18-732 Assignment 4A - Due Tuesday 4/28

Eliot Wong - ecwong, Steve Matsumoto - smatsumo May 2, 2015

Operational Semantics

Binary Operations

$$\frac{x1 = isTainted(exp1) \quad x2 = isTainted(exp2)}{exp1 \circ exp2 \downarrow x1 \vee x2}$$

Unary Operations

$$\frac{x = isTainted(exp)}{exp \downarrow x}$$

Assignments

$$\frac{x \leftarrow exp \quad t = isTainted(exp)}{x \downarrow t}$$

Memory Operations

$$\frac{t1 = isTainted(exp1); \quad t2 = isTainted(mem[exp1])}{mem[exp1] \downarrow t1 \vee t2}$$

Code

Most of the semantics for the checker were written into a new function in eval.c. The function taint_exp is called following eval_exp with the same arguments which will perform a similar analysis through the AST except checking for tainted variables and memory operations.

In the tables.c file, there are a couple new functions, taint_var and taint_addr. Both perform similar operations to return a taint value for a specified variable or memory location. Also update_var and store have both been modified to include a new parameter taint. This will be stored within the variable or memory struct for later references.

- **Binary Operations**: eval.c - Lines 132-319
- **Unary Operations**: eval.c - Lines 321-326
- **Assignments**: eval.c - Lines 116-122, tables.c - Lines 33-51, 53-79
- **Memory Operations**: eval.c - Lines 125-131, tables.c - Lines 81-112, 129-149

Test Cases

good1: A secret value is used in an assert, but is then reset with a clean value, causing no secrets to be leaked.

good2: mem[1] is set to a secret value, but then reset and used in other memory-based computations. In the end, no secrets are leaked even though the tainted address is reused.

good3: A secret value is set, but then a conditional is used to overwrite the secret value with a different one. Thus no secrets are leaked when the variable is printed.

bad1: x is a secret value, and y is not, but mem[y] is set to the value of x, tainting it. We show how computations involving each of these three causes only y to be printed, the others being suppressed due to taint.

bad2: a is a secret value, and b is not, while c is based on a conditional that compares a and b. Therefore, c inherits a's taint and is not printed, while b makes it through.

bad3: x reads from a tainted memory address, and its value is then used in z. Thus x and z are both tainted, while y (which reads a regular int) is printed normally.

control1: a and b are both secret, but an if-else conditional is used to determine whether they are zero or not and then print the result.

control2: mem[2] is set to a secret, and then x (assigned the value 2) is used for an if-else conditional based in mem[x]. However, no secret is suppressed due to the if-else conditional.

good1: A secret value is used in an assert, but is then reset with a clean value, causing no secrets to be leaked.

good2: mem[1] is set to a secret value, but then reset and used in other memory-based computations. In the end, no secrets are leaked even though the tainted address is reused.

good3: A secret value is set, but then a conditional is used to overwrite the secret value with a different one. Thus no secrets are leaked when the variable is printed.

bad1: x is a secret value, and y is not, but mem[y] is set to the value of x, tainting it. We show how computations involving each of these three causes only y to be printed, the others being suppressed due to taint.

bad2: a is a secret value, and b is not, while c is based on a conditional that compares a and b. Therefore, c inherits a's taint and is not printed, while b makes it through.

bad3: x reads from a tainted memory address, and its value is then used in z. Thus x and z are both tainted, while y (which reads a regular int) is printed normally.

control1: a and b are both secret, but an if-else conditional is used to determine whether they are zero or not and then print the result.

control2: mem[2] is set to a secret, and then x (assigned the value 2) is used for an if-else conditional based in mem[x]. However, no secret is suppressed due to the if-else conditional.