Static analysis of high-level languages – Exercises (with solutions)

Ana Matos
Software Security 2020/2021, IST

November 16, 2020

Aim To become familiar with basic notations and concepts for defining and understanding a type system. To analyse the limitations and guarantees offered by a static type-based enforcement mechanism.

To gain first-hand experience on how the limitations of a static type-based analysis can be exploited into attacks. To observe how different language constructs can create dependencies between information, and how a type system can restrict information flows originated by them.

1 Type checking programs

Group 3 is marked with an * as it is intended to be solved with guidance.

1. The following type system was given in class for the language WHILE (syntax and semantics are as usual).

   - Γ maps variables $x, y \ldots$ to security levels τ.
   - Syntax of types for expressions are τ, and types for statements are τ cmd.
   - $(\{L,H\}, \leq)$ forms a High-Low security lattice, where $\wedge$ gives the greatest lower bound between two security levels.
   - Typing judgments for expressions e are of the form $\Gamma \vdash e : \tau$, and for statements S are of the form $\Gamma \vdash S : \tau$ cmd.
   - The type given to expressions is an upper bound to the levels of the variables that appear in it.

$$\frac{}{\Gamma \vdash \text{skip} : \top \text{ cmd}}$$

$$\frac{\Gamma(x) = \tau \quad \Gamma \vdash e : \tau' \quad \tau' \leq \tau}{\Gamma \vdash x := e : \tau \text{ cmd}}$$

$$\frac{\Gamma \vdash e : \tau \quad \Gamma \vdash S_1 : \tau_1 \text{ cmd} \quad \Gamma \vdash S_2 : \tau_2 \text{ cmd} \quad \tau \leq \tau_1, \tau_2}{\Gamma \vdash \text{if } e \text{ then } S_1 \text{ else } S_2 : \tau_1 \wedge \tau_2 \text{ cmd}}$$

$$\frac{\Gamma \vdash e : \tau' \quad \Gamma \vdash S : \tau \text{ cmd} \quad \tau' \leq \tau}{\Gamma \vdash \text{while } e \text{ do } S : \tau \text{ cmd}}$$

$$\frac{\Gamma \vdash S_1 : \tau_1 \text{ cmd} \quad \Gamma \vdash S_2 : \tau_2 \text{ cmd}}{\Gamma \vdash S_1; S_2 : \tau_1 \wedge \tau_2 \text{ cmd}}$$

(a) Are the following programs typable with respect to the given type system? Justify your answer by either giving a type derivation, or showing why that is not possible.

   i. $(\text{if } y_H \text{ then } (\text{while true do skip}) \text{ else skip}); x_L := 1$
   ii. $(\text{if } y_H \text{ then } (\text{while true do skip}) \text{ else } x_L := 1); x_L := 1$
   iii. $\text{if } y_H \text{ then skip else } (\text{if } x_L \text{ then } z_L := 1 \text{ else } z_L := 0)$

(b) Give an example of a context where the above program 1(a)ii should be considered insecure and justify your answer.

(c) Is the above type system complete with respect to Input-Output Deterministic Noninterference? Justify your answer.

(d) The above type system is sound with respect to Input-Output Deterministic Noninterference. Explain rigorously what this means.

(e) Give an example of a program that is not typable in the above type system, but that is secure with respect to Input-Output Deterministic Noninterference.

Hints on understanding the above type system:

- The type of each statement is derived from the types of its components, except for the type of the assignment, which is the security level of the variable that is assigned. Each statement is given a security level that represents the greatest lower bound to the security levels of the variables that are assigned to in the program.
- The flow restrictions of the form $\tau_1 \leq \tau_2$ restrict the way that expressions and statements can be composed into larger statements. In practice, they prevent assignments of high expressions to low variables, and low assignments from occuring under high guards.

2. The following type system for the language WHILE (syntax and semantics are as usual) is sound with respect to Deterministic Input-Output Noninterference.

   - Γ maps variables $x, y \ldots$ to security levels τ.
   - Syntax of types for expressions are τ.
   - $(\{L, H\}, \leq)$ forms a High-Low security lattice, where $\vee$ gives the least upper bound between two security levels.
   - Typing judgments for expressions e are of the form $\Gamma \vdash e : \tau$, and for statements S are of the form $\Gamma; \tau \vdash S$.
   - The type given to expressions is an upper bound to the levels of the variables that appear in it.

$$\frac{}{\Gamma; pc \vdash \text{skip}}$$

$$\frac{\Gamma \vdash e : \tau \quad \tau \leq \Gamma(x) \quad pc \leq \Gamma(x)}{\Gamma; pc \vdash x := e}$$

$$\frac{\Gamma \vdash e : \tau \quad \Gamma; \tau \vee pc \vdash S_1 \quad \Gamma; \tau \vee pc \vdash S_2}{\Gamma; pc \vdash \text{if } e \text{ then } S_1 \text{ else } S_2}$$

$$\frac{\Gamma \vdash e : \tau \quad \Gamma; \tau \vee pc \vdash S}{\Gamma; pc \vdash \text{while } e \text{ do } S}$$

$$\frac{\Gamma; pc \vdash S_1 \quad \Gamma; pc \vdash S_2}{\Gamma; pc \vdash S_1; S_2}$$

   (a) Are the following programs typable with respect to the given type system? Justify your answer by either giving a type derivation, or showing why that is not possible.

      i. if y_H then skip else (if x_L then $z_L := 1$ else $z_L := 0$)
      ii. (if y_H then (while true do skip) else skip); $x_L := 1$
      iii. if y_H then skip else (if x_L then $z_L := 1$ else $z_L := 0$)

   (b) Give an example of a program that is not typable in the above type system, but that is secure with respect to Input-Output Deterministic Noninterference.

Hints on understanding the above type system:

- Statements are only typable if they respect the flow restrictions.
- The level pc to the left of the $\vdash$ in the typing judgments represents the level of the program counter, i.e., of the information that the control flow currently might depend on. More concretely, it is an upper bound on the levels of the variables that are tested by conditionals and loops.
- Assignments under a program counter level pc in practice carry pc level of read information. Typability onder a higher pc is more restrictive, as the restriction on the assignment becomes harder to fullfil.

3. (*) Consider the following type system for the language WHILE (syntax and semantics are as usual).

- Γ maps variables $x, y \ldots$ to security levels $\tau, \sigma, \delta \in \{L, H\}$.
- Syntax of types for expressions are τ, and types for statements are (τ, σ) cmd.
- $(\{L,H\}, \leq)$ forms a High-Low security lattice, where $\wedge$ gives the greatest lower bound between two security levels and $\vee$ gives the least upper bound between two security levels.
- Typing judgments for expressions e are of the form $\Gamma \vdash e : \tau$, and for statements S are of the form $\Gamma \vdash S : (\theta, \sigma)$ cmd.
- The type given to expressions is an upper bound to the levels of the variables that appear in it.

$$\frac{}{\Gamma \vdash \text{skip} : (H, L) \text{ cmd}}$$

$$\frac{\Gamma(x) = \theta \quad \Gamma \vdash e : \delta \quad \delta \leq \theta}{\Gamma \vdash x_\theta := e : (\theta, L) \text{ cmd}}$$

$$\frac{\Gamma \vdash e : \delta \quad \Gamma \vdash S_1 : (\theta_1, \sigma) \text{ cmd} \quad \Gamma \vdash S_2 : (\theta_2, \sigma) \text{ cmd} \quad \delta \leq \theta_1, \theta_2}{\Gamma \vdash \text{if } e \text{ then } S_1 \text{ else } S_2 : (\theta_1 \wedge \theta_2, \delta \vee \sigma) \text{ cmd}}$$

$$\frac{\Gamma \vdash e : \delta \quad \Gamma \vdash S : (\theta, \sigma) \text{ cmd} \quad \delta \vee \sigma \leq \theta}{\Gamma \vdash \text{while } e \text{ do } S : (\theta, \delta \vee \sigma) \text{ cmd}}$$

$$\frac{\Gamma \vdash S_1 : (\theta_1, \sigma_1) \text{ cmd} \quad \Gamma \vdash S_2 : (\theta_2, \sigma_2) \text{ cmd} \quad \sigma_1 \leq \theta_2}{\Gamma \vdash S_1; S_2 : (\theta_1 \wedge \theta_2), (\sigma_1 \vee \sigma_2) \text{ cmd}}$$

(a) Write a program that is accepted by the type system in the previous question, but that is rejected by the above type system. Can you generalize it into a class of programs?

(b) In which contexts would it make sense to reject programs such as the ones mentioned in the previous question?

Hints on understanding the above type system:

- The type of each statement is derived from the types of its components, except for the type of the assignment, which is the security leve of the variable that is assigned. Each statement is given two security levels: one that represents the greatest lower bound to the security levels of the variables that are assigned to in the program, and another that represents the least upper bound to the security levels of the variables that are tested by conditionals and loops.
- The flow restrictions of the form $\tau_1 \leq \tau_2$ restrict the way that expressions and statements can be composed into larger statements. In practice, besides preventing assignments of high expressions to low variables and low assignments from occuring under high guards, they also prevent low assignments from occuring *after* high guards. This stronger restriction prevents timing leaks.

2 Exploiting limitations of type systems

1. Take the Information Flow Challenge at `http://ifc-challenge.appspot.com/steps/start`.

   (a) For each of the steps, state what are the information flow channels that you can exploit in order to win the game.

   (b) Why is it that for steps 3 and 4, the game is different (the secret is fixed)?

   (c) Which of the type systems are sound with respect to Deterministic Input-Output Noninterference?

   (d) For the type systems that are not sound with respect to Deterministic Input-Output Noninterference, can you suggest ways of fixing them?