Secure Programming via Libraries (T3)
ECI 2011 - Day 2

Alejandro Russo

Chalmers University of Technology

Exercise 1 Monad `Sec`

For this exercise, we only assume two security levels: `H` and `L` for secret and public information, respectively.

1. We want to implement a function that takes two public booleans and returns the integer that those boolean values represent when interpreted as binary. The most significant bit is the last boolean. More precisely, we want to implement the function

   ```
   binToInt2 :: Sec L Bool → Sec L Bool → Sec H Int
   ```

   For instance, if we apply the function to monadic values containing `True` and `False` (in that order), the returned secret number must be `1`. If the function is applied, on the other hand, to monadic values containing `False` and `True` (in that order), then the secret number must be `2`.
2. Implement the function

   ```
   binToInt :: [Sec L Int] → Sec H Int
   ```

   that takes a list of public booleans and return the secret number obtaining by interpreting that list as a binary number. The most significant bit is the last element of the list.

Exercise 2 Monadic values of type `Sec` inside monadic values of type `Sec`

When writing problem, it might happen that it appears nested `Sec s` values. For instance, a program might be manipulating values of type `Sec H (Sec L a)`, `Sec L (Sec H a)`, and so on. It is sometime useful to remove such nested levels if possible. For example, having a value of type `Sec H (Sec H Int)` is indicating that we have a computation producing a secret (first `Sec H`), which is another computation producing a secret integer (`Sec H Int`).

1. Implement a function of type

   ```
   simplify1 :: Sec H (Sec H a) → Sec H a
   ```

   that shows that nested `Sec H` can be simplified.
2. Implement a function of type

   ```
   simplify2 :: Sec H (Sec L a) → Sec H a
   ```

   that indicates that a public value inside a `Sec H` is indeed a secret value.
3. Implement a function of type

   ```
   simplify3 :: Sec L (Sec H a) → Sec H a
   ```

   using `simplify1` and the API of `SecLib`.
4. Can you generalize `simplify1`, `simplify2`, and `simplify3`? (Hint: you might need to use polymorphism and `Less`)

Exercise 3 Monad `SecIO`

In the lecture we describe how to write a function that securely copy files, i.e., how to ensure that secret files cannot be copy into public ones. Similarly, in this exercise, you need to implement a secure concatenation of files. Function `cat2` takes three files as arguments and concatenate the content of the

first two into the third one. This function must preserve non-interference, i.e., it should not be possible to concatenate two secret files and store the result of that into a public one. More precisely, the concatenation function in the untrustworthy module must have the type

```
cat2 :: (Less s1 s3, Less s2 s3) ⇒
        File s1 → File s2 → File s3 → SecIO s3 ()
```

Given the following untrustworthy module

```
module CatU where

import SecLib.LatticeLH
import SecLib.Untrustworthy

cat2 :: (Less s1 s3, Less s2 s3) ⇒
        File s1 → File s2 → File s3 → SecIO s3 ()
```

1. Complete the definition of `cat2` (so far, we have only the type of it).
   Once you have done that, you can use the following trustworthy module that uses the untrustworthy one.

   ```
   module CatT where

   import SecLib.LatticeLH
   import SecLib.Trustworthy

   import CatU (cat2)

   s_file1 :: File H
   s_file1 = mkFile "SecretFile1"

   s_file2 :: File H
   s_file2 = mkFile "SecretFile2"

   p_file :: File L
   p_file = mkFile "PublicFile"


   execute :: IO ()
   execute =  do revealIO $ cat2 s_file1 p_file s_file2
                 return ()
   ```

   Load the module `CatT` in `ghci` and run the untrustworthy code by calling `execute`.
2. What does it happen if execute is instead defined as

   ```
   execute =  do revealIO $ cat2 s_file1 s_file2 p_file
                 return ()
   ```

Exercise 4 Potentially dangerous `Sec` or `SecIO` computations

As the trusted programmer, you are responsible to determine the type of the untrusted functions. In the lectures, we show that untrustworthy computation returning values of type, for instance, `Sec H (IO ())` or `Sec H (Sec L (IO ()))` might compromise security if the trustworthy code execute the monadic value of type `IO ()`.

Which one of the following types might be potentially dangerous to consider when determining the security type of a computation returned by untrustworthy code. Justify every answer that you provide.

1. `SecIO H (IO ())`
2. `IO (SecIO L Int)`
3. `SecIO L (SecIO H Int)`

4. `Sec H (SecIO L (IO ()))` Moreover, can you ever execute the `SecIO` computation inside `Sec H` using `revealIO`?
5. `SecIO H (SecIO L Int)`