Static Taint Analysis for C with LLVM

Xavier Noumbissi

Department of Electrical and Computer Engineering
University of Waterloo

Problem: Prevent Software Vulnerabilities

- Format String Attacks
- SQL Injection
- Cross Site Scripting, etc.

Solution: Track Used of Non Trusted Program Input

- Non trusted program input: Tainted Input
- Source: origin of tainted input
- Sink: use of tainted input
- Taint Propagation: tracking tainted input

Example

```
int main() {
  int x, b1, b2, y;
  scanf("%d", &x);
  b1 = even(x);
  b2 = odd(3);
  y = compute(x);
  return 0;
}

int compute(int x) {
  int sum, i;
  if (x == 2)
    scanf("%d", &sum);
  else
    sum = 0;
  for(i = 0; i < x; ++i)
    sum += i;
  return sum;
}

int odd(int x) {
  if (x == 1)
    return 0;
  else
    return even(x - 1);
}

int even(int x) {
  if (x == 0)
    return 1;
  else
    return odd(x - 1);
}
```

Figure 1. Motivating Example

Solution: In This Project

- Implicit Taint Propagation: due to Control Flow
- Explicit Taint Propagation: due to Data Flow

Expected Contributions

- Algorithm to statically detect use of tainted values in C programs
- Handling of interprocedural taint propagation
- Implementation of the algorithm in LLVM

Taint Analysis: Relevant C Program Statements

Statement Type	C Code
COPY	$p = q$
LOAD	$p = *q$
STORE	$*p = q$
ADDROF	$p = \&a$
SOURCE	*call gets*

Taint Analysis: Transfer Functions

- COPY $[p = q]$: taint p iff q is tainted
- LOAD $[p = *q]$: taint p iff it exists $t_q = *q \wedge t_q$ is tainted
- STORE $[*p = q]$: nothing
- ADDROF $[p = \&a]$: nothing
- SOURCE $[call\ gets(p)]$: taint all t_q $\texttt{s.t}$ $t_q = *p$

Interprocedural, Context-Sensitive Analysis

- Analysis of a callee start with the taint assumptions from the caller

Taint Information from Source

- Developer specify sources and sinks in configuration file
- Analysis do not analyze sources and sinks
- Analysis use annotations for sources: taint propagation

Algorithm Analyze: Implements Interprocedural Analysis

```
input  : func : Proc,
         initDataFlow: Inst → (Var → 2^Inst)
output:
s0 ← first(f)
input[s0] ← initDataFlow(s0)
worklist ← {s0}
while worklist = ∅ do
    i ← next(worklist)
    output[i] ← Flow(Analyze, i)
    foreach j ∈ succs(i) do
        if output[i]   input[j] then
            input[j] ← input[j]   output[i]
            worklist ← worklist ∪ {j}
        end
    end
end
```

Algorithm 1: Analyze

Algorithm Flow: Implements Transfer Functions

```
input  : caller : Proc, s : Inst
output:
switch TypeOf(s) do
    case COPY [p = q]
        if inFlow_s[q] = ∅ then
            outFlow_s[p] ← inFlow_s[q] ∪ {s};
        end
    endsw
    case LOAD [p = *q]
        foreach a ∈ pt_[s](q) do
            if inFlow_s[a] = ∅ then
                outFlow_s[p] ← inFlow_s[p] ∪ {s}
            end
        end
    endsw
    case SOURCE [call func(a_0, a_1, ..., a_n)]
        foreach k ∈ {0, 1, ..., n} do
            if taint(k) then
                outFlow_s[a_k] ← inFlow_s[a_k] ∪ {s}
            end
        end
    endsw
    case CALL [call func(a_0, a_1, ..., a_n)]
        if caller = func then
            foreach a_k, k ∈ {0, 1, ..., n} do
                f_k ← formal(func, a_k)
                if a_k ∈ P then
                    foreach b ∈ pt_[s](a_k) do
                        inFlow_s[f_k] ← inFlow_s[f_k] ∪ inFlow_s[b]
                    end
                end
                else if a_k ∈ A then
                    t_k ← toplevel(a_k)
                    foreach b ∈ pt_[s](t_k) do
                        inFlow_s[f_k] ← inFlow_s[f_k] ∪ inFlow_s[b]
                    end
                end
            end
            Flow(caller, func)
        end
    endsw
    case ADDROF [p = &a]
    case STORE [*p = q]
    case SINK [call func]
    endsw
```

Implementation

- LLVM infrastructure is ready
- Need to implement the analysis

Future Work

- Make analysis modular
- Better handling of (mutual-) recursive function calls

Conclusion

- Implementation should take less than 1 months
- Need to do evaluation on real world C programs
- We are optimistic about future results

Thank You!

Comments & Questions