Security II - Information Flow Control

Stefano Calzavara

Università Ca' Foscari Venezia

April 10, 2020

Università
Ca'Foscari
Venezia

Introduction

A secret is something that is told to one person at a time...

How do we protect secrets in computer science?

- access control: protect secret data using a password
- encryption: protect secret data using a shared key
- sandboxing: protect secret data by isolating processes

This approach is terribly coarse-grained: once a secret is accessible, we cannot limit its use!

Beyond Access Control

Real-world software often has access to confidential data

- think about all the nice apps running in your mobile phone!
- camouflaged malware might exfiltrate sensitive information
- benign programs might accidentally leak personal data
- how do we reason about the security of such software?

The area of information flow control studies the security of programs manipulating confidential information.

Attacker Model

We assume the set of variables $\mathcal{V}$ is partitioned in two by $\Gamma : \mathcal{V} \rightarrow \{L, H\}$

- $\Gamma(x) = L$ means that x has low confidentiality (public)
- $\Gamma(x) = H$ means that x as high confidentiality (private)

The attacker observes the execution of a program c and tries to derive conclusions on the content of high confidentiality variables by inspecting the content of low confidentiality variables alone.

We assume the attacker has access to the source code of c

Confidentiality?

The following program is clearly insecure:

```
h := read_pin();
l := h;
```

What about this program?

```
h := read_pin();
l := h * 2;
```

Confidentiality?

The following program is clearly insecure:

```
h := read_pin();
l := h;
```

What about this program?

```
h := read_pin();
l := h * 2;
```

The program is insecure, because the attacker can halve the value of l to reconstruct the value of h

Confidentiality?

What about this program?

```
h := read_pin();
if (h > 5000)
  l := 0;
else
  l := 1;
```

Confidentiality?

What about this program?

```
h := read_pin();
if (h > 5000)
  l := 0;
else
  l := 1;
```

The program is insecure, because it suffers from an implicit flow: part of the confidential information is leaked via the control flow

Confidentiality?

What about this program?

```
h := read_pin();
l := 0;
while (h > 0) {
  h := h - 1;
  l := l + 1;
}
```

Confidentiality?

What about this program?

```
h := read_pin();
l := 0;
while (h > 0) {
  h := h - 1;
  l := l + 1;
}
```

The program is insecure, because the content of variable *h* is eventually leaked into variable *l*, again via the control flow

Confidentiality?

What about this program?

```
h := read_pin();
l := 0;
while (h > 5000)
  h := h;
l := 1;
```

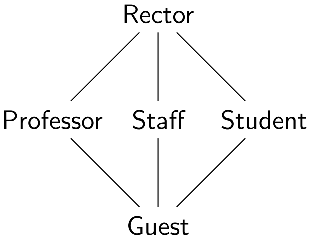
Confidentiality?

What about this program?

```
h := read_pin();
l := 0;
while (h > 5000)
  h := h;
l := 1;
```

It depends! Can the attacker observe termination or not?

Security Lattice



Our ideas can be generalized to arbitrary security lattices $(\mathcal{L}, \sqsubseteq)$:

- lattice = poset with unique least upper bounds $\sqcup$ and greatest lower bounds $\sqcap$
- we assume $\Gamma : \mathcal{V} \rightarrow \mathcal{L}$ and we represent the attacker as some $\ell \in \mathcal{L}$
- for $\ell \in \mathcal{L}$, we let $L = \{\ell' \in \mathcal{L} \mid \ell' \sqsubseteq \ell\}$ and $H = \{\ell' \in \mathcal{L} \mid \ell' \not\sqsubseteq \ell\}$

Non-Interference

We assume the attacker cannot observe termination!

Definition (ℓ-Equivalence)

Two memories μ, μ' are ℓ-equivalent, written $\mu \approx_\ell \mu'$, if and only if $\forall x \in \mathcal{V} : \Gamma(x) \sqsubseteq \ell \Rightarrow \mu(x) = \mu'(x)$.

Definition (Non-Interference)

A program c satisfies non-interference iff, for all labels ℓ and memories μ_1, μ_2 such that $\mu_1 \approx_\ell \mu_2$, we have: if $\langle c, \mu_1 \rangle \Downarrow \mu_1'$ and $\langle c, \mu_2' \rangle \Downarrow \mu_2'$, then $\mu_1' \approx_\ell \mu_2'$.

Non-Interference: Example

```
if (h > 5000)
  l := 0;
else
  l := 1;
```

Before execution:

$$\mu_1 = \{h \mapsto 6789, l \mapsto 0\} \approx_L \mu_2 = \{h \mapsto 1111, l \mapsto 0\}$$

After execution:

$$\mu_1' = \{h \mapsto 6789, l \mapsto 0\} \not\approx_L \mu_2' = \{h \mapsto 1111, l \mapsto 1\}$$

This is a counter-example to non-interference!

Proving Non-Interference

Definition (Non-Interference)

A program c satisfies non-interference iff, for all labels ℓ and memories μ_1, μ_2 such that $\mu_1 \approx_\ell \mu_2$, we have: if $\langle c, \mu_1 \rangle \Downarrow \mu_1'$ and $\langle c, \mu_2' \rangle \Downarrow \mu_2'$, then $\mu_1' \approx_\ell \mu_2'$.

Finding counter-examples is useful, but how can we prove that NI does actually hold?

- key problem: for all memories μ_1, μ_2 (universal quantification)
- shall we do a manual proof for every c we want to show secure?

Security by Typing

Let $pc \in \mathcal{L}$ stand for the program counter label, which is used to track implicit flows. This is raised by conditionals and loops.

Two forms of type rules:

- $\Gamma \vdash e : \ell$ reading as expression e has label ℓ under the typing environment Γ
- $\Gamma, pc \vdash c$ reading as command c is well-typed under the typing environment Γ and the program counter label pc

We do not discriminate between integers and booleans for simplicity.

Typing Rules for Expressions

For expressions, we use rules of the form $\Gamma \vdash e : \ell$

$$\Gamma \vdash v : \ell \qquad \Gamma \vdash x : \Gamma(x) \qquad \frac{\Gamma \vdash e_1 : \ell \qquad \Gamma \vdash e_2 : \ell}{\Gamma \vdash e_1 \oplus e_2 : \ell}$$

$$\frac{\Gamma \vdash e_1 : \ell \qquad \Gamma \vdash e_2 : \ell}{\Gamma \vdash e_1 \leq e_2 : \ell} \qquad \frac{\Gamma \vdash e : \ell \qquad \ell \sqsubseteq \ell'}{\Gamma \vdash e : \ell'}$$

Typing Rules for Commands

For commands, we use rules of the form $\Gamma, pc \vdash c$

$$\Gamma, pc \vdash \textbf{skip} \qquad \frac{\Gamma \vdash e : \ell \qquad \ell \sqcup pc \sqsubseteq \Gamma(x)}{\Gamma, pc \vdash x := e} \qquad \frac{\Gamma, pc \vdash c_1 \qquad \Gamma, pc \vdash c_2}{\Gamma, pc \vdash c_1; c_2}$$

$$\frac{\Gamma \vdash e : \ell \qquad \Gamma, \ell \sqcup pc \vdash c_1 \qquad \Gamma, \ell \sqcup pc \vdash c_2}{\Gamma, pc \vdash \textbf{if } e \textbf{ then } c_1 \textbf{ else } c_2}$$

$$\frac{\Gamma \vdash e : \ell \qquad \Gamma, \ell \sqcup pc \vdash c}{\Gamma, pc \vdash \textbf{while } e \textbf{ do } c} \qquad \frac{\Gamma, pc \vdash c \qquad pc' \sqsubseteq pc}{\Gamma, pc' \vdash c}$$

Typing Example: Safe Assignments

Let $\Gamma = \{h \mapsto H, l \mapsto L\}$

$$\dfrac{\dfrac{\Gamma \vdash l + 4 : L \qquad L \sqsubseteq \Gamma(h)}{\Gamma, L \vdash h := l + 4} \qquad \dfrac{\Gamma \vdash l - 3 : L \qquad L \sqsubseteq \Gamma(l)}{\Gamma, L \vdash l := l - 3}}{\Gamma, L \vdash h := l + 4; l := l - 3}$$

Exercise: complete the rest of the type derivation

Typing Example: Unsafe Assignment

Let $\Gamma = \{h \mapsto H, l \mapsto L\}$

$$\dfrac{\dfrac{\Gamma \vdash h : H \qquad \dfrac{\Gamma \vdash l : L \qquad L \sqsubseteq H}{\Gamma \vdash l : H}}{\Gamma \vdash h + l : H} \qquad H \not\sqsubseteq \Gamma(l)}{\Gamma, L \vdash l := h + l}$$

Notice that we could instead type-check $h := h + l$. Can you show it?

Typing Example: Conditionals

Let $\Gamma = \{h \mapsto H\}$

$$\dfrac{\dfrac{\dots}{\Gamma \vdash h \leq 30 : H} \qquad \dfrac{\Gamma \vdash 5 : L \qquad L \sqcup H \sqsubseteq \Gamma(h)}{\Gamma, H \vdash h := 5} \qquad \Gamma, H \vdash \textbf{skip}}{\Gamma, L \vdash \textbf{if } h \leq 30 \textbf{ then } h := 5 \textbf{ else skip}}$$

Notice that if we replaced the assignment $h := 5$ with $l := 5$ the program would not type-check anymore!

More Examples

Do these programs satisfy NI? Do they type-check or not?

$$\textbf{while } l \leq 34 \textbf{ do } l := l + 1$$

$$\textbf{while } h \leq 34 \textbf{ do } \{l := l + 1; h := h + 1\}$$

$$l := 0; \textbf{while } h \leq 34 \textbf{ do } \{h := h\}; l := 1$$

$$l := h; l := 0$$

$$\textbf{if } h \leq 34 \textbf{ then } l := 0 \textbf{ else } l := 0$$

Exercise: try to type-check these simple examples!

Security Theorem

We can prove the following result:

Theorem

If $\Gamma, pc \vdash c$, then c satisfies non-interference.

In other words, typing is sound. However, we already showed that typing is not complete, i.e., there exist programs which satisfy NI but do not type-check. This is common for type systems.

Example

The program $l := h; l := 0$ is secure, but does not type-check!

Integrity

NI formalizes confidentiality by requiring that high variables (private) do not affect low variables (public).

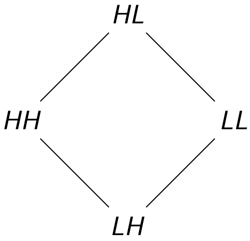
A dual argument holds for integrity, where we can formally require that low variables (tainted) do not affect high variables (trusted).

Example

The following program violates integrity:

$$\textbf{if } l > 0 \textbf{ then } h := 0 \textbf{ else } h := 1$$

Confidentiality + Integrity



We can also combine confidentiality and integrity in the same security lattice:

- confidentiality: $L \sqsubseteq_C H$
- integrity: $H \sqsubseteq_I L$

Moving up in the lattice enforces additional restrictions on the use of data.