Language-Based Technology for Security

Assignment #1

Antonio Strippoli, Giulio Paparelli

09/04/2021

For the realization of the project it was decided to reuse the **Ocaml** code that we have seen in class.

To represent possible permissions (in our case we limited ourselves to *read* and *write*), a new type `permission` was added with two constructors `Pread` and `Pwrite`. **To represent a set of permissions** we used `Set Module`, since it improve performances and readability of the code for the operations of intersection between the permissions assigned and requested by each caller.

Initially the request *"We also assume that the language is equipped with a primitive construct to check a permission"* was implemented through a syntactic structure `checkPermissions` in `expr`. This path was then abandoned as it seemed unrealistic to provide a construct to the programmer to verify the permissions that he himself assigns.

We have therefore (fictitiously) implemented three syntactic structures:

- `Read`, which requires `Pread` permission;
- `Write`, which requires `Pwrite` permission;
- `Open`, which requires `Pread` and `Pwrite` permissions.

The interpreter evaluates each of these structures by checking the presence of the necessary permissions using the `stackInspection` function, which takes as parameters:

- `p_asked`: the set of **required permissions** to perform unsafe operations;
- `p_stack`: a list of permissions sets that models the **permissions of functions currently on the stack** (env).

The function hence checks the following: $\forall$ p_caller *in* p_stack $\Rightarrow$ p_caller $\cap$ p_asked = p_asked.

To implement stack permissions handling, the following syntactic categories have been changed:

- `Fun`: **built-in** (in addition to the definition environment) **the permissions desired by the developer** both in the definition of a function and in the generation of its closure;
- `Call`: we have **changed the evaluation of the closure**, which now updates p_stack with the permissions found in the closure in a similar way to how the environment is updated.

Finally, we have **tested the interpreter** specifically with regard to *stack inspection*, defining two nested functions with permissions that are not (always) consistent, so as to be blocked when illegal operations were attempted.