frama-C Software Analyzers

Program Transformation for Non-interference Verification on Programs with Pointers

Mounir Assaf (CEA LIST)
Julien Signoles (CEA LIST)
Frédéric Tronel (Supélec)
Éric Totel (Supélec)

GDR GPL - LTP, 2013-11-18

- Information security
  - Confidentiality
  - Integrity
  - Availability
- Traditionally, dissemination of information is prevented through access control
  - What piece of information can be accessed? by whom?
  - Yet, is this piece of information handled correctly when accessed?
- Information Flow Control
  - Tracks how information is propagated through a program
  - Verifies that information flows are secure

Information Flow Control

- Static analyses:
  - Seminal work [Denning & Denning,77]
  - First formalization and soundness proof for a simple imperative language [Volpano et al.,96]
  - Jif: IFC extension to Java language [Myers et al.,01]
  - Flow Caml : IFC extension to OCaml [Simonet et al.,03]
- Dynamic analyses:
  - Operating system level [Enck et al.,10], [Andriatsimandefitra et al.]
  - Application level [Hiet et al.,09], [Austin & Flanagan,09 & 10]
- Hybrid analyses:
  - [Leguernic et al.,07], [Russo & Sabelfeld,10], [Chandra & Franz, 07], [Nair et al., 08], [Besson et al., 13]

Information Flow Control

- Provable secure information flow monitoring:
  - A gap between theoretical toy languages and real life languages [Leguernic et al,07], [Russo & Sabelfeld,10]
  - Previous monitoring approaches considering languages with rich constructs do not consider proving soundness [Chandra & Franz,07], [Nair et al.,08]
  - Pointer-induced flows not that much investigated [Moore & Chong,11], [Austin & Flanagan,09]
  - No monitor inlining approach considering pointers [Chudnov & Naumann,10], [Magazinius et al.,12]
- Our approach
  - Sound hybrid information flow monitor
  - Sound inlining approach

for a language with pointers and aliasing

Frama-C Software Analyzers

Information flows

Monitor Semantics

Monitor Inlining

Conclusion

Explicit flows

- produced whenever information is transfered directly from source to destination

destination = source

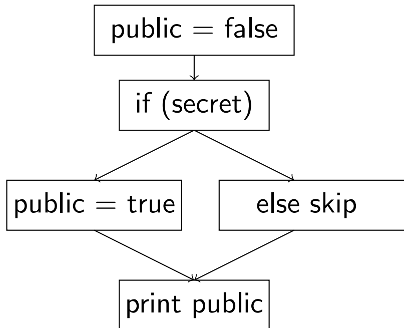
← - - - - - - - - - - - - - - - -

Assignments generate explicit flows

- Explicit flow from *source* to *destination*

Implicit flows

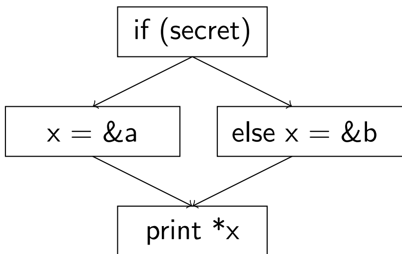
- produced "whenever" an assignment is conditioned on the value of an expression



- Implicit flow from variable *secret* to variable *public*

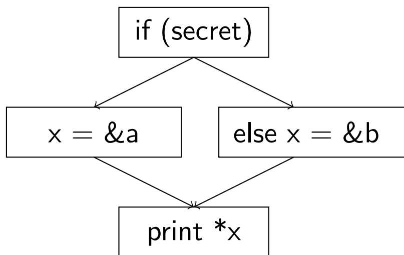
Pointer-induced flows

- produced whenever a pointer is dereferenced



Pointer-induced flows

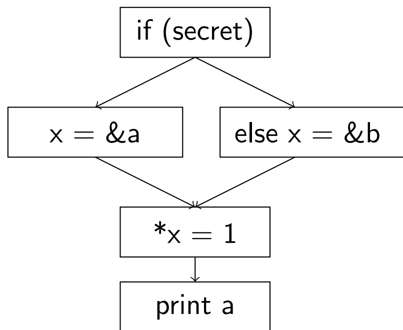
- produced whenever a pointer is dereferenced



- Implicit flow from *secret* to *pointer x*
- Pointer-induced flow from *pointer x* to $*x$
- Information flow from *secret* to $*x$.

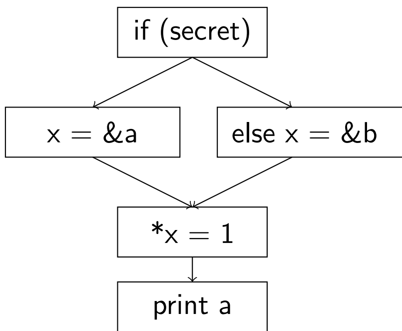
Pointer-induced flows

- produced whenever a pointer is dereferenced



Pointer-induced flows

- produced whenever a pointer is dereferenced



- Assignment $*x = 1$ generates pointer-induced flows from *pointer x* to all variables that *x may point to*

Attacker model

- They know the program source code and public outputs
- They control public inputs

Security Levels :

S : secret

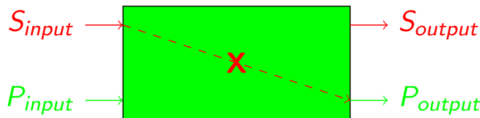
P : public

$P \rightarrow S$

Non-interference

Attacker model

- They know the program source code and public outputs
- They control public inputs

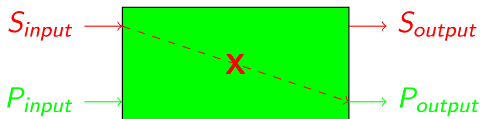


Security Levels :

S : secret
P : public
$P \to S$

Attacker model

- They know the program source code and public outputs
- They control public inputs
- Roughly, non-interference is a security property stating non-dependence of public outputs from secret inputs (in the case of confidentiality)



Security Levels :

S : secret

P : public

P → S

Information flows

Monitor Semantics

Monitor Inlining

Conclusion

Monitor semantics

'Clight' semantics [Leroy & Blazy,09]

Instruction semantics

$$E \vdash c, M \quad \Rightarrow M'$$

l-value evaluation
(address)

$$E \vdash a, M \quad \Leftarrow loc$$

r-value evaluation
(contents)

$$E \vdash a, M \quad \Rightarrow val$$

Monitor semantics

Extended 'Clight' semantics [Leroy & Blazy,09]

- Memory Γ: a memory mapping a location to a security label
- Tracking information flows by tainting security labels

Instruction semantics

$$E \vdash c, M, \Gamma, \underline{pc} \Rightarrow M', \Gamma'$$

l-value evaluation
(address)

$$E \vdash a, M, \Gamma \Leftarrow loc, s_{loc}$$

r-value evaluation
(contents)

$$E \vdash a, M, \Gamma \Rightarrow val, s_{val}$$

Eg. Right value evaluations

Right value evaluations of an expression

$$M \triangleq \{l_x \mapsto ptr(l_y); l_y \mapsto v\}$$

$$RV \frac{LV_{MEM} \dfrac{E \vdash x, M \Rightarrow ptr(l_y)}{E \vdash *x, M \Leftarrow l_y} \qquad M(l_y) = v}{E \vdash *x, M \Rightarrow v}$$

Right value evaluations of an expression

- The label associated to the l-value of a is propagated to the one associated to its r-value
- "Program Transformation for Non-interference Verification on Programs with Pointers" [Assaf et al., IFIP SEC 2013]

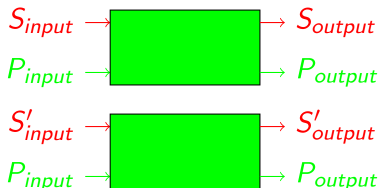
$$M \triangleq \{l_x \mapsto ptr(l_y); l_y \mapsto v\}$$

$$\Gamma \triangleq \{l_x \mapsto s_x; l_y \mapsto s_y\}$$

$$RV \frac{LV_{MEM} \dfrac{E \vdash x, M, \Gamma \Rightarrow ptr(l_y), s_x}{E \vdash *x, M, \Gamma \Leftarrow l_y, s_x} \qquad M(l_y) = v \qquad \Gamma(l_y) = s_y \qquad s = s_y \bigsqcup s_x}{E \vdash *x, M, \Gamma \Rightarrow v, s}$$

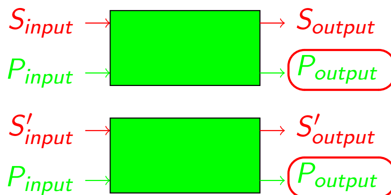
Theorem 1: Soundness with respect to termination insensitive non-interference

- Two terminating executions differing only on secret inputs deliver the same public outputs



Theorem 1: Soundness with respect to termination insensitive non-interference

- Two terminating executions differing only on secret inputs deliver the same public outputs



Information flows

Monitor Semantics

Monitor Inlining

Conclusion

The intuition

Encapsulating the semantics of the security memory Γ into the program

$\underline{pc} = \underline{public} \sqcup \underline{secret}$

```
if (public == secret) {
    auth = 1
```

$\quad \underline{auth} = \underline{pc} \sqcup public$

$\quad \underline{log_fail} = \underline{log_fail} \sqcup \underline{pc}$

```
} else {
    auth = 0
```

$\quad \underline{auth} = \underline{pc} \sqcup \underline{public}$

```
    log_fail = 0
```

$\quad \underline{log_fail} = \underline{log_fail} \sqcup \underline{pc}$

```
}
```

$assert\ \underline{auth} \sqsubseteq public$

$\mathtt{output}_{public}\ \mathtt{auth}$

```
int auth
label auth
int *leak
label leak
label* leak_p1
leak = &auth
```

$\underline{leak} = public$

$\underline{leak_p1} = \&\underline{auth}$

assert $\underline{leak} \sqcup *\underline{leak_p1} \sqsubseteq public$

output$_{public}$ *leak

- Aliasing Lemma: two expressions are aliased iff their respective auxiliary variables are aliased.

A Program Transformation

- Instrumenting the program to track the security level of each data handled by programs
- A security analysis through
  - Hybrid monitoring by running the transformed program T(P)
  - Static analysis techniques using off-the-shelf tools such as Value Analysis Frama-C's plugin
- Theorem 2: soundness wrt. the initial program behavior
- Theorem 3: soundness wrt. the monitor semantics (hence wrt. non-interference)

Information flows

Monitor Semantics

Monitor Inlining

Conclusion

Conclusion

- A sound hybrid information flow monitor for a language supporting pointers and aliasing
- A sound inlining approach for our monitor based on a program transformation
- Future work:
  - Completing the prototype implementation of our Frama-C plug-in, case study
  - Extending the formalization to richer C constructs
    - Pointer arithmetics, declassification annotations, arrays
    - Function calls, dynamic allocations, casts. . .
  - Ongoing work on quantitative information flow