CS5218 Assignment 2 – Taint Analysis

Student Information

Name : Hady Januar Willi
Student ID : A0174396R
School Email : E0210429@u.nus.edu
Email : hadywilli@outlook.com

Tainted Variables Analysis Design

$$TV_{entry}(\ell) = \begin{cases} \emptyset & if\ \ell = init(S_*) \\ \cup \{TV_{exit}(\ell') | (\ell', \ell) \in flow(S_*)\} & otherwise \end{cases}$$

$$TV_{exit}(\ell) = \left(TV_{entry}(\ell)\right) \cup gen_{TV}\left(B^{\ell}\right) where\ B^{\ell}\ \in blocks(S_*)$$

$$gen_{TV}([z := a]^{\ell}) = \begin{Bmatrix} z & if\ a\ is\ tainted\ or\ is\ source \\ \emptyset & otherwise \end{Bmatrix}$$
$$gen_{TV}([z := a \blacksquare b]^{\ell}) = \begin{Bmatrix} z & if\ a\ or\ b\ is\ tainted\ or\ is\ source \\ \emptyset & otherwise \end{Bmatrix} where\ \blacksquare\ is\ a\ binary\ operation$$
$$gen_{TV}([skip]^{\ell}) = \emptyset$$
$$gen_{TV}([b]^{\ell}) = \emptyset$$

$$kill_{TV}([z := a]^{\ell}) = \{ \begin{matrix} \emptyset & if\ a\ is\ tainted\ or\ source \\ z & otherwise \end{matrix} \}$$
$$kill_{TV}([z := a \blacksquare b]^{\ell}) = \begin{Bmatrix} \emptyset & if\ a\ or\ b\ is\ tainted\ or\ is\ source \\ z & otherwise \end{Bmatrix} where\ \blacksquare\ is\ a\ binary\ operation$$
$$kill_{TV}([skip]^{\ell}) = \emptyset$$
$$kill_{TV}([b]^{l}) = \emptyset$$

Pattern	Instance
L	$P(Var_*)$
$\sqsubseteq$	$\subseteq$
$\sqcup$	$\cup$
$\bot$	$\emptyset$
ι	$\emptyset$
E	$init(S_*)$
F	$flow(S_*)$
$\mathcal{F}$	$\{f : L \to L \mid \exists l_g : f(l) = (l \setminus l_k) \cup l_g\}$
f_ℓ	$f_\ell(l) = \left(l \setminus kill(B^{\ell})\right) \cup gen\left(B^{\ell}\right) where\ B^{\ell} \in blocks(S_*)$

This analysis is a may analysis where the variable listed may or may not be tainted in that block depending on tainted variables from the previous block. If the variable is assigned to a constant or untainted variables or operation of untainted variables.

Analysis Implementation

The analysis designed above is implemented using LLVM version 7 in a clean Ubuntu 18.04 virtual machine. The full LLVM installation script is included together with the submission package.

The implementation for both task 2 and task 3 changed the function names to reflect closer to monotone framework terminologies. The kill and gen are evaluated at the same time for considering variables and registers considered as sink. The implementation uses simple depth first search by recursive flow function. The notion of node is LLVM basic block and path is LLVM basic block parent – child relation in LLVM flow. Three types of instructions are considered in this case for taint analysis:

1. Load. For each load instruction in the basic block, if source or any tainted variable is the operand, the instruction is stored in a sink list in the block. (i.e. load %0 source, add %0 to sink list). Otherwise remove it from the sink list.
2. Store. For each store instruction in the basic block, the operand is added into the current node tainted set and sink list if the store source either is found in sink list or is source. Otherwise remove it from the tainted set and sink list. This is the instruction that generates tainted variable set.
3. Binary Operations. For each binary operation in the basic block, the instruction is added to the sink if either operand is found in sink list or is source. Otherwise remove the variable from sink list.

Graph traversal continues if there is a successor, and if either one of the two conditions below is met:

1. The current node is not traversed yet
2. The tainted variable set size changed for the current node

Submission Folder Structure

```
> zip                -- Folder containing all files
    > bin            -- Folder containing compiled assignment binary
    > graphs         -- Folder containing dot and pdf files generated from LLVM intermediate representation
    > installation   -- Folder containing LLVM 7 installation script
    > llvms          -- Folder containing LLVM intermediate representation
    > source         -- Folder containing assignment source code
    > target         -- Folder containing target C code
```

Running the Implementation

After extracting the file, the following command can be run in sequence, the run command executes against example 1, 2, and 3 in assignment handout:

```
./compile
./llvm
./run
```

After compilation, to run against any other LLVM intermediate representation file, the following command can be used:
./bin/assignment2 other-file.ll

To generate the graphs the following command can be run:
./graph

If all else fails, the compiled binary is also included in the assignment, and can be used with the following command which hopefully runs with or without LLVM libraries:
./bin/assignment2 test.ll