Security-typed Languages Lecture 1

Farzaneh Derakhshan, Tao Gu, Aditya Oak

18 June 2019

1 Information Flow: Brief History

It turns out that strong typing and dependent types are not enough for preventing undefined behaviours. *Meltdown attack* is an example of such behaviour:

```
try{
i:=secret          //will generate exception.
a=M[i]             //load address into cache
} catch(illegal read){
for j in 0 ... n {
...                //time access to M[j]} }
```

In the above example, OS read protection is broken and the attacker can read the Kernel. Specs are almost always nondeterministic and the implementation may refine them. Attackers can use this to create correlation between secrets and time and access the secrets. Attack examples such as Meltdown, Spectre, and Foreshadow suggest that we need a more expressive specification of information flow. In particular, we need that that the contents of inaccessible memory do not affect execution time.
In this rest of this section we briefly introduce the history of information flow:

- Lampson, 1971: "A Note on the confinement problem."
  Lampson explored the problem of information leak and discussed that conventional means are not enough to avoid that. They also introduced a category of *covert channels* as those that are not intended for information transfer at all, including the service program's effect on the system load.

- Bell, Lopadula, 1976: "Secure Computer System: Unified exposition and multics interpretation."
  They introduced a computing system with different security levels: public, confidential, secret, and top secret. The rule is that no read up and (more controversially) no write down can happen on the more secret level. For example, information can flow from a public level to the a confidential level, but not the other way around.

- **Denning**, 1976: "A lattice model of secure information flow."
  Denning recognised that mathematical structure of lattices is similar to the relation of policies. Using this observation they built a lattice of policies: For example above policy levels form lattice

  $$\textit{public} \sqsubseteq \textit{confidential} \sqsubseteq \textit{secret} \sqsubseteq \textit{top-secret:}$$

  We can also incorporate the subject of a secret data in lattice

  $$(\textit{secret}, \{\textit{nuclear}\}) \sqsubseteq (\textit{secret}; \{\mathsf{nuclear}; \mathsf{terror}\})$$

- **Denning and Denning**, 1977: "Certification of Programs for Secure Information Flow."
  They used static analysis on programs to verify flow on the respective lattice

- **Biba** : "Integrity considerations for secure computer systems."
  They explained how integrity of information is affected by flow. Trusted information is of higher integrity, so the flow of information is always from trusted information to untrusted information. But, we have *trusted information* $\sqsubseteq$ *untrusted information.* This depicts a duality between trusted and confidentiality.

2 IMP: syntax and rules

In this section, we introduce a language IMP, and some deduction rules which reason about the information flow in IMP programs.

2.1 Syntax

$$\begin{aligned}
\mathsf{AExp}\ a &::= x \mid n \mid a_1 \oplus a_2 \\
\mathsf{BExp}\ b &::= \mathsf{true} \mid \mathsf{false} \mid b_1 \owedge b_2 \mid a_1 \oominus a_2 \\
\mathsf{Command}\ c &::= \mathsf{skip} \mid x := a \mid c_1; c_2 \mid \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 \mid \mathsf{while}\ b\ \mathsf{do}\ c
\end{aligned}$$

Here $x \in \mathsf{Var}$, $n \in \mathbb{N}$, $\owedge$. AExp, BExp and Command are respectively arithmetic expressions, boolean expressions, and commands. As for the operators, $\owedge$ stands for conjunction, $\oominus$ stands for equivalence, $x := a$ is value assignment.

Let $\mathcal{L}$ be a lattice with a specific bottom element $\bot$. The join and meet in $\mathcal{L}$ are $\sqcup$ and $\sqcap$. Relation $\sqsubseteq$ is the normal partial order on lattice, defined as $\ell_1 \sqsubseteq \ell_2 \iff \ell_1 \sqcup \ell_2 = \ell_2$. If $\ell_1 \sqsubseteq \ell_2$, then we say "$\ell_1$ is lower than ℓ_2" or "ℓ_2 is higher than ℓ_1". Elements in $\mathcal{L}$ are called labels.

Definition 1. The *judgements* are of the following three forms:

$$\vdash a : \ell \qquad \vdash b : \ell \qquad \mathsf{pc} \vdash c$$

where $a \in \mathsf{AExp}$, $b \in \mathsf{BExp}$, $c \in \mathsf{Command}$, and $\mathsf{pc}, \ell \in \mathcal{L}$ (pc stands for "program counter").

Intuitively, labels in $\mathcal{L}$ can be seen as "security levels" or "information", and the bottom element $\bot$ simply means "public", or "least information". Then $\vdash a : \ell$ can be read either as "term a has security level ℓ" or as "term a requires information in ℓ", and $\mathsf{pc} \vdash c$ can be read as either "the information in pc can flow into command c" or "to execute command c one needs at least information in pc". The function $\Gamma : \mathsf{Var} \to \mathcal{L}$ assigns to each variable x a label $\Gamma(x)$, which is the security level of variable x.

Example 2. The following expressions are all judgements:

$$\vdash n : \bot \quad \vdash \mathsf{true} : \bot \quad \vdash x : \Gamma(x)$$

2.2 Deduction rules

The rules for IMP are as follows:

$$\frac{\vdash a_1 : \ell_1 \qquad \vdash a_2 : \ell_2}{\vdash a_1 \oplus a_2 : \ell_1 \sqcup \ell_2} \qquad \frac{\mathsf{pc} \vdash c_1 \qquad \mathsf{pc} \vdash c_2}{\mathsf{pc} \vdash c_1; c_2}$$

$$\frac{\mathsf{pc} \vdash c_1 \qquad \mathsf{pc} \vdash c_2}{\mathsf{pc} \vdash c_1; c_2} \qquad \frac{}{\mathsf{pc} \vdash \mathsf{skip}}$$

$$\frac{\vdash a : \ell \qquad \ell \sqsubseteq \Gamma(x) \qquad \mathsf{pc} \sqsubseteq \Gamma(x)}{\mathsf{pc} \vdash x := a}$$

$$\frac{\vdash b : \ell \qquad \ell \sqcup \mathsf{pc} \vdash c_1 \qquad \ell \sqcup \mathsf{pc} \vdash c_1}{\mathsf{pc} \vdash \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2}$$

$$\frac{\vdash b : \ell \qquad \ell \sqcup \mathsf{pc} \vdash c}{\mathsf{pc} \vdash \mathsf{while}\ b\ \mathsf{do}\ c}$$

According to our intuition of judgements above, these rules should have a quite straightforward reading. Take the rule for value assignment as an example. One can read it as, if the information level of term a is ℓ, and both ℓ and pc are lower than the information level of variable x, then the information at level pc flows into the assignment command $x := a$.

Example 3. Let pub and secret respectively have the lowest and highest security level, say $\bot$ and $\top$. Consider the following program (in pseudocode): As a result, we have $\mathsf{pub} = \mathsf{secret}$, which means that we can know the value of secret from that of pub. This destroys the security order of pub and secret: intuitively the high-level information should not affect low-level information. The above language IMP provides a way to formalize this intuition:

$$\top \not\vdash \mathsf{pub} := \mathsf{true}$$

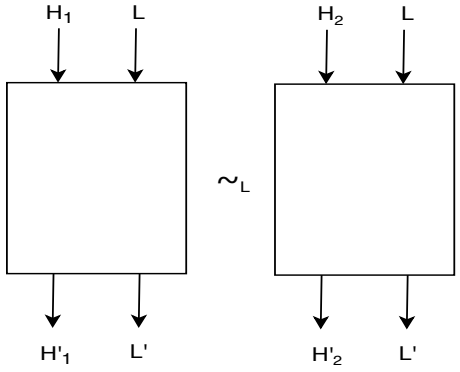
which means that "the information at level $\top$ does emphnot flow into pub".

Algorithm 1 Test if secret is true or false

```
1: pub = false
2: if secret then
3:     pub = true
4: Return pub
```

3 Noninterference

In 1982, Goguen and Meseguer proposed noninterfernce model. A system has high (secret) and low (public) inputs and outputs. Such a system is said to have noninterference property if given the same low inputs, system always produces same low outputs irrespective of the changes in the high inputs.



When high inputs are changed, behaviour of the system remains indistinguishable for an observer who can only observe the low outputs. We formalize this notion of indistinguishability. Let s be the state of a system. $s_1 \sim_L s_2$ denotes two states s_1 and s_2 that are indistinguishable from the perspective of a low observer. Let $[\![S]\!]$ be the observable behaviour when executing from s. Indistinguishablity of the two behaviours is denoted by $[\![S_1]\!] \approx_L [\![S_2]\!]$.

- TINI (termination insensitive noninterference) : Program may terminate on some high inputs and may not on others.
- TSNI (termination sensitive noninterference) : high inputs do not decide whether a program terminates or not. TSNI is relatively harder to enforce.

  TINI + termination $\Rightarrow$ TSNI

Let state s be composed of two parts s_H (high) and s_L (low), then indistinguishability of two states s_1 and s_2 is given by

$$s_1 \sim_L s_2 \iff (s_{1_L} = s_{2_L})$$

4 Real Systems

- SPARK Ada : Based on [Denning:1977]
- Jif / JFlow [myers1999j ow]: Java + Information flow
- Flow Caml [1]
- Laminar [Roy:2009]
- Paragon [broberg2013paragon]
- JOANA [2]
- IFC for web applications
- SecVerilog [3] : Verilog + Information Flow

[1] https://www.normalesup.org/~simonet/soft/flowcaml
[2] https://pp.ipd.kit.edu/projects/joana
[3] http://www.cs.cornell.edu/projects/secverilog