Background

- Denning and Denning, Certification of Programs for Secure Information Flow, CACM 20(7), July 1977
- Presentation summarized in Bishop Chapter 15
Program analysis

- What if we try to track information flow within a program?
- We have access control for files, processes and users
  - what about variables?

Explicit flows

- $x := 17$
- $l := h$
- $h := l$
Implicit flows

- How can we write \( l := h \)?
- Assume \( l \) and \( h \) are Booleans
  - if \( h \) then \( l := true \) else \( l := false \)
  - \( l := true \); if not \( h \) then \( l := false \) else skip
  - \( l := false \); while \( h \) do \( l := true \)

Simple “while” language

- Sabelfeld and Myers Figures 2 and 3
  - \( C \) ::= \( \text{skip} \)
  - \( \text{var} := \text{exp} \)
  - \( C1; C2 \)
  - if \( \text{exp} \) then \( C1 \) else \( C2 \)
  - while \( \text{exp} \) do \( C \)
Type system

- Judgment forms:
  - Every variable in exp is at or below level
    \[ \vdash \text{exp} : \text{level} \]
  - Every assignment in C is at or above pc
    \[ \lbrack \text{pc} \rbrack \vdash \text{C} \]

Inference Rules

\[
\begin{align*}
\text{[E1–2]} & \quad \vdash \text{exp} : \text{high} \quad h \notin \text{Vars(exp)} \quad \vdash \text{exp} : \text{low} \\
\text{[C1–3]} & \quad \lbrack \text{pc} \rbrack \vdash \text{skip} \quad \lbrack \text{pc} \rbrack \vdash l := \text{exp} \quad \vdash \text{exp} : \text{low} \quad \lbrack \text{low} \rbrack \vdash l := \text{exp} \\
\text{[C4–5]} & \quad \lbrack \text{pc} \rbrack \vdash C_1 \quad \lbrack \text{pc} \rbrack \vdash C_2 \quad \vdash \text{exp} : \text{pc} \quad \lbrack \text{pc} \rbrack \vdash \text{while} \ \text{exp} \ \text{do} \ C \quad \lbrack \text{pc} \rbrack \vdash C \\
\text{[C6–7]} & \quad \vdash \text{exp} : \text{pc} \quad \lbrack \text{pc} \rbrack \vdash C_1 \quad \lbrack \text{pc} \rbrack \vdash C_2 \quad \lbrack \text{high} \rbrack \vdash C \quad \lbrack \text{low} \rbrack \vdash C
\end{align*}
\]
What is a flow?

- A variable of confidential input does not cause a variation of public output

Simple Program

- Multiplication by repeated addition

\[
\begin{align*}
\{ a,b \geq 0 \} \\
x &:= a; \\
r &:= 0; \\
\text{while } (x > 0) \text{ do} \\
&\quad x := x - 1; \\
&\quad r := r + b \\
\{ r = a \times b \} \\
\end{align*}
\]

Direct Flows:
- a -> x
- b -> r

Indirect Flow:
- x -> r
Exercise

1. \texttt{h := not l}
2. \texttt{h := if l then false else true}
3. \texttt{if l then h := false else h := true}
4. \texttt{h := true; if l then h := false else skip}
5. \texttt{l := not h}
6. \texttt{l := if h then false else true}
7. \texttt{if h then l := false else l := true}
8. \texttt{l := true; if h then l := false else skip}

Theoretical results

- Volpano, Irvine and Smith (JCS '96) showed Soundness
  - “If an expression \( e \) can be given a type \( \tau \) in our system, then Simple Security says ... that only variables at level \( \tau \) or lower in \( e \) will have their contents read when \( e \) is evaluated (no read up)....
  - On the other hand, if a command \( c \) can be given a type \([\tau]\mid\cdot\) then Confinement says ... that no variable below level \( \tau \) is updated in \( c \) (no write down).”
Information Flow Languages

- Two serious implementations of information-flow languages
  - Jif = Java + Information Flow
    - Andrew Myers and others, Cornell
    - [http://www.cs.cornell.edu/jif/](http://www.cs.cornell.edu/jif/)
  - FlowCaml
    - Vincent Simonet
    - [http://cristal.inria.fr/~simonet/soft/flowcaml/](http://cristal.inria.fr/~simonet/soft/flowcaml/)

FlowCaml

- An ML-style language with type inference
- Windows executable flowcaml gives an interactive type checker
  - Note: It does not execute the programs, batch compiler flowcamlc compiles them
Declaring values

let x = 1;;
let x1 : !alice int = 42;;
let x2 : !bob int = 53;;

Anonymous functions and lists

let succ = function x -> x + 1;;
let half = function x -> x lsr 1;;
let l1 = [1; 2; 3; 4];;
let l2 = [x1; x2];;
Defining functions

let rec length = function
    [] -> 0
  | _ :: tl -> 1 + length tl;;

let rec mem0 = function
    [] -> false
  | hd :: tl -> hd = 0 || mem0 tl
    ;;

Demo
Does it work?

• In practice it is not broadly adopted
  – Technical issue is the complexity of managing policy
  – I suspect there are social issues as well ...
    the technical issues are not show stoppers

Recall

• Consider an example (in no particular language)

  H = readHighDatabase()
  L = readLowUserInput()
  If f(H,L)
    then printLow “Success”
    else printLow “Fail”

• Assume H is high and L is Low
But!!!

- Consider an example (in no particular language)
  
  \[ H = \text{readHighDatabase} \left( \text{"passwd"} \right) \]
  
  \[ L = \text{readLowUserInput}() \]

  \[ \text{If checkPassword(H,L)} \]
  
  \[ \text{then printLow \text{"Success"} } \]
  
  \[ \text{else printLow \text{"Fail"} } \]

- We do this every day!

Password checking paradox

- Why shouldn’t we allow someone to write the password program?
- Why should we?
Policy

• The password paradox is solved by explicit policy
• Similar issues arise with crypto algorithms
  – LoCypher = encrypt (HighClear, goodKey)
• Cf.
  – LoCypher = encrypt (HighClear, badKey)

FlowCaml and Policy

• FlowCaml solves the policy problem by dividing the program into two parts:
  – Flow caml portion (.fml), with all flows checked
  – Regular caml portion with an annotated interface
• The downgrading of encryption or password validation queries is not done within the flow-checked portion
Policy

• Zdancewic uses other techniques, including explicit downgrade assertions for confidentiality

• Basic philosophy: uniform enforcement with explicit escape mechanism
  – Focus analysis on the exceptions

Further reading