LANGUAGED-BASED TECHNOLOGY FOR SECURITY
Homework Assignment 1

Due: April 27 2022

Introduction

In this homework, you will design a simple functional language having primitive abstractions for securing the execution of mobile code. Mobile code is a programming approach where programs are considered as data, and therefore are accessible, transmitted, and evaluated.

Mobile code adds new challenges for making software secure. Note that web applications have made mobile code a part of everyday life: visiting a web page typically loads JavaScript code from multiple providers into browsers. Cloud services allow third parties to provide applications that are dynamically combined with the core cloud functionality. Finally, even traditional desktop applications dynamically download plugins from external providers.

In this assignment, we refer to mobile code as software that travels on a heterogeneous network, crossing protection domains, and automatically executed upon arrival at the destination. Protection domains can be as big as corporate clouds and as small as smartphones.

The adoption of code mobility introduces a scenario where private and sensitive information is being pushed outside the perimeter of a trusted execution environment. Indeed, the benefits of freely sharing mobile code come at a cost: dynamically combining code from multiple sources might yield an insecure application. The main security safeguard is the same-origin policy which attempts to limit applications to communication with their originating location. This policy prevents many useful applications yet also fails to address all the ways that untrusted code can create security vulnerabilities.

The assignment

In this homework you will design and build in OCAML the trusted execution environment of a functional language with linguistic primitives for mobile code and secure programming.

The programming language and its trusted execution environment support free and flexible sharing of mobile code while enforcing the security policies at hand. Also assemblies of code and data from various providers satisfy all participants' security requirements.

The trusted execution environment must include

- The interpreter of the language
- Run-time modules that provide facilities needed to control and enforce the security policies constraining access to local resources by mobile code.

To handle code mobility, we assume that the language provides the primitive operator **execute**(e). The execute operator evaluates its argument expression (i.e. the expression e has to be understood as the mobile code to be executed).

For instance, the program

execute((fun x = x+1) 5);;

evaluate the mobile code ((fun x = x+1) 5) yielding 6 as result.

The program

let mysum = (fun x -> (fun y -> x + y));;

execute(let result = mysum(5, 5));

when evalutated yields 10 as result only in the case that the execution environment grants to the mobile code the access to the local function **mysum**.

Finally, the program below

let mypin = 12345;;

execute(let result = myping in send(result))

when evalutated results in a security violation (private data transmitted over the network). In this example the execution environment grants to the mobile code the access to local variables.

To avoid security violations, we assume that mobile code (i.e. the execute operator) is evaluated inside a isolated context (e.g. a sandbox) that suitably restricts its permissions. To this purpose we assume that the sandbox is a

primitive construct which is equipped with a set of permissions (e.g. {read, write}, {send}, {}) over a set of security relevant actions.

We also assume that the sandbox supports checking of the declared permissions. In other words, the execution of the mobile code proceeds if permissions are enabled; otherwise, execution fails.

Note

You can design the programming language and its trusted execution environment in total freedom. For example, you can freely equip the language with further suitable linguistic abstractions. but you have to justify all the choices made.

Learning objectives

The goals of the homework are to appreciate and understand the trade-offs in the design and implementation of an experimental secure-aware programming language. Considering the issues of mobile-code security in the setting provided by the assignment has the advantage of addressing a more general, abstract instantiation of the real problems.

Submission Format

The submitted homework assignment must include:

1. The definition of the programming language along with some examples.
2. The source code of the trusted execution environment of the language.
3. A short description of the design choices.

The parts (1-3) above are *mandatory*.

The homework also includes an *optional* part briefly discussed below. Property-based testing (PBT) -- or automatic specification-based testing -- is a form of black-box testing where a software artifact is subjected to a large number of checks derived from formal statements of its intended behavioral properties. For example, if the program under test is a function **f** that takes a list of numbers as its input and (hopefully) returns the same list in sorted order, then a reasonable specification of **f** might be **sorted(f(oldlist)) permutation(f(oldlist),oldlist),** where sorted is a simple function that checks whether its input is ordered and permutation checks whether one of its argument lists is a permutation of the other. To test whether **f** satisfies this specification, we generate a large number of random input lists, apply **f** to each one, and check that sorted and permutation both yield true.

PBT was popularized in the functional programming world by the QuickCheck library for Haskell. QuickCheck has been ported to many other platforms (e.g. see https://github.com/c-cube/qcheck for OCAML).

The optional part of the homework consists in developing the property based testing of the proposed Trusted Execution Environment.

Submissions Guidelines

The assignment must be submitted no later than midnight of the day it is due.

More instructions on the submission next week.