell}v_a = e]\!] \overset{def}{=} \{\langle \ell, \psi\rangle \rightarrow \langle fin[\![{}^{\ell}v_a = e]\!], \psi'\rangle\}$

where $\psi' = \bigwedge \left\{ y \rightarrow v_a \mid y \in \mathscr{V}[\![e]\!] \wedge y \neq v_a \right\} \wedge \left(\psi \ominus \bigwedge\{u \rightarrow v_a \mid u \in \mathbb{V}\} \right)$

$\mathscr{T}^\sharp[\![if\ {}^{\ell}b\ then\ c_1\ else\ c_2\ {}^{\ell'}endif]\!]$

$\overset{def}{=}\quad \mathscr{T}^\sharp[\![c_1]\!] \cup \mathscr{T}^\sharp[\![c_2]\!]$
$\bigcup\{\langle \ell, \psi\rangle \rightarrow \langle in[\![c_1]\!], \psi\rangle\} \bigcup\{\langle \ell, \psi\rangle \rightarrow \langle in[\![c_2]\!], \psi\rangle\}$
$\bigcup\{\langle \ell', \psi\rangle \rightarrow \langle fin[\![if\ {}^{\ell}b\ then\ c_1\ else\ c_2\ {}^{\ell'}endif]\!], \psi'\rangle\}$
$\bigcup\{\langle \ell', \psi\rangle \rightarrow \langle fin[\![if\ {}^{\ell}b\ then\ c_1\ else\ c_2\ {}^{\ell'}endif]\!], \psi''\rangle\}$

where $\psi' = \bigwedge \left\{ y \rightarrow z \mid y \in \mathscr{V}[\![b]\!] \wedge z \in \mathtt{BV}(c_1) \wedge y \neq z \right\} \wedge \psi$
where $\psi'' = \bigwedge \left\{ y \rightarrow z \mid y \in \mathscr{V}[\![b]\!] \wedge z \in \mathtt{BV}(c_2) \wedge y \neq z \right\} \wedge \psi$

$\mathscr{T}^\sharp[\![while\ {}^{\ell}b\ do\ c\ {}^{\ell'}done]\!]$

$\overset{def}{=}\quad \mathscr{T}^\sharp[\![c]\!] \bigcup\{\langle \ell, \psi\rangle \rightarrow \langle in[\![c]\!], \psi\rangle\} \bigcup\{\langle \ell', \psi\rangle \rightarrow \langle fin[\![while\ {}^{\ell}b\ do\ c\ {}^{\ell'}done]\!], \psi'\rangle\}$

where $\psi' = \bigwedge \left\{ y \rightarrow z \mid y \in \mathscr{V}[\![b]\!] \wedge z \in \mathtt{BV}(c) \wedge y \neq z \right\} \wedge \psi$

$\mathscr{T}^\sharp[\![c_1;\ c_2]\!] \overset{def}{=} \mathscr{T}^\sharp[\![c_1]\!] \cup \mathscr{T}^\sharp[\![c_2]\!]$

Table 4: Definition of Abstract Transition Function $\mathscr{T}^\sharp$

ID$^\sharp$	*Name*$^\sharp$	*Sal*$^\sharp$
1	Alberto	[1110, 1110]
2	Matteo	[1638, 1638]
3	Francesco	[2255, 2255]
4	Smith	[1840, 1840]

(a) Table "*Emp*$^\sharp$"

Type$^\sharp$	*Rank*$^\sharp$	*BASIC*$^\sharp$	*HRA*$^\sharp$	*DA*$^\sharp$
Security	[S1, S3]	[520, 800]	[15, 20]	[65, 65]
Technical	[T1, T4]	[840, 1000]	[20, 25]	[75, 75]
Admin	[A1, A2]	[1100, 1240]	[25, 25]	[80, 80]

(b) Table "*Job*$^\sharp$"

Figure 4: Database $dB^\sharp$

tuples in "*Job*$^\sharp$". Observe that, according to the assumption that at least one employee must exist in each job-type, there exist at least one '*Sal*$^\sharp$' in "*Emp*$^\sharp$" for which "$Sal^\sharp =^\sharp [936, 1480]$" is true, according to the following:

$$[l_i, h_i] =^\sharp [l_j, h_j] \triangleq \begin{cases} \text{true if } (l_i \geq l_j \wedge h_i \leq h_j) \\ \text{false if } h_i < l_j \vee l_i > h_j \\ \top \text{ otherwise} \end{cases}$$

Similar for "$Sal^\sharp =^\sharp [1638, 2000]$" and "$Sal^\sharp =^\sharp [2255, 2542]$". Therefore, the evaluation of Q_6 on dB always gives the same result *w.r.t.* the property $\mathscr{N}$, irrespective of the states of "*Emp*".

We can perform similar analysis of the code {$\$x = 4 * \$w * log\ 2$; UPDATE t SET $a = a + \$x$; } in the *domain of intervals*, yielding to "*no update*" of the values in public attribute a.

The interaction of the logical and numerical domains can be formalized by using the reduced Product [17]. Consider an abstract domain $\mathscr{N}$ where *numerical* values are abstracted by the *domain of intervals*. Suppose $\mathscr{T}^*$ and $\mathscr{T}^{*\sharp}$ represent the set of concrete and abstract traces respectively. Let $(\wp(\mathscr{T}^*), \alpha_0, \gamma_0, \wp(\mathscr{T}^{*\sharp}))$ and $(\wp(\mathscr{T}^*), \alpha_1, \gamma_1, \mathscr{N})$ be two Galois Connections, and let $\Upsilon : \wp(\mathscr{T}^{*\sharp}) \times \mathscr{N} \rightarrow \wp(\mathscr{T}^{*\sharp})$ be a reduced product operator defined as $\Upsilon(\mathcal{X}, \mathfrak{R}) = \mathcal{X}'$, where $\mathcal{X} \in \wp(\mathscr{T}^{*\sharp})$ is a set of partial traces, $\mathfrak{R} \in \mathscr{N}$, and

$$\mathcal{X}' = \Big\{ \langle \ell_i, \psi_k \rangle \mid \langle \ell_i, \psi_j \rangle \in \mathcal{X} \wedge \psi_k = (\psi_j \ominus \{x \rightarrow y \mid y \in \mathfrak{R}\}) \Big\}$$

In the example above, by analyzing Q_6 in the abstract domain $\mathscr{N}$ where numerical variables are abstracted by the *domain of intervals*, we see that the value of $v_a.w_1$ generated by Q_6 is always constant throughout the program execution *w.r.t.* $\mathscr{N}$. As $\psi_6 \in$Pos and $v_a.w_1 \in \mathscr{N}$, the reduced product operator Υ removes from ψ_6 all dependences in the form "$x \rightarrow v_a.w_1$" (that are representing false alarms), and makes the analysis more accurate and efficient.

6. CONCLUSIONS AND FUTURE PLANS

Information leakage detection becomes more challenging when we consider database query languages. As database queries often apply aggregate functions, we are now trying to enhance the framework with declassification policies [15]. We are also investigating the possible application of semantics-based analysis [5], aiming at removing more false positives.

Acknowledgement

Work partially supported by PRIN project "Security Horizons".

7. REFERENCES

[1] S. Cavadini. Secure slices of insecure programs. In *Proc. of the ACM symposium on Information, computer and communications security*, pages 112–122, Tokyo, Japan, 2008. ACM Press.

[2] L. Chen, A. Miné, and P. Cousot. A sound floating-point polyhedra abstract domain. In *Proc. of the 6th Asian Symposium on Programming Languages and Systems*, pages 3–18, Bangalore, India, 2008. ACM Press.

[3] D. E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19:236–243, 1976.

[4] R. Halder and A. Cortesi. Abstract interpretation of database query languages. *Computer Languages, Systems & Structures*, 38:123–157, 2012.

[5] R. Halder and A. Cortesi. Abstract program slicing of database query languages. In *Proc. of the 28th Annual ACM Symposium on Applied Computing*, pages 838–845, Coimbra, Portugal, 2013. ACM Press.

[6] C. Hammer. Experiences with pdg-based ifc. In *Proceedings of the Second int. conf. on Engineering Secure Software and Systems*, pages 44–60, Pisa, Italy, 2010. Springer-Verlag.

[7] C. Hammer and G. Snelting. Flow-sensitive, context-sensitive, and object-sensitive information flow control based on program dependence graphs. *International Journal of Information Security*, 8:399–422, 2009.

[8] B. Li. Analyzing information-flow in java program based on slicing technique. *SIGSOFT Softw. Eng. Notes*, 27:98–103, 2002.

[9] A. Lochbihler and G. Snelting. On temporal path conditions in dependence graphs. *Journal of Automated Software Engineering*, 16:263–290, 2009.

[10] F. Logozzo. Class invariants as abstract interpretation of trace semantics. *Computer Languages, Systems & Structures*, 35:100–142, 2009.

[11] A. Miné. The octagon abstract domain. *Higher Order Symbol. Comput.*, 19:31–100, 2006.

[12] A. C. Myers. Jflow: practical mostly-static information flow control. In *Proc. of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 228–241, San Antonio, Texas, USA, 1999. ACM Press.

[13] F. Pottier and V. Simonet. Information flow inference for ml. *ACM Transactions on Programming Languages and Systems*, 25:117–158, 2003.

[14] A. Sabelfeld and A. C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21:5–19, 2003.

[15] A. Sabelfeld and D. Sands. Declassification: Dimensions and principles. *Journal of Computer Security*, 17:517–548, 2009.

[16] G. Smith. Principles of secure information flow analysis. In *Malware Detection*, volume 27 of *Advances in Information Security*, pages 291–307. Springer US, 2007.

[17] M. Zanioli and A. Cortesi. Information leakage analysis by abstract interpretation. In *Proc. of the 37th int. conf. on Current trends in theory and practice of computer science*, pages 545–557, Slovakia, 2011. Springer LNCS 6543.

[18] M. Zanioli, P. Ferrara, and A. Cortesi. Sails: static analysis of information leakage with sample. In *Proc. of the 27th Annual ACM Symposium on Applied Computing*, pages 1308–1313, Trento, Italy, 2012. ACM Press.