Software Security
Information Flow
(Chapter 5 of lecture notes on language-based security)

Erik Poll
Digital Security group
Radboud University Nijmegen

Rules for expressions

e : t means **e contains information of level t or *lower***

- **variable** x:t if x is a variable of type t
- **operations**
  $$\frac{e:t \qquad e':t}{e+e' : t}$$
  for some binary operation + (similar for n-ary)
- **subtyping**
  $$\frac{e:t \qquad t \leq t'}{e:t'}$$

Rules for commands

s : ok t means s only writes to – ie. leaks to –level t or *higher*

- assignment
$$\frac{e : t \qquad x \text{ is a variable of type } t}{x{:=}e \ : ok\ t}$$
- composition
$$\frac{c1 : ok\ t \qquad c2 : ok\ t}{c1;c2 \ : ok\ t}$$
- if-then-else
$$\frac{e : t \qquad c1 : ok\ t \qquad c2 : ok\ t}{\text{if } e \text{ then } c1 \text{ else } c2 \ : ok\ t}$$
- while
$$\frac{e : t \qquad c : ok\ t}{\text{while } e \text{ do } c \ : ok\ t}$$
- subtyping
$$\frac{c : ok\ t \qquad t \geq t'}{c : ok\ t'}$$

NB ok t $\leq$ ok t' iff t $\geq$ t'
(anti-monotonicity)

Beware

Beware of the confusing difference in directions

e : t **means** **e contains information of level t or *<u>lower</u>***

s : ok t **means** **s only writes to level t or *<u>higher</u>***

For people familiar will Bell – LaPadula access control :
there you have the same confusion,
in the “no read up” & “no write down” rules

The tricky issues

Implicit flows

```
if (hi) { lo = ... }
while (hi) { lo = ... }
```

are problematic

But isn't

```
if (hi) { ... }
while (hi) { lo = ... }
```

already problematic?

If attackers can observe termination or observe timing then any branching on confidential info is a potential leak

Rules for commands – incl. termination leaks

How do we make these rules save for termination or timing leaks?

if-then-else

$$\frac{e : t \qquad c1 : ok\ t \qquad c2 : ok\ t}{\text{if } e \text{ then } c1 \text{ else } c2 \ : ok\ t}$$

while

$$\frac{e : t \qquad c : ok\ t}{\text{while } e \text{ do } c \ : \ ok\ t}$$

Only allow them for t = L (lowest level of confidentiality)

NB this is extremely restrictive, as you cannot do *any* branching on confidential information

How can we be sure that such type systems are "correct"?

Soundness & Completeness

- soundness of the type system:

  programs that are well-typed do no leak

- completeness of the type system:

  programs that do not leak can be typed

Is the type system on preceding slides

- *sound?*
- *complete?*

How can we determine this?

Counterexamples for completeness

It is easy to give examples that are not typable
but do not leak, eg

- `if (false) then { lo = hi; }`
- `lo = hi + 1 - hi;`
- `lo = hi; lo = 42;`

For the last statement this depends on subtle differences in the attacker model: can the attacker do observations *during execution* or only *at the end of execution* ?

Soundness

- Is this type system sound?
- How do we define what we want to prevent?
  - Recall the tricky examples of implicit flows
- This can be done using notion of non-interference,

Non-interference gives a precise semantics for what "information flow" means, and what attacker can observe

Soundness wrt non-interference

__Definition__ For memories (or program states) μ and ν,
we write $\mu \approx_L \nu$ iff μ and ν agree on low variables.

__Definition__ (Non-interference)

A program C does not leak information if, for all $\mu \approx_L \nu$:
if executing C in μ terminates and results in μ',
and executing C in ν terminates and results in ν',
then $\mu' \approx_L \nu'$

__Theorem__ (Soundness)

if C : ok t then C does not leak information

Termination as covert channel?

Definition (Non-interference) termination-*in*sensitive

A program C does not leak information if, for all $\mu \approx_L \nu$:

if executing C in μ terminates and results in μ',

and executing C in ν terminates and results in ν',

then $\mu' \approx_L \nu'$

Does this rule out (non) termination as hidden channel (as observation to distinguish two runs)?

Definition (Termination-sensitive non-interference)

A program C does not leak information if, for all $\mu \approx_L \nu$:

if executing C in μ terminates in μ',

then executing C in ν also terminates, and results in some ν' with $\mu' \approx_L \nu'$

Other notions of secure information flow

Other definitions of what it means to be secure (in the sense of non-leaking) are needed if

- if programs can throw exceptions
  - exceptions are another covert channel, just like non-termination
- if programs are multi-threaded or non-deterministic
  - because execution of a program can then result in several outcomes
    - multi-threaded programs are non-deterministic, because results can depend on scheduling

The problem with secure information flow

Does `login(String pwd)`
leak confidential info?

Does `String encryt(String s, Key k)`
produce confidential info?

The problem with secure information flow

- *Practical* problem with secure information flow: the extreme restrictions it imposes, esp. when it come to ruling out implicit flows
  -
  - Even if we do not worry about termination or timing leaks

- For most practical applications, we need a looser notion of information flow than non-interference
  - Some controlled form of declassification

Declassification

More *permissive* forms of information flow can allow de-classification, eg

- **for confidentiality:**
  - **output of encryption operation is labelled as public, even though it depends on secret data**
  - **leaking one bit of information about password by login procedure can be - *has* to be - acceptable**

- **for integrity:**
  - **output of input validation routine may be trusted, even though it depends on untrusted data**
  - **output of routine that checks digital signature may be trusted, even though it depends on untrusted data**

Information Flow in practice

- Information flow for integrity – aka tainting – is commonly used in SAST and DAST tools, as discussed last week
  Eg
  - PREfast
  - perl tainting mode
  - most SAST tools such as Fortify, CodeQL or Semmle

- These are often unsound and/or incomplete
  as concession to practicality

Pragmatic approaches typically worry less – if at all – about implicit flows

Indeed, are implicit flows an issue for integrity?

- For confidentialy implicit flows can clearly be dangerous; for integrity this is not so clear.

Summary

- What is information flow (informally)?

  explicit flows , implicit flows, covert channels
- How can we *statically* control information flow, using type systems?
- How can we formally define what information flow is?

  non-interference,

  in termination-sensitive or termination-*in*sensitive variant

You can read all this in Chapter 5 of the lecture notes on Language-Based Security

Possible exam questions

- Explaining if there is unwanted information for integrity or confidentiality in example programs

- Giving and/or motivating a typing rule for information flow for termination-sensitive or insensitive
- Giving and/or explaining the definition of non-interference, for integrity or confidentiality

(but not the possibilistic & probabilistic versions)