Information-Flow Security

Farzane Karami

Supervisor: Olaf Owe

Department of Informatics, University of Oslo

GEMINI IoT PhD-Seminar 15 May 2019

UiO : University of Oslo

Introduction

- Information-flow security
- Controlling how information is propagated by a system
- Preventing dissemination of confidential information

Introduction

- Information-flow security
- Controlling how information is propagated by a system
- Preventing dissemination of confidential information
- Access control

Introduction

- Information-flow security
- Controlling how information is propagated by a system
- Preventing dissemination of confidential information
- Access control
- Making sure that the program handles information securely

Information-flow security

- A language-based technique
- Tracking flow of information during a program execution
- Preventing leakage of confidential information

- An attacker is able to observe public outputs of a program
- Public outputs must be independent of secret inputs

Information-flow security

- A language-based technique
- Tracking flow of information during a program execution
- Preventing leakage of confidential information

- An attacker is able to observe public outputs of a program
- Public outputs must be independent of secret inputs
- Noninterference semantics [1]:
  - In two executions, a program is run with different secret inputs but the same public values, the public outputs will be the same.
  - An attacker cannot see any difference between these executions

Information-flow security

- Two kinds of flow of information
  - Explicit flow: $l := h$
  - Implicit flow:
    $l :=$ **true**; **if** h **then** $l :=$ **false**; **else** **skip**;

Information-flow security

Note: Techniques for enforcing information-flow security [2]

- Static secure type-systems:

Information-flow security

Note: Techniques for enforcing information-flow security [2]

- Static secure type-systems:
  - The types of program variables and expressions are augmented with security levels
  - Typing rules:
    - $\vdash exp : high$
    - $\dfrac{h \notin exp}{\vdash exp : low}$
    - $\dfrac{exp : low}{[low] \vdash l := exp}$

Information-flow security

Note: Techniques for enforcing information-flow security [2]

- Static secure type-systems:
  - The types of program variables and expressions are augmented with security levels
  - Typing rules:
    - $\vdash exp : high$
    - $\dfrac{h \notin exp}{\vdash exp : low}$
    - $\dfrac{exp : low}{[low] \vdash l := exp}$
  - Compiler

Information-flow security

Note: Techniques for enforcing information-flow security [2]

- Static secure type-systems:
  - The types of program variables and expressions are augmented with security levels
  - Typing rules:
    - $\vdash exp : high$
    - $\dfrac{h \notin exp}{\vdash exp : low}$
    - $\dfrac{exp : low}{[low] \vdash l := exp}$
  - Compiler
- Dynamic analysis: security checks are performed at run-time

Static vs dynamic enforcement

- Static techniques:
  - Less runtime overhead
  - Conservative
- Dynamic techniques:
  - More runtime overhead
  - The exact secrecy levels are available $\longrightarrow$ more precise
  - More permissive

$$\texttt{if}\ \ l < 0\ \ \texttt{then}\ \ l := 1;\ \ \texttt{else}\ \ l := h;$$

	ST	DT
Run-time efficiency	+	−
Exact security and permissiveness	−	+

Static vs dynamic enforcement

- Static techniques:
  - Less runtime overhead
  - Conservative
- Dynamic techniques:
  - More runtime overhead
  - The exact secrecy levels are available $\longrightarrow$ more precise
  - More permissive

$$\texttt{if } l < 0 \texttt{ then } l := 1; \texttt{ else } l := h;$$

	ST	**DT**
Run-time efficiency	+	−
Exact security and permissiveness	−	+

Information-flow security & Active object languages

- Distributed systems
- Active object languages
  - Scala/Akka
  - ABS/Creol
  - Rebeca
  - Encore
  - ASP

Information-flow security & Active object languages

- Distributed systems
- Active object languages
  - Scala/Akka
  - ABS/Creol
  - Rebeca
  - Encore
  - ASP
- Goal: To enforce information-flow security in a program
- Security aspects highly depend on communication paradigms between autonomous nodes

Active object languages

What are active object languages?

- A specific category of concurrent programming languages
- Active objects are created with their own threads, behaving autonomously
- They communicate with each other through method calls
  - **Asynchronous call (one-way)**: o!m(e)
  - **Synchronous call (two-way)**: x:=o.m(e)

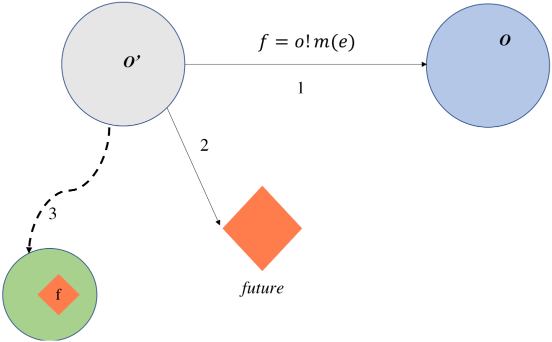
Communication paradigms

- Future mechanism: A flexible way of sharing results

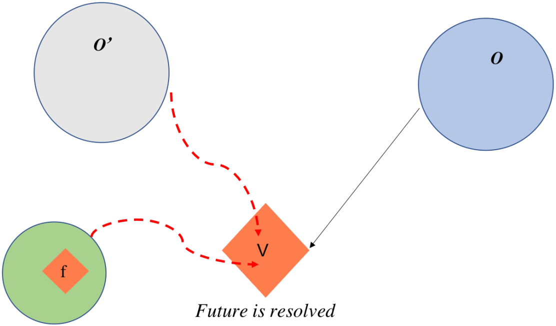
Communication paradigms

- Future mechanism: A flexible way of sharing results
  - **Futures**: f =:o!m(e)
  - A future is a placeholder created as a result of an asynchronous and remote method call
  - Eventually contains the result of the method call
  - When the caller needs the future value it requests it

First-class futures



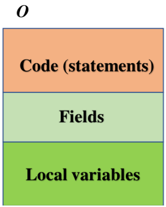
First-class futures



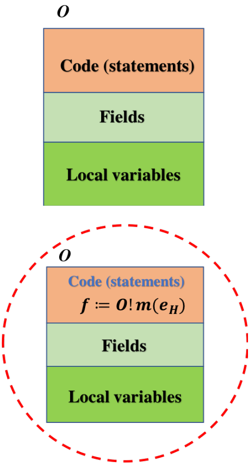
Wrappers

- Here we exploit the notion of wrapper to enforce information-flow security
- A wrapper is a kind of membrane defined around an object
- A wrapper controls security levels of communicated messages
- Preventing sending secret data to low level objects
- Confidentiality of a future

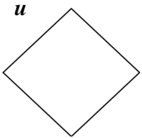
Run-time elements: objects



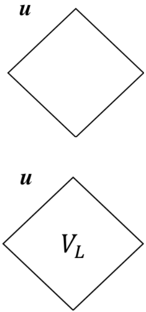
Run-time elements: objects



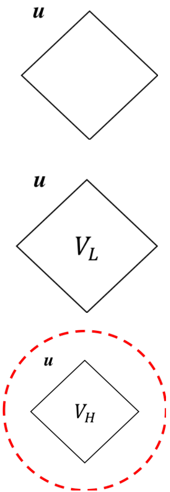
Run-time elements: futures



Run-time elements: futures

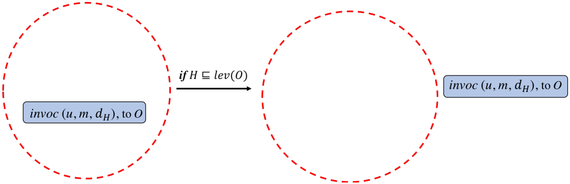


Run-time elements: futures



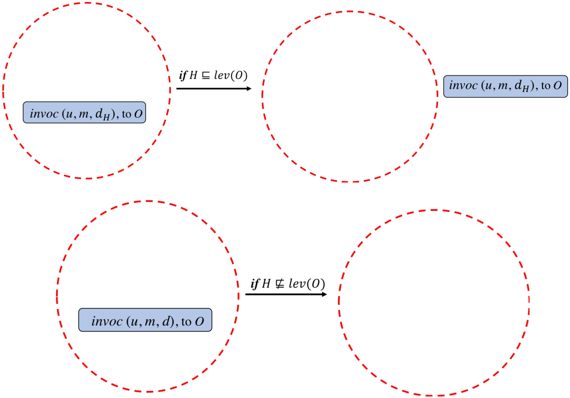
Invocation message / Caller side

- If at least one of the actual parameters is high

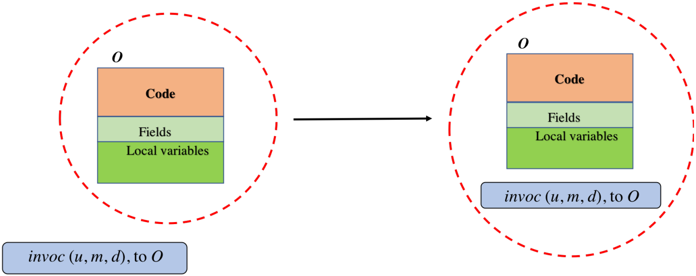


Invocation message / Caller side

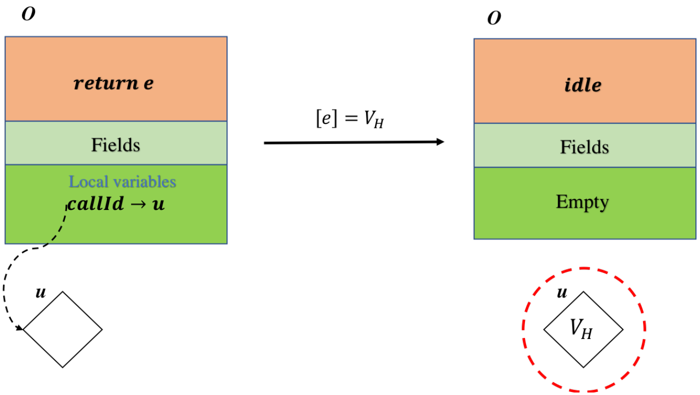
- If at least one of the actual parameters is high



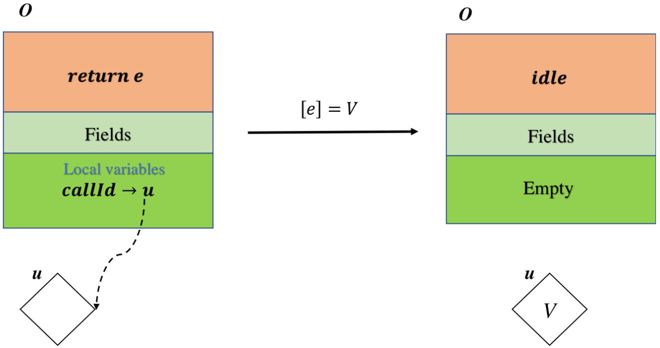
Invocation message / Callee side



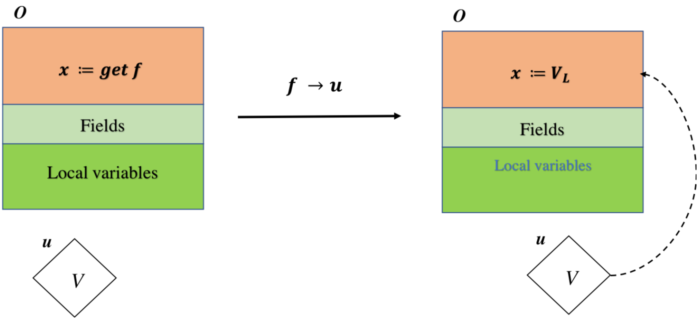
Method call / Callee side



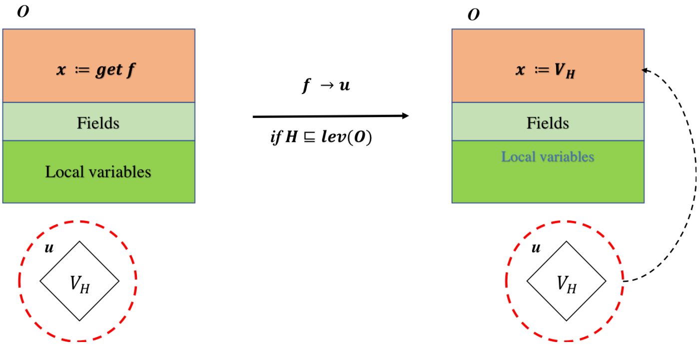
Method call / Callee side



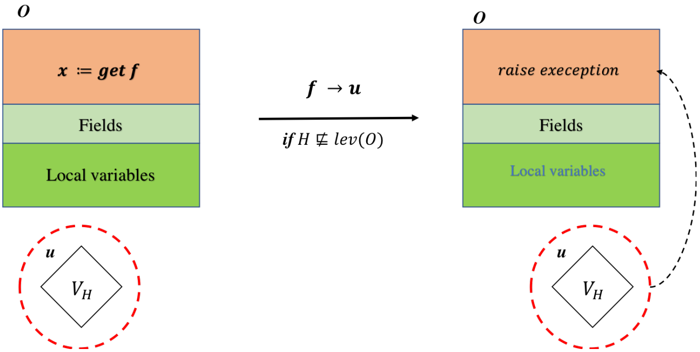
Get operation



Get operation



Get operation



UiO : University of Oslo

Conclusion

- A wrapper enforce dynamic information-flow security
- Runt-time checking for all objects in a system $\longrightarrow$ run-time overhead
- By combination of static analysis with dynamic checking to have less run-time overhead
- If statically it is shown that an object is safe$\longrightarrow$ it does not a wrapper for run-time checking

References

[1] Joseph A Goguen and José Meseguer.
Security policies and security models.
In *Security and Privacy, 1982 IEEE Symposium on*, pages 11–11. IEEE, 1982.

[2] Andrei Sabelfeld and Andrew C Myers.
Language-based information-flow security.
IEEE Journal on selected areas in communications, 21(1):5–19, 2003.

Thank You! :)