Program Analysis, January–April 2014

Project 1, 5 March 2014
Due Friday, 4 April 2014

Instructions for submitting solutions

Please submit your solutions electronically by email to facilitate evaluation.

Preferrably, send an electronic document in PDF (you can generate it in LaTeX or OpenOffice or Word or whatever, but send only PDF). If you can't do this, send scanned copies of handwritten pages.

Detection of information leakage

Sensitive information is often stored in databases and processed by programs. For example, passwords, credit card information etc. are routinely stored and processed. The problem is to ensure that such sensitive information is not accessible from outside a set of "trusted" functions/procedures/methods – in other words to prevent *leakage* of sensitive information.

Assume that the programming language under consideration has the following features:

- The usual control constructs such as sequencing, if-then-else, while loops and functions.
- Value and reference parameter passing.
- Data types including int, String and simple classes/records/structures (without subtyping/inheritance).
- Each field of a class can optionally be tagged as a `sensitive` field.
- Each class has a set of associated 'methods' – only these methods are meant to access the sensitive fields. Any potential access of a sensitive field outside of a method is an information leakage.

A trivial example with an information leak is:

```
class Account {
    ...
    String owner;
    sensitive String password;
    ...
}

void Account:getDetails (ref String s1, ref String s2, ...)
{
    s1 = owner;
    s2 = password;
    ...
}
```

```
main ()
{
    Account ac;
    String nm, pwd;
    ...
    ac = getAcFromDataBase ("name1");
    ...
    ac.getDetails (nm, pwd, ...);
    print ("Account info:\n\tName:$nm\n\tPassword:$pwd\n);
    /* This is an obvious leakage!!  */
}
```

Devise a program analysis that will help detect all such possible leaks of sensitive information. Note that the results of the analysis need not by itself point out the sources of information leakage, but it should be possible to easily compute such sources of leakage from the result of the analysis[1].

You are free to choose any technique of analysis, but make sure the following are covered:

1. The information structure (lattice or type system), its ordering etc.
2. The semantic functions or type inference rules.
3. A justification, preferably with a worst case complexity, as to why the analysis would terminate for any program in the given language.
4. A justification why the computed information is sound and consistent with the semantics of the language.
5. How can the result of this analysis be used to identify sources of information leakage.

[1]For example, interval analysis by itself does not help in detecting array index bounds violations, but given the results of interval analysis and sizes of each array, it is trivial to determine likely array bound violations.