Secure Architecture Principles

Information flow control

D. Denning and P. Denning

Certification of Programs for Secure Information Flow

(CACM 1976)

Review Access Control

- Discretionary access control (DAC)
  - Philosophy: users have the discretion to specify policy themselves
  - Commonly, information belongs to the owner of object
  - Access control lists, privilege lists, capabilities
- Mandatory access control (MAC)
  - Philosophy: central authority mandates policy
  - Information belongs to the authority, not to the individual users
  - MLS and BLP, Chinese wall, Clark-Wilson, etc.

Beyond Access Control

- Malicious program could do (after passing ACL):
  - Write information into a public temp file
  - Use IPC to communicate with process run by attacker
  - Leak information in metadata (billing reports, nonces chosen in protocols, ...)
  - Use shared resources and OS API to encode information (e.g., file locking, CPU cycles)
- Secure information flow: control propagation of sensitive data after it has been accessed

Information-flow control Model

- Set S of subjects
- Set O of objects
- Set L of security labels
  - Function “+” that combines security labels:
    - $\ell 1 + \ell 2$ is label of information derived from $\ell 1$ and $\ell 2$
    - + is associative and commutative
- Function L(X) that gives label of entity (subject or object) X
  - labels might be static: don't change throughout execution
  - or dynamic: label of entity changes based on history of execution

IFC example lattice: Two points

- L = {low, high} (called Label or Classification)
- ℓ1 + ℓ2 =
  - low if ℓ1=ℓ2=low
  - high otherwise
- bottom = low
- Top, ⊤ = high
- low → high, low → low, high → high
- think of this as MLS with only...
  - Unclassified (low) and Top Secret (high)
  - no compartments
- simple and captures important ideas, so use of two-point lattice is standard in information-flow literature

Information Flow Within Programs

- Access control for program variables
  - Finer-grained than processes

- Use program analysis to prove that the program has no undesirable flows

Explicit and Implicit Flows

- Goal: **prevent information flow from "high" variables to "low" variables**

- Flow can be **explicit** …

  h := <secret>

  x := h

  l := x

- … or **implicit**

  boolean h := <secret>

  if (h) { l := true} else { l := false }

Compile-Time Certification

- Declare classification of information allowed to be stored in each variable
  - x: integer class { A,B }
- Classification of function parameter = classification of argument
- Classification of function result =
  - union of parameter classes
- Certification becomes type checking!

Assignments and Compound statements

- Assignment: left-hand side must be able to receive all classes in right-hand side

  x = w+y+z requires $L\{w,y,z\} = L(w) + L(y) + L(z) \leq L(x)$

- Compound statement

  begin

  x = y+z;

  a = b+c –x

  end

  requires $L\{y,z\} \leq L(x)$ and $L\{b,c,x\} \leq L(a)$

Conditionals and Functions

- Conditional:

  classification of “then/else” must contain classification of “if” part (why?)

- Functions:

```
int sum (int x class{A}) {
    int out class{A,B} ;
    out = out + x;
}
```

requires $A \leq B$ and $B \leq B$

Iterative Statements

- In iterative statements, information can flow from the absence of execution

  while $f(x_1, x_2, \ldots, x_n)$ do S

  – Information flows from variables in the conditional statement to variables assigned in S (why?)
- For an iterative statement to be secure …
  – Statement terminates
  – Body S is secure
  – $L\{x_1, x_2, \ldots, x_n\} \leq L\{\text{target of an assignment in S}\}$

Non-Interference

- (informal) Definition (from Wikipedia)
  - a computer is modeled as a machine with inputs and outputs. Inputs and outputs are classified as either *low* or *high*
  - A computer has the non-interference property *if and only if* any sequence of low inputs will produce the same low outputs, regardless of what the high level inputs are

Non-Interference

[Goguen and Meseguer]

- Observable behavior of the program should not depend on confidential data
  - Example: private local data should not “interfere” with network communications

Declassification

- Non-interference can be too strong
  - Programs release confidential information as part of normal operation
  - "Alice will release her data after you pay her $10"
- Idea: allow the program to release confidential data, but only through a certain computation
- Example: logging in using a secure password

  `if (password == input) login(); else fail();`
  - Information about password must be released …

    … but only through the result of comparison

Covert channel

- Password checking (CWE-385)

```
def validate_password(actual_pw, typed_pw):
    if len(actual_pw) <> len(typed_pw):
        return 0
    for i in len(actual_pw):
        if actual_pw[i] <> typed_pw[i]:
            return 0
return 1
```

- Does Low input (typed_pw) produce the same low output in terms of (time taken to validate_password(), return value)?