Final Report - A Sound Type System for Secure Flow Analysis
Independent Study - Coq Proof Assistant

Danilo Dominguez Perez

Introduction

This document presents an study of the implementation of the proofs for a sound type system presented in Volpano et al. [1]. During this project, we have deviate from the paper proofs in order to make the mechanize proofs easier to implement. Additionally, we have faces several challenges that are overlooked in the paper proofs. In this document, we include an analysis of our experience using the interactive theorem prover Coq [2] and make some suggestions for the community.

Challenges

Implementation of Big-Step Semantics

The first challenge faced is the implementation of big step semantics for the ***letvar*** construct.

letvar *x* ***:=*** *e* ***in*** *c*

The *letvar* construct binds the result of an expression to a variable *"x"* and substitute the value of the expression in appearence of variable *x* in the command "*c"*. I am not following exactly the semantics they have defined in the paper. I will use two environments, one for locations μ and one for variables δ. The environment for variables will keep a partial map from variables to location. The environment for locations will keep a partial map from locations to natural numbers.

Implementation of Heaps

Heaps were implemented using the built-in lists provided by Coq. Additionally, several functions over the heaps were defined to be used in the operational semantics and also in supporting lemmas. The following list describe some of these functions:

- update_heap: updates the value of a location if the location was in the domain, otherwise it extend the heap, adding the value at the end of the list.
- dom_heap: return the list of the locations in the heap.
- loc_is_in_dom: returns true of the location is in the domain of the heap. Our first implementation was a normal fixpoint that finds through the list if the location is in the heap. However, this implementation causes some problems in supporting lemmas (especially lemmas 6.6 and 6.7 in [1]). Therefore, we implemented this function as a relation (loc_is_in_dom') that we use in our proofs. We also proved that the two definitions are equivalent.
- first_loc: returns the first location available that is not in the domain of the heap. This property was necessary to prove determinism of the operational semantics.

We also proved some important properties about heaps that were used in the supporting lemmas described in [1]:

- heap_invariance: is a location was already in the domain of a heap, then an update of the heap does not change the domain

```
Lemma heap_invariance' : forall l hp n,
  loc_is_in_dom' l hp  ->
  dom_heap hp = dom_heap (update_heap hp l n).
```

- equivalence between lookup for a location that is not in the domain of the heap

```
Lemma not_in_dom_lookup_none : forall hp l,
  lookup_heap hp l = None <-> ~loc_is_in_dom l hp.
```

See the file *SecTypesDef.v* for all the definitions.

Implementation of the Type System and Theorems

In the paper [1], just some rules from the commands were defined in a syntax directed version. In our case, all the typing rules for the commands (*has_type_com*) are syntax directed rules.

```
| T_Seq : forall Lambda Gamma t t' c c',
    Lambda ,, Gamma |- c \in (TCmd t) ->
    Lambda ,, Gamma |- c' \in (TCmd t) ->
    t' <= t ->
    Lambda ,, Gamma |- CSeq c c' \in (TCmd t')
```

However, to prove *Security Confinement*, we had to define a lemma for suptyping using a subtyping relation (*subtype*):

```
Lemma security_subtype : forall Lambda Gamma c T T',
  Lambda ,, Gamma |- c \in T ->
  subtype T' T ->
  Lambda ,, Gamma |- c \in T'.
```

With this lemma and the syntax directed rules, *Security Confinement* was easily proved.

Supporting Lemmas for Type Soundness

The supporting lemmas for the proof of type soundness are defined in lemma 6.6 and lemma 6.7 in [1]. The paper define these proof assuming many properties over the heaps. In our case we had to prove several lemmas in order to finish the proof 6.6. For instance, we rely on the lemmas:

- heap_invariance: previously described
- dom_remove_heap: if a location is not in the heap, then if the heap is extended with the location and then the location is removed, the domain does not change.

```
Lemma dom_remove_heap : forall hp hp' l n,
```

```
~loc_is_in_dom' l hp ->
dom_heap hp' = dom_heap (update_heap hp l n) ->
dom_heap hp = dom_heap (remove_heap hp' l).
```

For the lemma 6.7 and the final type soundness proof, we faced some difficulties. Therefore, we were not able to prove lemma 6.7 and the final type soundness.

Conclusion

We have implemented several of the proofs in the type system presented by Volpano [1]. Our limitations with the heaps and its properties make some of the proofs, that the paper overlook, hard to proof in Coq. Papers normally based their proof under assumptions that in a proof assistant such as Coq are inexistent. Normally. the programmer has to implement all the infrastructure and supporting lemmas that are assumed in the paper proofs. This is an aspect the must be improved in the standard libraries in Coq.

References

1. Volpano, Dennis, Cynthia Irvine, and Geoffrey Smith. "A sound type system for secure flow analysis." *Journal of computer security* 4.2 (1996): 167-187.
2. The Coq Proof Assistant. http://coq.inria.fr/