Compiler Infrastructure

Outline

- Codesurfer tool
- CCured (Phil)
- LLVM (Nirupama)

Detecting Buffer Overruns

PENNSTATE 1855

- Static analysis tool to detect buffer-overrun vulnerabilities in C source code
- Many previous tools have been built
  - Dynamic techniques – detect at runtime
  - Static techniques – remove vulnerable code before running
  - Combination – remove unnecessary runtime checks
- Advantages of static techniques vs. dynamic?

Tool Features

- Use static analysis to model C string manipulations as a linear program
- Build scalable solvers based on linear programming techniques
- Make program analysis context-sensitive
- Eliminate bugs from source code

System Architecture

- Figure 3.1
- C source → codesurfer → System dependence graph
  - Interprocedural control flow graph
- → Constraint Generator → Linear Constraints
  - Linear program constraints
- → Taint Analyzer → Linear Constraints
  - Remove those that are not suitable for solver
- → Constraint Solver → Ranges
- Then, warnings for cases that can lead to overflow

Example Program

- Focus on buf and header
  - Are they vulnerable?
- What does *fgets* do?
- How about *copy_buffer*?

Constraints

- Char* Constraints for *used* and *allocation*
- Char* Constraints for *min* and *max* value
- Integers just have value constraints
- Constraint from line 6
  - Header is assigned a value between size 1 and 2048
- Constraint from line 10
  - Relate buf, cc2 and function call, return

Constraints

- Are generated for the following statements
  - Buffer declarations
  - Assignments
  - Function calls
  - Returns
- Buffer declarations impact allocation constraints
- Assignments impact value constraints (for ints too)
- Function calls are modeled by constraints that summarize the effect of the call

Constraint Analysis

- Flow-insensitive
  - Do not account for order of statements
  - Find constraint in a statement
  - Collect constraints across statements
  - Composition of constraints does not account for order of statements or conditionals
- Context-insensitive
  - Does not distinguish among multiple call sites
  - Inputs of multiple calls may "mix" in the function
- Libraries are treated in a context-sensitive way

Pointers and Constraints

- Constraints represent buffers
- Choice for representing
  - Constraint on pointer to buffer or buffer memory itself
  - Choose former – false negatives: why?
- Pointer analysis to remove some false positives between pointers that are known to be related

Pointers and Constraints

- Use pointer analysis to eliminate some false positives
- Statement: *strcpy(p→f, buf)*
  - *p* can point to structure *s*
  - Thus, constraints should relate s.f and buf

Constraints and Linear Programs

- Statement: *counter*++
- The constraint *counter!max* >= *counter!max* + *1* is cannot be interpreted by a linear program solver
- Instead we create two constraints
  - *Counter'* = *counter* + *1*
  - *Counter* = *counter'*
  - Which are infeasible (more later)
- Also, constraints for pointer arithmetic are infeasible

Taint Analysis

- Perform taint analysis to make constraints amenable for linear programming solvers
  - Remove constraints with infinite values
    - E.g., User input
  - Remove constraints for uninitialized variables (no lower bound for max and upper bound for min)
    - E.g., Uninitialized vars
- Algorithm in 3.4
  - Returns subset of constraints with no infinite or uninitialized values

Constraint Solving

- Goal: obtain best possible estimate of the number of bytes used and allocated for each buffer in any execution of the program
  - Number of bytes used is the smallest range that satisfies all constraints
  - Number of bytes allocated is the smallest range that satisfies all constraints
- Will discuss two techniques later

Detecting Overruns

- Results of analysis
  - Header is allocated 2048 bytes, and between 1 and 2048 bytes can be used (is safe)
  - Same is true of buf
  - Ptr was found to have between 1024 and 2048 bytes allocated while 1 to 2048 bytes are used
    - Is a buffer overrun possible?
    - Could this be a false positive?
  - Copy allocated max is less than copy used max
  - cc1 and cc2 get same values, due to context-insensitivity

Linear Program Solvers

- Linear program
  - Minimize: *cx*
  - Subject to: *Ax* >= *b*
  - *A*: *m x n* matrix; *b*, *c* vectors of constants; *x is a* vector of variables
- System of *m* inequalities in *n* variables
  - Find values of vars such that system is satisfied and objective function takes its lowest possible value
- Works on finite, real values for x
- Methods to solve (Simplex)

Formulate as a Linear Program

- Set of constraints are linear, so can formulate a linear program
  - What is the objective function?
- Find smallest ranges for allocated and used values for buffers

- Problem: need to find *integer* values
  - That problem is NP-complete
- Solution: express A as a *unimodular matrix*
  - Every equation Ax = b where A is unimodular and A, b are both integer has an integer solution

Constraint Resolution (1)

- Problem: optimal solution may not exist
  - May not be *feasible*
  - May not be *optimal* (i.e., may be *unbounded*)
- This problem
  - No solution can be unbounded – due to taint analysis
  - Some infeasible constraints

Solution (part I)

- Try to remove some constraints to make solution feasible
  - Identify Irreducibly Inconsistent Sets (IISs)
- Use Elastic Filtering Algorithm to identify IISs
  - Takes set of linear constraints and identifies an IIS in these constraints
  - May have to run multiple times

Solution (part II)

- Approach for dealing with IISs
  - Find if C is feasible
  - If not, identify IISs as C'
  - C-C' is feasible
  - Set values of C' to infinite, and run taint analysis to remove some constraints C"
  - Result is feasible and bounded: C-(C'+C")

Another Approach

- Decompose constraints into subsets and solve each subset separately in an order that prevents backtracking
- Formulate constraints into DAG
  - Dependence among constraints
    - A variable depends on all constraints in which it appears n on LHS
    - Coalesce constraints that are mutually dependent (strongly connected)
  - Solve constraints in SCCs in topologically sorted order
  - If infeasible
    - Set to infinite ranges (no use of IIS)
- More precise representation of dependence leading to infeasibility than IIS

Summary

- Buffer overflow detection using static analysis to generate linear program constraints
  - Possible to create static analysis model
  - ICFG
  - Constraints from limited data flow (no joins)
  - Linear program is abstraction
- Need some additional effort to make abstraction work
- However, false negatives and false positives are possible

Questions