Trustworthiness

Willard Rafnsson

IT University of Copenhagen

In this assignment, we consider trustworthiness: how to convince ourselves and others (e.g. the consumer) that the software we write is secure. The goal is to gain better understanding of trustworthiness, to gain the ability to assess trustworthiness, and to gain first-hand experience with writing trustworthy software using information-flow control.

Problem 1 : Basis of Trust

We see how introducing assurance affects trust assumptions. Take-away: the assumptions never go away; they change & relocate to something we may be more comfortable with.

A *trust triple* is a representation of a trust assumption. It has the form $\langle \mathsf{A}, \mathsf{B}, \mathsf{W} \rangle$, where A denotes an artifact (e.g. code, a component, a tool), B a basis of trust (i.e. axiomatic, analytical, or synthesized), and W is either an entity or an artifact (ax: faith in what? an: which analysis? sy: what means of construction?). Example: You write a program S, compile it to binary B, and run B. Then $\langle S, \text{an}, \text{you} \rangle$, $\langle B, \text{sy}, S \rangle$, $\langle B, \text{sy}, \text{compiler} \rangle$, and $\langle \text{compiler}, \text{ax}, \text{compiler writer} \rangle$ are the trust assumptions; you carefully wrote S, B is built from S and compiler, and you blindly trust whoever wrote the compiler. We use trust triples to discover Ws we didn't know we were making trust assumptions about. We can go deeper (operating system, hardware, runtime, etc.); this is required for some assurance levels; however, presently we will stick to the "application level".

In the following, we disregard trust assumptions concerning authentication, exchange of data, accountability, and unmentioned parts of the SW/HW stack. In the **b)** parts, mention how the trust assumptions have changed from a previous scenario.

You are a software consumer. You purchase a binary B from Amasoft Inc., and run it.

Part 1 **a)** Fill in $\langle B, \_, \_ \rangle$. **b)** Why is this trust assumption (un)reasonable? (~1 sentences)

Suppose you also received a test suite T. Before you run B, you read T yourself to check input coverage & tested properties, and run T to confirm that B passes all tests.

Part 2 **a)** $\langle B, \_, \_ \rangle$, $\langle T, \_, \_ \rangle$. **b)** Why are these assumption (un)reasonable? (~1 sentences)

Suppose that instead of T, you receive a proof object P. You read P yourself, and check that P proves correctness of B using a third-party verifier program V.

Part 3 **a)** Propose trust triples. **b)** Why are these assumption (un)reasonable? (~1 sentences)

Suppose you read V, to verify that V is correct.

Part 4 **a)** Propose trust triples. **b)** Why are these assumption (un)reasonable? (~1 sentences)

Suppose, instead of relying on a proof, you run B'—the result of rewriting B using a third-party program rewriter R (e.g. a monitor).

Part 5 **a)** Propose trust triples. **b)** Why are these assumption (un)reasonable? (~1 sentences)

You are now instead managing a team at AmaSoft, Inc., that you task with writing a module with security goals, for a larger software product. Amy & Bob are on your team.

Amy writes a crypto module M (using standard library only). You compile the module to binary B, and intend to use B.

Part 6 **a)** Propose trust triples. **b)** Why are these assumption (un)reasonable? (~1 sentences)

You ask Bob to review M; the code passes Bob's review.

Part 7 **a)** Propose trust triples. **b)** Why are these assumption (un)reasonable? (~1 sentences)

You ask Amy & Bob to write a test suite T. Before you use B, you read T yourself to check input coverage & tested properties, and run T to confirm that B passes all tests.

Part 8 **a)** Propose trust triples. **b)** Why are these assumption (un)reasonable? (~1 sentences)

You ask Amy to instead use `commons-crypto`, an open-source crypto-library. You read M to confirm that M indeed is just a thin wrapper around the library.

Part 9 **a)** Propose trust triples. **b)** Why are these assumption (un)reasonable? (~1 sentences)

Amy re-implements M in Paragon. You check the security policy embedded in M, and confirm that the Paragon compiler successfully compiles M.

Part 10 **a)** Propose trust triples. **b)** Why are these assumption (un)reasonable? (~1 sentences)

Problem 2 : IFC

We see the various nuances in ensuring that only permitted flows of information occur in a program. We consider the same simplified setting as in class, i.e. `Imp`.

An `Imp` program takes a memory as input, and produces a memory as output. Each variable is labeled by a level. Levels form a partial order, expressing which information flows are permitted. Two memories are ℓ-equivalent if they agree on all variables labeled $\sqsubseteq \ell$. A program is *not* information-flow secure, if, for some ℓ and two ℓ-equivalent input memories, the program terminates with two ℓ-different output memories.

Thus, if a program is secure, then a principal who observes final values of variables labeled $\sqsubseteq \ell$, learns nothing about initial values of variables labeled $\not\sqsubseteq \ell$.

We consider a partial order with only two levels, H (high) and L (low), with $\mathsf{L} \sqsubseteq \mathsf{H}$. We will consider the following two programs.

```
1 female := 0;
2 if (cpr % 2 == 0) {
3   female := 1;
4 }
5 female := 1;
```

```
1 pub := pub + 5;
2 while (sec % 2 == 0) {
3   sec = sec * 2;
4 }
```

First, consider the program on the left. Suppose `female` is labeled L and `cpr` labeled H.

Part 1 This program is information-flow secure. Explain why. (~ 2 sentences)

Part 2 Does `check` accept this program? Justify your answer. (~ 2 sentences)

Part 3 What happens if we run this program `monitored` with `cpr = 2106905060`? (~ 3 sentences)

Part 4 What happens if we run this program `monitored` with `cpr = 2106905065`? (~ 3 sentences)

Part 5 What *would* an L-observer have to observe (besides input and output memories), for this program to *not* be information-flow secure? (~ 1 sentences)

We now consider the program on the right. Suppose `pub` is labeled L and `sec` labeled H.

Part 6 This program is information-flow secure. Explain why. (~ 2 sentences)

Part 7 Does `check` accept this program? Justify your answer. (~ 2 sentences)

Part 8 What happens if we run this program `monitored`? (~ 2 sentences)

Part 9 What *would* an L-observer have to observe (besides input and output memories), for this program to *not* be information-flow secure? (~ 1 sentences)

Hint: It's a side channel.

Part 10 How much does this program leak in that case? (~ 1 sentences)

Hint: At most, per execution.

Problem 3: IFC for App-Specific Security Goals

We see how information-flow control can enforce application-specific security goals.

Static Policies

Accompanying this assignment, you will find a code skeleton that implements PrettyBook: a pretty useless social network. PrettyBook operates on information that concerns three principals: `Amy`, `Bob`, and `You`. `Amy` and `Bob` are `You`r friends. `You` would like to share messages that you type with your friends.

Part 1 How must the principals `Amy`, `Bob` and `You` be ordered for messages from `You` to be permitted to flow to `Amy` and `Bob`, respectively? Draw the partial order.

Part 2 How would you implement the above-stated ordering in the provided implementation of PrettyBook? Only the lines specified (with comments) need to be modified. Provide the updated lines in your solution.

Dynamic Policies

Accompanying this assignment, you find a modified version of PrettyBook, that now has extended menu options. You'd like to share messages that you type with your friends, only while they are permitted to receive your message, as dictated by a *flow lock*.

Part 3 How would you modify the level for `Amy`, `Bob` and `You`, to achieve this effect?

Part 4 How would you modify the implementation of PrettyBook, to achieve this effect?

Hint: As an aid, we have marked the sections in the code where changes are needed.