CIS 301: Lecture Notes on Logic for Security

Torben Amtoft
Department of Computing and Information Sciences
Kansas State University

December 7, 2003

These notes are inspired by [1].

1 Secure Information Flow

Assume we are dealing with two kinds of variables: those of high security (classified); and those of low security (non-classified). Our goal is that users with low clearance should not be able to gain information about the values of the classified variables. In the following, this notion will be made precise.

For the sake of simplicity, let us assume that there are only two variables in play: `l` (for low) and `h` (for high). We want to protect ourselves against an attacker (spy) who

- knows the initial value of `l`;
- knows the program that is running;
- can observe the final value of `l`;
- can *not* observe intermediate states of program execution.

A program is said to be *secure* if such an attacker cannot detect anything about the initial value of `h`.

1.1 Examples

The program below is *not* secure.

$$\texttt{l} := \texttt{h} + 7 \tag{1}$$

For by subtracting 7 from the final value of `l`, the attacker gets the initial value of `h`. On the other hand, the program below is clearly secure.

$$\texttt{l} := \texttt{l} + 47 \tag{2}$$

One rotten apple does not always spoil the whole barrel; having the insecure program in (1) as a preamble may still yield a secure program as in

$$\mathtt{l} := \mathtt{h} + 7;\ \mathtt{l} := 27 \tag{3}$$

since we assumed that the attacker cannot observe intermediate values of `l`. Also the following program is secure:

$$\mathtt{h} := \mathtt{l} \tag{4}$$

For even though the attacker learns the *final* value of `h` (as it equals the initial value of `l` which is known), he is still clueless about the *initial* value of `h`.

The following program is just a fancy way of writing $\mathtt{l} := \mathtt{h} + 7$ (since we do not care about the final value of `h`)

$$\mathtt{l} := 7;\ \mathtt{while}\ \mathtt{h} > 0\ \mathtt{do}\ \mathtt{h} := \mathtt{h} - 1;\ \mathtt{l} := \mathtt{l} + 1\ \mathtt{od} \tag{5}$$

and is therefore insecure. Also, the following program is insecure

$$\mathtt{if}\ \mathtt{h} = 6789\ \mathtt{then}\ \mathtt{l} := 0\ \mathtt{else}\ \mathtt{l} := 1\ \mathtt{fi} \tag{6}$$

since if the final value of `l` is zero, we know that `h` was initially 6789.

1.2 Specification

By putting quantifiers in front of Hoare triples, we can formally express security:

Definition: The program P is secure iff

$$\begin{array}{l} \forall l_0\ \exists l_1\ \forall h_0\ \exists h_1 \\ \qquad \{\mathtt{l} = l_0 \land \mathtt{h} = h_0\} \\ \qquad\qquad P \\ \qquad \{\mathtt{l} = l_1 \land \mathtt{h} = h_1\} \end{array}$$

To put it another way, the final value (l_1) of `l` must depend only on the initial value (l_0) of `l` and *not* on the initial value (h_0) of `h`.

By negating this definition (and applying De Morgan's laws repeatedly), we arrive at:

Observation: The program P is insecure iff

$$\begin{array}{l} \exists l_0\ \forall l_1\ \exists h_0\ \neg\exists h_1 \\ \qquad \{\mathtt{l} = l_0 \land \mathtt{h} = h_0\} \\ \qquad\qquad P \\ \qquad \{\mathtt{l} = l_1 \land \mathtt{h} = h_1\} \end{array}$$

To put it another way, a program is insecure if for all possible final values of `l`, there exists an initial value of `h` that produces a different final value of `l`.

1.3 Examples Revisited

We first address the programs that are secure, and show that they do indeed meet the requirement stated in our Definition. In each case, we are given some l_0 and must find l_1 such that

$$\forall h_0\ \exists h_1 \quad \begin{array}{l} \{\mathtt{l} = l_0 \land \mathtt{h} = h_0\} \\ \qquad P \\ \{\mathtt{l} = l_1 \land \mathtt{h} = h_1\} \end{array}$$

For the program in (2), we choose l_1 as $l_0 + 47$; this does the job since

$$\forall h_0\ \exists h_1 \quad \begin{array}{l} \{\mathtt{l} = l_0 \land \mathtt{h} = h_0\} \\ \qquad \mathtt{l} := \mathtt{l} + 47 \\ \{\mathtt{l} = l_0 + 47 \land \mathtt{h} = h_1\} \end{array}$$

For the program in (3), we can choose l_1 as 27; for the program in (4), we simply choose l_1 as l_0.

We next address the programs that are *not* secure, and show (cf. our Observation) that no matter how l_1 has been chosen, we can find h_0 such that it does *not* hold that

$$\begin{array}{l} \{\mathtt{l} = l_0 \land \mathtt{h} = h_0\} \\ \qquad P \\ \{\mathtt{l} = l_1\} \end{array}$$

For the programs in (1) and (5), we can just pick an h_0 different from $l_1 - 7$, say $h_0 = l_1$. For clearly it does not hold that

$$\begin{array}{l} \{\mathtt{l} = l_0 \land \mathtt{h} = l_1\} \\ \qquad \mathtt{l} := \mathtt{h} + 7 \\ \{\mathtt{l} = l_1\} \end{array}$$

For the program in (6), we proceed by cases on l_1: if l_1 is zero, then we can choose (among many possibilities) h_0 to be 2345 since it does not hold that

$$\begin{array}{l} \{\mathtt{l} = l_0 \land \mathtt{h} = 2345\} \\ \qquad \mathtt{if}\ \mathtt{h} = 6789\ \mathtt{then}\ \mathtt{l} := 0\ \mathtt{else}\ \mathtt{l} := 1\ \mathtt{fi} \\ \{\mathtt{l} = 0\} \end{array}$$

Alternatively, if l_1 is one, then we choose h_0 to be 6789 since it does not hold that

$$\begin{array}{l} \{\mathtt{l} = l_0 \land \mathtt{h} = 6789\} \\ \qquad \mathtt{if}\ \mathtt{h} = 6789\ \mathtt{then}\ \mathtt{l} := 0\ \mathtt{else}\ \mathtt{l} := 1\ \mathtt{fi} \\ \{\mathtt{l} = 1\} \end{array}$$

(If l_1 is neither zero nor one, we can choose any value for h_0.)

1.4 Declassification

A severe limitation of our theory is exposed by the last example (6) which is considered insecure even though very little information may actually be leaked to the attacker. Think of `h` as denoting a PIN code, with the attacker testing whether it happens to be 6789; if the PIN codes were selected randomly, the chance of the test revealing the PIN code is very small (1 to 10,000). It is currently an important challenge for research in (language based) security to formalize these considerations!

1.5 Data Integrity

We might consider an alternative interpretation of the variables `l` and `h`: `l` denotes a licensed entity, whereas `h` denotes a hacked (untrustworthy) entity. The integrity requirement is now:

> Licensed data should *not* depend on hacked data.

It is interesting to notice that the framework described on the preceding pages covers also that situation! In particular, a program satisfies the above integrity requirement if and only if it is considered secure (according to our Definition). For example, (3) is safe as the licensed variable `l` will eventually contain 27 which does not depend on hacked data, whereas (6) is unsafe as the value of the hacked variable `h` influences the value of the licensed variable.

References

[1] Ádám Darvas, Reiner Hähnle, and David Sands. A theorem proving approach to analysis of secure information flow. In *Workshop on Issues in the Theory of Security (WITS'03). Affiliated to ETAPS 2003, Warsaw, Poland.*, 2003.