Software Security - Trace Properties, Self-Composition and Symbolic Execution

José Fragoso Santos, Assistant Professor @ DEI, IST

December 2, 2019

1 Trace Properties

Question 1.1. Recall the approach for defining trace properties introduced in the lecture. For example, the property `ZeroX`, describing all traces terminating with a state mapping the program variable `x` to 0, is *formally defined* below:

$$\begin{aligned}\texttt{ZeroX} \triangleq \{[\langle\rho_0, s_0\rangle, ..., \langle\rho_n, s_n\rangle] \mid s_n = \mathsf{skip} \ \wedge \ \rho_n(\texttt{x}) = 0 \\ \wedge\ \forall_{0 \leq i < n} \langle\rho_i, s_i\rangle \rightarrow \langle\rho_{i+1}, s_{i+1}\rangle\}\end{aligned}$$

Give the formal definition of the following trace properties:

1. `AllZero`, describing the traces that terminate with a state mapping all of its variables to zero;
2. `MonX`, describing all traces such that the value of the program variable `x` never gets decremented;
3. `AllMon`, describing all traces such that the values of all program variables never get decremented;
4. `XGreaterThanY`, describing all traces such that, whenever both variables `x` and `y` are defined, `x` is greater than `y`;
5. `XGreaterThanAll`, describing all traces such that, whenever `x` is defined, it is greater than all other program variables;
6. `YBookKeepX`, describing all traces such that the variable `x` is only allowed to swap sign once, and, if it does, the variable `y` will eventually be set to the old value of `x` and will keep that value until the execution finishes.

Which of the properties above are *liveness properties* and which are *safety properties*? Justify your answer.

Question 1.2. Recall the syntax of `WHILE` programs introduced in the lecture:

$e_1, e_2 \in \mathcal{E} \triangleq \ n \mid \texttt{x} \mid \ominus\ e_1 \mid e_1 \oplus e_2$

$s_1, s_2 \in \mathcal{S} \triangleq \ \mathsf{skip} \mid \texttt{x} := e \mid s_1; s_2 \mid \mathsf{if}\ (e)\ \{\ s_1\ \}\ \mathsf{else}\ \{\ s_2\ \} \mid \mathsf{while}\,(e)\{s_1\}$

For each property in Question 1.1, write a `While` statement that satisfies the property and a `While` statement that does **not**. For the statement that does not, give a program trace that exhibits the *bug*.

2 Self-Composition and Symbolic Execution

Question 2.1. Consider the following `WHILE` statements:

1. $\mathsf{if}\ (\mathtt{h})\ \{\ \mathtt{l} := \mathtt{l} + \mathtt{z}\ \}\ \mathsf{else}\ \{\ \mathsf{skip}\ \}$

2. $\mathsf{if}\ (\mathtt{h})\ \{\ \mathtt{x} := \mathtt{x} + \mathtt{z}\ \}\ \mathsf{else}\ \{\ \mathsf{skip}\ \};$
   $\mathsf{if}\ (!\mathtt{h})\ \{\ \mathtt{y} := \mathtt{y} + \mathtt{z}\ \}\ \mathsf{else}\ \{\ \mathsf{skip}\ \};$
   $\mathtt{x} := 0;\ \mathtt{y} := 0;\ \mathtt{l} := \mathtt{x} + \mathtt{y}$

3. $\mathtt{y} := 1;$
   $\mathsf{while}\,(\mathtt{x} > 0)\ \{$
   $\quad \mathsf{if}\ (\mathtt{y} > \mathtt{h})\ \{\ \mathsf{skip}\ \}\ \mathsf{else}\ \{\ \mathtt{y} := \mathtt{y} * \mathtt{x}\ \};$
   $\quad \mathtt{x} := \mathtt{x} - 1$
   $\}$

Use self-composition and symbolic execution to check which of the above programs satisfy non-interference. Assume the standard lattice $\mathcal{L} = \langle \{L, H\}, \sqsubseteq \rangle$ and a security labelling, Γ, such that only the program variable $\mathtt{h}$ is mapped to the security level H.

Question 2.2. Which of the programs above would be considered secure by a standard type system for information flow control? Which would be considered insecure? What can be concluded about the precision of self-composition + symbolic execution when compared to type systems for information flow control?