Language-Based Information-Flow Security

By Andrei Sabelfield and Andrew C. Myers, presented by Matt McCutchen
Information-flow security goals

- Confidentiality - focus of this survey
  - Loss is permanent
- Integrity
  - Can be recovered by restoring from trusted backups
  - Sometimes hard to distinguish from "correctness"
End-to-end confidentiality

- Process receives H (confidential) & L (public) inputs, produces H & L outputs
  - Could have more than 2 labels - later
- Attacker sees the L outputs
- Attacker should not be able to learn (much about) the H inputs
Noninterference

• Changing H inputs "does not change" L outputs
  • E.g.: Given the L inputs, the probability distributions of L outputs produced from different H inputs are computationally indistinguishable
  • At the level of denotation of a program fragment:
    \( \forall s_1, s_2. \ s_1 \approx_L s_2 \Rightarrow [[C]](s_1) \approx_L [[C]](s_2) \)

• Strongest possible property
Compared to access control

- Access control works in conjunction with often-unstated assumptions about information-flow behavior of the principals granted access
  - Ex: Web server has read access to CGI source code, should not send it to clients
- Better to verify the desired information flow properties directly
Types of information flows

- **Explicit**: assignments
  
  \[ l = h; \]

- **Implicit**: control
  
  \[
  \text{if } (h == 1) \\
  \quad l = 1; \\
  \text{else} \\
  \quad l = 0;
  \]

- **"Covert"**: everything else
  
  \[
  t1 = \text{time}(); \\
  \text{if } (h == 1) \\
  \quad \text{sleep(1);} \\
  l = \text{round(time()} - t1);
  \]
Language-based techniques

- Data types include H or L label
- Statements labeled to track implicit flows
- Compile-time and/or load-time verification of safety according to a set of rules
Example security type system

- Variables labeled H or L
- Statements labeled H or L
  - Substatements must have same label, but "H blocks" can be embedded in L
  - Top level is L
  - L statements cannot access H variables
  - H statements cannot modify L variables

```plaintext
int{L} x = 3;
int{H} y = 5, z = 0;
if (x < 5) {
    x *= 2;
    H {
        while (y > 0) {
            z += x * y;
            y--;
        }
    }
}
```
Noninterference proof idea

• Since:
  • L code does not use H variables
  • H code does not modify L variables

We can remove the H code and variables, and the L code will still execute the same way.

• L outputs are a function only of L inputs
More formally...

\[ \vdash \text{exp} : H \quad \frac{\text{exp has no H vars}}{\vdash \text{exp} : L} \]

\[ \vdash \text{[u]} \vdash \text{v := exp} \quad \frac{\text{v labeled u}}{\vdash \text{exp} : u} \]

\[ \vdash \text{[u]} \vdash \text{C}_1 \quad \vdash \text{[u]} \vdash \text{C}_2 \quad \frac{\text{(Seq)}}{\vdash \text{[u]} \vdash \text{C}_1; \text{C}_2} \]

\[ \vdash \text{g : u} \quad \vdash \text{[u]} \vdash \text{C}_1 \quad \vdash \text{[u]} \vdash \text{C}_2 \quad \frac{\text{(If)}}{\vdash \text{[u]} \vdash \text{if g then C}_1 \text{ else C}_2} \]

\[ \vdash \text{g : u} \quad \vdash \text{[u]} \vdash \text{C} \quad \frac{\text{(While)}}{\vdash \text{[u]} \vdash \text{while g do C}} \]

\[ \vdash \text{g : u} \quad \vdash \text{[H]} \vdash \text{C} \quad \frac{\text{(HBlock)}}{\vdash \text{[L]} \vdash \text{C}} \]

int\{L\} \ x = 3;
int\{H\} \ y = 5, \ z = 0;

S_1 \ if \ (x < 5) \ \{}
S_2 \ \ x \ *= \ 2;
S_3 \ \ H \ \{}
S_4 \ \ \ while \ (y > 0) \ \{}
S_5 \ \ \ \ z \ += \ x \ * \ y;
S_6 \ \ \ \ y--;
S_7 \ \ \ \ \} \\
\}
Bad code is rejected

• \( l = h; \)
  Must be in an H block to read \( h \), but then \( l \) cannot be written.

• \( \text{if } (h == 1) \)
  \( l = 1; \)
  \( \text{else} \)
  \( l = 0; \)

  The entire if must be in an H block to read \( h \), but then \( l \) cannot be written.
More sophisticated languages

- Exceptions
- Label polymorphism
  ```java
class Counter<u> {
    int{u} count;
    void inc():{u} {
      count++;
    }
}
```
- Runtime representation of labels, dynamically checked conversions
  - C.f. Java Class<T>, #cast
Covert channels

- Any way information can travel that we didn't model
Covert channels: examples

- Simple clock read
  ```
  t1 = time();
  if (h == 1)
      sleep(1);
  l = time() - t1;
  ```

- Order dependence
  ```
  flag = 1;
  if (h == 1)
      sleep(1);
  l = flag;
  ```
Covert channels: examples

- Hyper-Threading CPU cache

```c
if (h == 1)
    process_buf(b);

Evicts some of "big" from cache
```

```c
process_buf(big);

Fills cache
```

```c
t1 = time();
process_buf(big);
t2 = time();
l = (t2-t1 > threshold);
```
Dealing w/ covert channels

- Have to cut through the graph (well enough)
- Make timings independent of H data
  - "if" based on H data:
    - Compute the running time of both branches
    - If shorter branch is taken, sleep to the longer time
  - Cannot handle "while" based on H data
- Does not scale!
- CPU cache?
- Allow only order-independent parallel computation
- Many more techniques - later
Declassification policies

- May need to declassify H-derived data to L
  - Part of system's intended function
    - Election results
  - Computationally negligible leaks
    - Password checking
      ```
      String{H} password;
      bool{L} checkPassword(String{L} guess) {
        return (guess == password);
      }
      ```
    - Cryptographic algorithms
      ```
      <u> Key{u} newKey();
      <u> Bytes{L} encrypt(Bytes{u}, Key{u});
      ```
Practical issues

- Feasibility of writing verified programs for realistic problems
- Would like to verify and run untrusted code
- Composition of processes in a system