Instructors: Matt Fredrikson, Jean Yang TA: Samuel Yeom

Due date: April 6 at 11:59pm

Assignment 4

Last updated on April 2

This assignment will show you how to apply the what you learned about information flow type systems to the server you implemented in previous assignments. First, you will provide the operational semantics and information flow typing rules for the scripting language from Assignment 1, and prove the soundness of your type system. You will then implement this type system as a type checker for your server. Finally, you will compare this type checker to the reference monitor you worked with for Assignment 2.

Recall the scripting language you implemented in Assignments 1 and 2. In this system, different principals can modify values on the system. We want to prevent unintended leaks of information between values that have been updated by different principals, and we will use information flow types to do so.

For the purpose of modeling our system, we make the following simplifying modifications:

1. Instead of allowing the program to create principals, we assume the system has a fixed set of principals p.
2. We only care about `read` and `write` permissions, and not `append` or delegate. Appending now requires `write` permissions, and only admin can set or delete delegations. As before, admin has read and write permissions on all variables, and the principal that creates a variable has read and write permissions on that variable.
3. For the non-extra credit parts of the assignment, we will ignore records.

Below is a subset of the language:

⟨**prog**⟩	::=	`as principal` p `password` s `do \n` ⟨**cmd**⟩ $***$
⟨**cmd**⟩	::=	`exit \n` | `return` ⟨**expr**⟩ `\n` | ⟨**prim_cmd**⟩ `\n` ⟨**cmd**⟩
⟨**prim_cmd**⟩	::=	`create principal` p s
	|	`set` x `=` ⟨**expr**⟩
	|	`append to` x `with` ⟨**expr**⟩
	|	`filtereach` y `in` x `with` ⟨**expr**⟩
	|	`set delegation` x ⟨**right**⟩ `->` p
	|	`delete delegation` x ⟨**right**⟩ `->` p
⟨**expr**⟩	::=	⟨**value**⟩ | `[]` | {⟨**fieldvals**⟩}
	|	`if` ⟨**bool_expr**⟩ `then` ⟨**expr**⟩ `else` ⟨**expr**⟩
⟨**bool_expr**⟩	::=	`true` | `false` | `equal`(⟨**value**⟩, ⟨**value**⟩) | `notequal`(⟨**value**⟩, ⟨**value**⟩)
⟨**fieldvals**⟩	::=	x = ⟨**value**⟩ | x = ⟨**value**⟩, ⟨**fieldvals**⟩
⟨**value**⟩	::=	x | $x.y$ | s
⟨**right**⟩	::=	`read` | `write`

In the theoretical portion of this assignment only, you may ignore the parts that are grayed out. You have the option to analyze {⟨**fieldvals**⟩} and $x.y$ for extra credit. All of the above syntax,

including the grayed out parts, must be implemented in OCaml for the practical portion of this assignment.

1 Information flow types: theory (30 points)

In this problem, you will provide an operational semantics for this language and typing rules and then prove (partial) soundness of the type system. For full soundness, we want to prove properties about both *confidentiality* and *integrity*. For the purposes of this assignment, however, we will focus on confidentiality: more specifically, noninterference.

(a) (1 point) Provide two examples of information leaks, and describe how a type system could prevent such leaks.

(b) (1 point) What should the environment σ map?

(c) (4 points) Provide an operational semantics for expressions of the form

$$\langle \sigma, \langle \textbf{expr} \rangle \rangle \Downarrow s,$$

excluding records. (Note that records are grayed out because these are extra credit.)

(d) (6 points) Provide an operational semantics for commands of the form

$$\langle \sigma, \langle \textbf{cmd} \rangle \rangle \Downarrow \sigma',$$

excluding records. Consider each $\langle \textbf{prim\_cmd} \rangle$ separately.

(e) (2 points) Each program runs under a specific principal p with a specific set of permissions (that may change during the course of execution). The goal of type system is to make sure that no action occurs that violates these permissions. Even though we have different principals here, we can use a type system similar to the one we saw in class, where variables that a principal is allowed to read are assigned L (low, as opposed to H—high), and variables a principal is allowed to write are assigned U (untrusted, as opposed to T—trusted). What is the security lattice $L = (SC, \leq, \sqcup, \sqcap, \bot)$, where $SC = \{LU, LT, HU, HT\}$ (as seen in Lecture 12)?

(f) (1 point) When a program is run as principal p, what does Γ map?

(g) (1 point) When, if at all, do we need to track a program counter variable **pc**? Explain your answer.

(h) (4 points) Provide the rules for the typing judgment $\Gamma_p \vdash \langle \textbf{expr} \rangle : \tau$ for expressions. Doing this will also involve providing rules for the typing judgment $\Gamma_p \vdash \langle \textbf{bool\_expr} \rangle : \tau$.

(i) (4 points) Provide the rules for the typing judgment $\Gamma_p \vdash \langle \textbf{cmd} \rangle$ for commands. Consider each case of $\langle \textbf{prim\_cmd} \rangle$ separately.

(j) (1 point) State the Noninterference property we want to prove to show soundness.

(k) (4 points) Prove Simple Security for your typing rules.

(l) (1 point) Prove Confinement for your typing rules.

(m) (**Extra credit:** 5 points) Prove Noninterference for your typing rules.

(n) (**Extra credit:** 4 points) Add rules for records, and extend the proofs of Simple Security and Confinement.

Solution

(a)

(b)

(c)

(d)

(e)

(f)

(g)

(h)

(i)

(j)

(k)

(l)

2 Information flow types: practice (20 points)

Now we will ask you to implement your information flow type system.

(a) (2 points) Declare a new data type `sc` for security classes, as well as functions for implementing the lattice.

(b) (3 points) Implement a function `typecheck_exp: principal -> exp -> sc` that checks expressions. You do not need to implement checking for programs not defined according to the modified syntax, and you may return an `Unimplemented` error in these cases. (We will not write test cases containing any of those cases.)

(c) (5 points) Implement a function `typecheck_cmd: principal -> cmd -> bool`.

(d) (8 points) Implement a comprehensive test suite that includes both tests that type-check and those that do not. In the space provided below, briefly document your test cases and their expected outputs.

(e) (2 points) Discuss the advantages of using type-checking for information flow, as opposed to the reference monitor from Assignment 2. Under what circumstances, including hypothetical extensions to the scripting language and/or permissions, might each be favorable?

Solution

(d)

(e)