Language-based Security Analysis of Database Applications

Raju Halder
Department of Computer Science and Engineering
Indian Institute of Technology Patna, India
halder@iitp.ac.in

Abstract—**In today's information-age, databases are at the heart of information systems. Unauthorized leakage of confidential database information, while computed by the associated database applications, may put the system at risk. Language-based information flow analysis is a promising field of research to detect possible information leakage in any software systems. So far, researchers pay little attention to the case of applications embedding database languages. In this paper, we address the need of proper analysis of data manipulation languages, and we overview the possible extension of language-based approaches to the case of information systems supporting databases at the back-end.**

Keywords-**Language-based Information Flow, Static Analysis, Information System, Database Query Languages**

I. Introduction

The prime objective of information systems security is to protect information by ensuring confidentiality, integrity, availability to an acceptable level [1]. For example, prevention of unauthorized disclosure of sensitive information, e.g. health records, credit history, banking transactions, online purchase history, personal information on social networking web services, etc. Standard security measures such as access control, encryption, etc. are mainly focused on controlling the release of information from sources or during the transmission over public channels. However, apart from these, sensitive data may be leaked maliciously or even accidentally, through a bug in the code, once it is released from the source and is propagated through the application legitimately. The following code-fragments depict two different scenarios of information leakage [2], [3]:

```
l := h                                  Explicit/Direct flow
if(h=0) l:=5; else l:=10;   Implicit/Indirect flow
```

Assuming the content of variables 'h' and 'l' are private and public respectively, the value of 'h' can be inferred by an attacker observing 'l' on the output channel.

As a major breakthrough, various language-based secure information flow analysis models are proposed, aiming at detecting possible information leakage in softwares [2], [4], [5], [6], [1], [7], [8]. Most of these works are based on the non-interference notion that says that a variation of confidential data given as input to a program does not cause any variation of publicly observable data [2], [9].

As databases play important role in most of the information systems, writing secure code for applications embedding database languages is really challenging in this context. So far, researchers pay little attention to this direction [10], [11]. In this paper, we address the need of proper analysis of data manipulation languages, and we overview the possible extension of language-based approaches to the case of information systems supporting databases at the back-end.

The rest of this paper is organized in the following manner. Section II recalls some basics on language-based information flow analysis. Section III briefly describes the existing works in the literature. In section IV, we discuss the possible extension to the case of information systems equipped with databases at the back-end. Finally, section V concludes.

II. Language-based Information flow Analysis

The aim of information flow analysis is to verify the confidentiality and the integrity of information while propagating along the control structure of computer programs during their execution [1]. Confidentiality refers to limiting the access and disclosure of sensitive information to authorized users only. For instance, when we purchase something online, our private data, e.g. credit card number, must be sent only to the merchant without disclosing to any third person during the transmission. Dually, the notion of integrity indicates that data or messages cannot be modified undetectably by any unauthorized person.

Works in this direction have been starting with the pioneer work of Denning in 1976 [12]. As a starting point, the analysis classified the program variables into various security classes. The simplest one is to consider two: Public/Low (denoted L) and Private/High (denoted H). Considering a mathematical lattice-model of security classes with order $L \leq H$, the secure information flow policy is defined on the lattice: an upward-flow in the lattice is only permissible to preserve confidentiality. Dually, in case of integrity, the lattice-model labels the

978-1-4799-4445-3/15/$31.00

variables as Tainted (denoted T) and Untainted (denoted U), and follows a dual flow-policy.

The basic semantic notion of secure information flow is non-interference principle that says "a variation of confidential data does not cause any variation to public data" [2], [9].

Definition 1 (Non Interference)**.** 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.

The non-interference principle is very restrictive and deemed to be impractical in practice. For instance, in password validation program all stored sensitive passwords are compared with user-given text and a boolean value is transmitted on the public channel as the result. To allow such intensional leakage, the notion of declassification [13] is introduced, where controlled information release is permitted.

Observe that an implicit flow is an indirect flow along the control structure of a program which acts as a covert channel and makes the security analysis more challenging. In general, some closely related security challenges on improper information leakage through various covert channels are [2]:

- *Termination channels* which provide information based on the termination or nontermination of computations.
- *Timing channels* which provide information based on the time at which an action occurs or the amount of time a computation takes.
- *Probabilistic channels* which provide information based on the probability distribution of observable data.
- *Resource exhaustion channels* which provides information by the possible exhaustion of finite, shared resources.
- *Power channels* which reveal information based on the amount of power consumed by the computing system.

The analysis requires to concern about the type of attackers observing the computing system (hence the type of covert channels) and the observational characteristics of attackers.

III. Related Works

A comprehensive survey on language-based information-flow analysis is reported in [2]. Most popular static analysis techniques are based on type systems ([2], [3], [14]), dependence graphs ([4], [5], [15], [6], [16]), formal approaches ([17], [7], [1], [18], [19]), etc. Besides the conservative nature of static analysis, the run-time monitoring systems detect unauthorized information flow dynamically; however, precision of the analysis completely depends on the execution overload, and of course, it is very prone to false negative [20], [21].

A. Security Type Systems ([2], [3], [14])

A type system is basically a formal system of type inference rules for making judgments about programs. In the context of language-based security analysis, the security type system considers various security types (*e.g.*, *low* and *high*) and a collection of typing rules which determine the type of expressions/commands to guarantee a secure information flow. Some of the typing rules from [2] are mentioned below:

- Expression Type: $\frac{}{\vdash exp:\ high}$ $\quad$ $\frac{h \notin \texttt{Var}(exp)}{\vdash exp:\ low}$
- Explicit-flow Rules: $\frac{\vdash exp:\ low}{[low] \vdash l := exp}$
- Implicit-flow Rules: $\frac{\vdash exp:\ pc \quad [pc] \vdash c_1 \quad [pc] \vdash c_2}{[pc] \vdash if\ \ exp\ \ then\ \ c_1\ \ else\ \ c_2}$
- Subsumption Rule: $\frac{[high] \vdash c}{[low] \vdash c}$

The notation [*pc*] denotes the security context which can be either [*low*] or [*high*]. According to the subsumption rule, if a program is typable in a high context then it is also typable in a low context. This allows to reset the program security context to low after a high conditional or a loop.

A major drawback of type-based approach is the lack of expressiveness. While this technique is provably sound, it is not flow-sensitive which may produce false alarm. For instance, consider the following code: {`if(h=1) then l:= 10; else l:= 5; ......; l:= 0;`}. Although the program is secure with respect to the classical noninterference principle as the output is always zero, but the type-based approach produces false alarm according to the implicit-flow rule.

B. Dependence Graphs ([4], [5], [15], [6], [16])

Program Dependence Graph (PDG) is an intermediate representation of program which makes explicit both the data- and control-flow in the program. As PDGs are flow-sensitive, the analysis improves *w.r.t.* the type-based approach. For instance, in PDG-based approaches, the code $\{$① `if(h=1) then` ② `l:= 10;` ③ `else l:= 5; ......;` ⑦ `l:= 0;`$\}$ is secure as there is no path ① $\xrightarrow{*}$ ⑦ in the corresponding PDG. Various extensions of PDG exist, for example System Dependence Graph (SDG) in case of inter-procedural call to capture context-sensitivity, Class Dependence Graph (ClDG) in case of Object-Oriented Languages to capture object-sensitivity on dynamic dispatch, etc [5]. Once the dependence graph of a program is constructed, static analysis is performed on the graph to identify the presence of possible insecure flow. An worth mentioning approach is backward slicing which collects all possible paths influencing the observable nodes: to be secure, the levels of variables in a path must not exceed the level of observable variable in the output-node of that path. Semantics-based improvement (*e.g.* path-conditions) is also proposed to disregard semantically unreachable paths [5].

C. Formal Approaches ([17], [7], [1], [18], [19])

Various formal approaches, *e.g.* Abstract Interpretation theory, Axiomatic Approaches, Model Checking, etc. are applied. Leino and Joshi [17] first introduced a semantics-based approach to analyzing secure information flow based on the semantic equivalence of programs. [1], [18] defined the concrete semantics of programs and lift it to an abstract domain suitable for flow analysis. In particular, they consider the domain of propositional formula representing variables' dependences. The abstract semantics is further refined by combining with numerical abstract domain which improves the precision of the analysis. A variety of logical forms are proposed to characterize information flow security. Amtoft and Banerjee [7] defined prelude semantics by treating program commands as prelude transformer. They introduced a logic based on the Abstract Interpretation of prelude semantics that makes independence between program variables explicit. The logic is based on Hoare logic. Recently, [19] proposed a model checking-based approach for reactive systems.

IV. Extension to Database Applications and Future Possibilities

Along with the advancement of web-based technologies, in today's information-age, information systems deal with a huge amount of sensitive information growing at an exponential rate day by day. For examples, online social networks, online banking systems, health information systems, credit-card industries, etc. The presence of data (either in structured or semi-structured form) in the underlying database and the applications embedding database languages open a new challenge for secure code writing of database applications to respect security principles. In practice, confidentiality of sensitive database information may be compromised while propagating through database applications accessing and processing them legitimately. However, so far, little attention is received in this direction to address such kind of leakage of database information via database applications [10], [11].

Consider the following update statement

$$\texttt{Q=UPDATE } t \texttt{ SET } l_1 = l_1 + 5 \texttt{ WHERE } h = 5$$

This update operation increases the values of an observable attribute l_1 for the tuples in which the value of private attribute h is 5. Therefore, observers can easily infer the secret value of the tuples by observing the changes in the values of l_1.

Similarly, due to the lack of proper care during code-writing, other database statements such as `SELECT`, `DELETE`, `INSERT` may also be the reason of information leakage.

In general, the approaches on secure information flow analysis make (implicitly or explicitly) use of variables' dependences and flow information. As we observe in section III that various approaches use various methods to capture such dependences and flow information. Same can be extended to the case of database statements as well. For instance, the above `UPDATE` statement represents the following two dependences:

$$l_1 \rightarrow l_1 \qquad h \rightarrow l_1$$

The following example illustrates the presence of variables' dependences in various database statements.

Example 1. *Consider the following* `SELECT` *query:*

$$Q_1 = \texttt{SELECT } Dno,\ Sal,\ Age,\ \texttt{FROM } emp\ \texttt{INTO } v_a \\ \texttt{WHERE } Sal > 1500$$

Note that we use "`INTO` v_a*" in Q_1 in order to indicate that the result of the query is finally assigned to v_a, where v_a is a resultset type 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 'Dno', 'Sal', 'Age' respectively. Analyzing Q_1 we get the following variables' dependences:*

$$Sal \rightarrow Dno,\ Sal \rightarrow Sal,\ Sal \rightarrow Age,\ Dno \rightarrow V_a.w_1,\\ Sal \rightarrow V_a.w_2,\ Age \rightarrow V_a.w_3$$

Let us consider another query Q_2:

$$Q_2 = \texttt{SELECT } Dno,\ \texttt{AVG}(Children)\ \texttt{FROM } emp\ \texttt{INTO } v_a \\ \texttt{WHERE } Sal > 1500\ \texttt{GROUP BY } Dno\ \texttt{HAVING AVG}(Age) < 40 \\ \texttt{ORDER BY } Dno$$

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

$$Sal \rightarrow Dno,\ Sal \rightarrow Children,\ Dno \rightarrow Dno,\ Dno \rightarrow Children,\ Age \rightarrow Dno,\ Age \rightarrow Children,\ Dno \rightarrow V_a.w_1,\\ Children \rightarrow V_a.w_2$$

Following are the examples of `INSERT` *and* `DELETE` *statements:*

$$Q_3 = \texttt{INSERT INTO } emp(eID, Name, Age, Dno, Sal, Children) \\ \texttt{VALUES } (y, \text{'}Michael\text{'}, 40, z, 3800, 3\rangle)$$

where y and z denote application variables.

$$Q_4 = \texttt{DELETE FROM } emp\ \texttt{WHERE } Dno = 2$$

The dependences in Q_3 are $y \rightarrow eID$, $z \rightarrow Dno$, and the dependences in Q_4 are

$$Dno \rightarrow eID,\ Dno \rightarrow Name,\ Dno \rightarrow Age,\ Dno \rightarrow Dno,\ Dno \rightarrow Sal,\ Dno \rightarrow Children$$

The recent works [10], [11], an extension of [1], use the Abstract Interpretation-based framework combining the symbolic domain of propositional formula and the numerical abstract domain. The analysis computes abstract semantics by collecting possible variables' dependences in the form of propositional formulae at each program point. The unsatisfiability of any formula, after truth assignment to variables based on the security levels, reports a possible information leakage.

This is worthwhile to mention that the information flow analysis in information systems pose great challenges when they are associated with database transactions involving commit and rollback operations, stored programs (*e.g.* triggers), etc. As future possibilities, there is a wide scope of research to deal such challenges, to explore possible alternatives for secure information flow analysis of database applications, and of course, to improve the accuracy and efficiency of the analysis. The PDG-based approaches, as an alternative, can be extended to the case of database applications considering Database Oriented Program Dependence Graph (DOPDG) [22], [23]. A fine-grained representation of variables' dependences in DOPDG may improve the analysis in terms of precision and efficiency. Furthermore, semantics-based approaches may also lead to an interesting direction in this context [24]. Due to the often use of aggregate operations in queries, there is a need of declassification framework to allow some information flow in practical scenarios. Overall, these challenges can be extended considering the existence of heterogenous databases in distributed environment.

V. Conclusions

In this paper, we discuss a new direction of information flow security analysis, in the context of information systems, when database applications interact with back-end databases.

Although language-based information flow security analysis has been studied for more than four decades, its practical application is relatively poor [25], [26], [27], [28]. This may be due to the restrictive nature of the proposed models. Some interesting challenges are expressiveness of languages, concurrency in multithreaded and distributed systems, database transactions involving commit and rollback operations, stored programs (*e.g.* triggers), etc.

References

[1] 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*. Nov Smokovec, Slovakia: Springer LNCS 6543, 2011, pp. 545–557.

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

[3] G. Smith, "Principles of secure information flow analysis," in *Malware Detection*, ser. Advances in Information Security, M. Christodorescu, S. Jha, D. Maughan, D. Song, and C. Wang, Eds., vol. 27. Nov Smokovec, Slovakia: Springer US, 2007, pp. 291–307.

[4] 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*, vol. 8, pp. 399–422, 2009.

[5] C. Hammer, J. Krinke, and G. Snelting, "Information flow control for java based on path conditions in dependence graphs," in *Proceedings of the IEEE International Symposium on Secure Software Engineering (ISSSE 2006)*. Arlington, VA: IEEE, 2006, pp. 87–96.

[6] S. Cavadini, "Secure slices of insecure programs," in *Proc. of the ACM symposium on Information, computer and communications security*. Tokyo, Japan: ACM Press, 2008, pp. 112–122.

[7] T. Amtoft and A. Banerjee, "A logic for information flow analysis with an application to forward slicing of simple imperative programs," *Science of Compututer Programming*, vol. 64, pp. 3–28, 2007.

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

[9] D. Hedin and A. Sabelfeld, "A perspective on information-flow control," in *Proceedings of the 2011 Marktoberdorf Summer School*. IOS Press, 2011.

[10] R. Halder, M. Zanioli, and A. Cortesi, "Information leakage analysis of database query languages," in *Proceedings of the 29th Annual ACM Symposium on Applied Computing (SAC'14)*. Gyeongju, Korea: ACM Press, 24–28 March 2014, pp. 813–820.

[11] A. Cortesi and R. Halder, "Information-flow analysis of hibernate query language," in *Proc. of the 1st Int. Conf. on Future Data and Security Engineering*. Vietnam: Springer LNCS 8860, 2014, pp. 262–274.

[12] D. E. Denning, "A lattice model of secure information flow," *Communications of the ACM*, vol. 19, pp. 236–243, 1976.

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

[14] D. Volpano, C. Irvine, and G. Smith, "A sound type system for secure flow analysis," *J. Comput. Secur.*, vol. 4, pp. 167–187, 1996.

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

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

[17] R. Joshi and K. R. M. Leino, "A semantic approach to secure information flow," *Sci. Comput. Program.*, vol. 37, no. 1-3, pp. 113–138, 2000.

[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 (SAC'12)*. Trento, Italy: ACM Press, 2012, pp. 1308–1313.

[19] R. Dimitrova, B. Finkbeiner, M. Kovács, M. N. Rabe, and H. Seidl, "Model checking information flow in reactive systems," in *Proceedings of the 13th International Conference on Verification, Model Checking, and Abstract Interpretation*. Philadelphia, PA: Springer-Verlag LNCS, 2012, pp. 169–185.

[20] P. Shroff, S. Smith, and M. Thober, "Dynamic dependency monitoring to secure information flow," in *Proceedings of the 20th IEEE Computer Security Foundations Symposium*, ser. CSF '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 203–217. [Online]. Available: http://dx.doi.org/10.1109/CSF.2007.20

[21] T. Bao, Y. Zheng, Z. Lin, X. Zhang, and D. Xu, "Strict control dependence and its effect on dynamic information flow analyses," in *Proc. of the 19th int. symposium on Software testing and analysis*. Trento, Italy: ACM Press, 2010, pp. 13–24.

[22] D. Willmor, S. M. Embury, and J. Shao, "Program slicing in the presence of a database state," in *Proc. of the 20th Int. Conf. on Software Maintenance*. IEEE CS, 2004, pp. 448–452.

[23] R. Halder and A. Cortesi, "Abstract program slicing of database query languages," in *Proc. of the the 28th Symposium On Applied Computing*. Coimbra, Portugal: ACM Press, 2013, pp. 838–845.

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

[25] A. C. Myers, "Jflow: practical mostly-static information flow control," in *Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL'99)*. San Antonio, Texas, USA: ACM Press, January 20–22 1999, pp. 228–241.

[26] V. Simonet and I. Rocquencourt, "Flow caml in a nutshell," in *Proceedings of the first APPSEM-II workshop*, 2003, pp. 152–165.

[27] M. Avvenuti, C. Bernardeschi, and N. De Francesco, "Java bytecode verification for secure information flow," *SIGPLAN Not.*, vol. 38, no. 12, pp. 20–27, 2003.

[28] G. Bian, K. Nakayama, Y. Kobayashi, and M. Maekawa, "Java bytecode dependence analysis for secure information flow," *International Journal of Network Security*, vol. 4, no. 1, pp. 59–68, 2007.