Static Detection of Security Flaws in Object-Oriented Databases

Keishi Tajima

Kyoto Univ. & Univ. of Tokyo

† currently at Kobe University

‡ supported by IISF

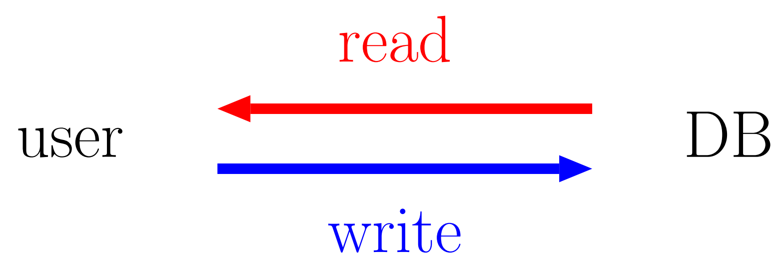
Background (1/4)

Database Access

User access to a database is either:

- actions to get information from the database, or
- actions to give information to the database.

They are usually represented as read and write operations.



Background (2/4)

Need for Control in Abstract Operation Level

- To give only partial information on some data.
- To allow update of some data only in specific procedure.

$$\Downarrow$$

Function-granularity Access Control

- "One can read this only through this function"
- "One can write this only through this function"

Background (3/4)

Example 1: allowing read operation only through a specific function

A job of checking budgets of stockbrokers:

function checkBudget(broker) =

>=(r_budget(broker), *(10, r_salary(broker)))

Background (4/4)

Example 2: allowing write operation only through a specific function

A job of updating salaries of stockbrokers:

```
function updateSalary(broker) =

    w_salary(broker,
             calcSalary(r_budget(broker), r_profit(broker)))
```

Problem

Security flaws in function-granularity access control

"Is that function effectively hiding read/write operations inside it?"

In Ex. 1:

- If one can know budgets of brokers, he can partially infer their salaries.
- If one can change budgets of brokers, he can totally infer their salaries.

In Ex. 2:

- If one can alter budgets of brokers, he can alter their salaries.

Goal of This Research

- To establish a foundation of security analysis for function-granularity access control.

- To develop a mechanism of static detection of security flaws.

Key Concepts

Inferability and Alterability

Generalization of read/write capability:

- inferability — ability to infer the result of a read operation
- alterability — ability to alter the value written in a write operation

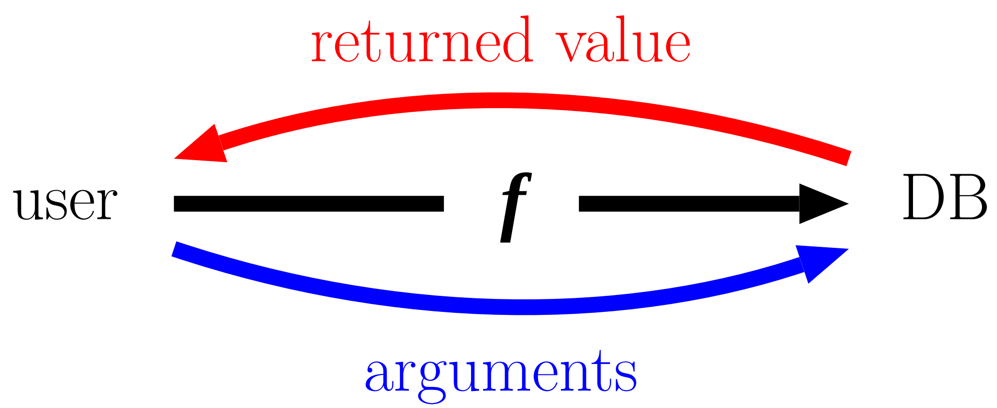
They are effectively equivalent to being able to read/write directly.

Inferability and Alterability (1/3)

Further generalization:

- inferability — ability to infer the returned value of a function invocation
- alterability — ability to alter the value of an argument of a function invocation

They correspond to two kind of capability in database access through functions.



Inferability and Alterability (2/3)

Classification of inrerability / alterability

- total inferability — ability to infer an exact value
- partial inferability — ability to infer some subsets

- total alterability — ability to alter it to any value in the domain
- partial alterability — ability to alter only within some subdomain

Inferability and Alterability (3/3)

Causality between capability

base cases

- inferability:
    1. constants,
    2. returned values of functions directly invoked by the user, or
    3. arguments of functions directly invoked by the user.
- alterability — arguments of functions directly invoked by the user

causality

- dependency between arguments and returned values of basic functions
- persistence
- alterability can cause inferability (e.g.: $>$, div, mod)

Static Detection of Security Flaws

Basic Strategy

1. Capability List — a set of functions one can invoke.
2. Security Requirement — a set of capability that he should not achieve.
3. We analyze programs of functions and determine whether each user can achieve specified capability.

$\boxed{\text{User Access through Functions}}$

Syntax of the function definition language

$$e ::= c \mid x \mid f_b(e, \ldots, e) \mid f_a(e, \ldots, e) \mid \mathsf{r\_}att(e) \mid \mathsf{w\_}att(e, e)$$

Query language

capability list = $\{\mathsf{r\_name}, \mathsf{r\_age}, \mathsf{profile}, \ldots\}$

select r_name(p), profile(p) from p $\in$ Person
where r_age(p) $>$ 20

Description of Security Requirements

An example of description

$(u,\ \mathsf{r\_salary}(\mathsf{employee} : \mathbf{pa}) : \mathbf{ti})$

u should not be able to invoke **r_salary(employee)** in the context where he can achieve

- partial alterability on the argument **employee**, and
- total inferability on the returned value.

Formal Definitions (1/2)

alterability on a_i of $f(\ldots, a_i, \ldots)$

$\exists\langle f_1, \ldots, f_n\rangle$

including indirect invocation of f, and

one can alter the value of a_i by changing the arguments of $f_1, \ldots, f_n$.

inferability on f

$\exists\langle f_1(v_1^1, \ldots, v_1^m) \to r_1, \quad \ldots, \quad f_n(v_n^1, \ldots, v_n^m) \to r_n\rangle$

including indirect invocation of f, and

an inference system $\mathcal{I}$ can infer the returned value of it.

Formal Definitions (2/2)

Inference system $\mathcal{I}$

$\mathcal{I}$ models users' inference on values of expressions in program codes.

$$term ::= [\langle e_1, \ldots, e_n \rangle \ \in \ S] \mid [e_1 \ = \ e_2]$$

Axioms and inference rules (part)

$$\rightarrow [\langle c \rangle \ \in \ \{c\}]$$
$$\rightarrow [\langle a_i^j \rangle \ \in \ \{v_i^j\}]$$
$$\rightarrow [\langle e_1, \ldots, e_n, f_b(e_1, \ldots, e_n) \rangle \ \in \ \{\langle v_1, \ldots, v_n, r \rangle \mid f_b(v_1, \ldots, v_n) = r\}]$$
$$[\langle e_i, e_j \rangle \in S_1], [\langle e_j, e_k \rangle \in S_2]$$
$$\rightarrow [\langle e_i, e_j, e_k \rangle \in \{\langle v_i, v_j, v_k \rangle \mid \langle v_i, v_j \rangle \in S_1, \langle v_j, v_k \rangle \in S_2\}]$$

Program Code Analysis (1/3)

Overview

1. We developed an inference system $\mathcal{J}$ which syntactically analyzes program codes and determine what capability users can achieve.

2. We compute a closure set of all terms deducable by $\mathcal{J}$.

3. If capability specified in security requirements are included in the closure set, we determine that there is a security flaw.

Program Code Analysis (2/3)

Inference system $\mathcal{J}$

$$term ::= \mathbf{ta}[e] \mid \mathbf{pa}[e] \mid \mathbf{ti}[e] \mid \mathbf{pi}[e] \mid {=}[e_1,\ e_2] \mid \ \ldots$$

Inference rule of $\mathcal{J}$ (part)

$$\begin{array}{l} \rightarrow \mathbf{ti}[c] \\ \rightarrow \mathbf{ta}[x] \quad \text{(argument of outermost function)} \\ \mathbf{ta}[e_3] \rightarrow \mathbf{ta}[\mathsf{r}\_att(e_2)] \quad \text{(if there exists } \mathsf{w}\_att(e_1,\ e_3)\text{)} \end{array}$$

Rules for basic functions are defined according to their semantics.

e.g.: rules for >= (part)

$$\begin{array}{lcl} \mathbf{pi}[e_1], \mathbf{pi}[e_2] & \rightarrow & \mathbf{ti}[>= (e_1, e_2)] \\ \mathbf{ti}[e_1], \mathbf{pa}[e_1], \mathbf{ti}[>= (e_1, e_2)] & \rightarrow & \mathbf{ti}[e_2] \end{array}$$

Program Code Analysis (3/3)

An example of analysis

function checkBudget(broker) $=$

$>=$(r_budget(broker), $*$(10, r_salary(broker)))

capability list of $u = \{$checkBudget, w_budget$\}$

security requirement $= (u,$ r_salary(broker):$\mathbf{ti})$

$$
\begin{array}{rcl}
 & \vdots & \\
=[\mathsf{v}, \mathsf{r\_budget(broker)}], \mathbf{ti}[\mathsf{v}] & \rightarrow & \mathbf{ti}[\mathsf{r\_budget(broker)}] \\
=[\mathsf{o}, \mathsf{broker}], \mathbf{pa}[\mathsf{v}] & \rightarrow & \mathbf{pa}[\mathsf{r\_budget(broker)}] \\
 & \rightarrow & \mathbf{ti}[>=(\ldots)] \\
\multicolumn{3}{l}{\mathbf{ti}[\mathsf{r\_budget(broker)}], \mathbf{pa}[\mathsf{r\_budget(broker)}], \mathbf{ti}[>=(\ldots)]} \\
 & \rightarrow & \mathbf{ti}[*(\mathsf{10}, \mathsf{r\_salary(broker)})] \\
 & \rightarrow & \mathbf{ti}[\mathsf{10}] \\
\mathbf{ti}[\mathsf{10}], \mathbf{ti}[*(\mathsf{10}, \mathsf{r\_salary(broker)})] & \rightarrow & \mathbf{ti}[\mathsf{r\_salary(broker)}]
\end{array}
$$

Conclusion

- We propose the concepts of inferability and alterability.
- We develop a method to statically determine whether given security requirements are satisfied or not.

Future work

- To include more complex program structures (conditional branch, recursion)
- More accurate analysis. Dynamic checking.