An information-flow calculus for the non-security expert

Alejandro Russo (russo@chalmers.se)
Visiting Associate Professor, Stanford, CA, U.S.A.
Chalmers, Göteborg, Sweden

Work-in-progress with Pablo Buiras (Chalmers), Deian Stefan (Stanford), and David Mazierès (Stanford)
Information-flow Scenario

Preserve confidentiality even in the presence of malicious code
Motivation
Security measures

- Access control
  - State-of-the-art
Security lattice

- It specifies the allowed flows of information
Example of Rules

Arrows for Secure Information Flow

[Li, Zdancewic 10]

\[ \Phi \vdash c : l \rightarrow l' \]

\[ \Phi_1 \vdash c_1 : l_1 \rightarrow l_2 \]
\[ \Phi_2 \vdash c_2 : l_3 \rightarrow l_4 \]

\[ \Phi_1 \cup \Phi_2 \cup \{l_2 \sqsubseteq l_3\} \vdash c_1 \ggg c_2 : l_1 \rightarrow l_4 \]
Information-flow Scenario
Towards a Monadic Calculus

Reader Monad
- The attacker might observe the systems data

Writer Monad
- The attacker writes input to the system

Information flow control is almost just about controlling reading and writing side-effects
Towards a Monadic Calculus

Restricted interface for the State monad!

Security State
(floating label)
A Floating Label System

[Stefan et al. 11]

Taint

\[ l'_f = l_f \sqcup l_f \]

Guard

\[ l'_f \subseteq l_d \]
Example of Rules

Arrows for Secure Information Flow

\[ [\text{Li, Zdancewic 10}] \]

\[ \Phi \vdash c : l \rightarrow l' \]

\[ \Phi_1 \vdash c_1 : l_1 \rightarrow l_2 \]
\[ \Phi_2 \vdash c_2 : l_3 \rightarrow l_4 \]

\[ \Phi_1 \cup \Phi_2 \cup \{ l_2 \preceq l_3 \} \vdash c_1 \ggg c_2 : l_1 \rightarrow l_4 \]

Guard
Example of Rules

Information-Flow Security for a Core of JavaScript

[Hedin, Sabelfeld 12]

\[ o = E[r] \quad o = \{ s_{l_3} \mapsto v_{l}, \ldots \} \]

\[ \text{GetOwnProperty}(r_{l_1}, s_{l_2}) = v_{l_{l_1}l_{l_2}l_{l_3}} \]

Taint
### Designing IFC Systems

<table>
<thead>
<tr>
<th>Function</th>
<th>Read effect</th>
<th>Write effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>newIORef</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>readIORef</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>writeIORef</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>modifyIORef</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

### Guard vs. Taint

<table>
<thead>
<tr>
<th>Function</th>
<th>Read effect</th>
<th>Write effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>newLIORef</td>
<td></td>
<td>Guard</td>
</tr>
<tr>
<td>readLIORef</td>
<td>Taint</td>
<td></td>
</tr>
<tr>
<td>writeLIORef</td>
<td>Guard</td>
<td></td>
</tr>
<tr>
<td>modifyLIORef</td>
<td>Taint</td>
<td>Guard</td>
</tr>
</tbody>
</table>
The IFC Monad

[Swierstra 08]

data ReadEffect l a where
Taint :: l -> a -> ReadEffect l a

data WriteEffect l a where
Guard :: l -> a -> WriteEffect l a

Free (ReadEffect l) a
Free (WriteEffect l) a
IFC l a = Free (WriteEffect l :+: ReadEffect l) a

Types reflects the behavior w.r.t taint and guard!
Label Creep

[Stefan et al. 11][Breeze 13]

- The floating label gets too high too soon!
The floating label gets too high too soon!
A specific Local (Reader Monad)

data Env l a where 
    Ask :: (l -> a) -> Env l a 
    Put :: l -> Env l a 

IFC l a = Free (WriteEffect l :+: 
    ReadEffect l :+: 
    Env l) a 

local :: forall l. Label l => IFC l () -> IFC l () 
local m = do (s :: l) <- IFC.ask 
    m 
    IFC.put s 
    return ()
Is it General Enough?

LIO

LB-Monitors

IFC
Final Remarks

- IFC = controlling reading and writing side-effects + a notion of scope (local)
  
  ```haskell
  type IFC l a = Free (ReadEffect l :+: WriteEffect l :+: Env l) a
  ```

- A non-security expert can have a good impression of the security checks (taint/guard)

- Floating label systems seems to be more convenient than traditional LB-monitors
Interested in Details?

https://github.com/alejandrorusso/ifc-wg2.8.git