# Symbolic Analysis for Buffer Overflow

Surinder Jain
The University of Sydney

4th SAPLING meeting
Sydney, Australia

# Buffer Overflow

- Update/Read beyond bounds of buffer
- Results in
  - Erratic program behaviour
  - Program crashes
  - Security  breaches
- Caused by
  - Array access outside array limits
  - Pointer  reference errors
  - Array indicies errors

# Array access errors

```
i=0
array a[b-1]
while c < b and i>m and i<n
   i=i+1
   j=j+b
   d=2*d
   c=c+1
   a[c]=0 /* buffer overflow */
   if  e > 0
      f=f+2
   else
      e=g*e
```

- Variable array index
- Modified in a loop

•Buffer overflow during $(b-c)^{th}$ i.e last iteration of the loop.

# Static Program Analysis

- Analyse Program behaviour
- Without running the program

# Techniques

- Data flow analysis
- Constraint based analysis
- Abstract Interpretation
- Model checking
- Symbolic Analysis

# Symbolic Analysis & Execution

- Enumerate program paths
- Symbolic execution of each program path
- Execute a program with symbolic values
- Symbolic domains, predicates, semantics
- Relate symbolic results to concrete interpretation

# Array bounds violation Analysis

- Enumerate program paths in a loop
- For each program path, do
  - Symbolic execution
- Compare array indices with array bounds

# Example

```
i=0
array a[b-1]
while c < b and i>m and i<n
    i=i+1
    j=j+b
    d=2*d
    c=c+1
    a[c]=0 /* buffer overflow */
    if   e  >  0
        f=f+2
    else
        e=g*e
```

- Number of Loop iterations

  $= \min(b-c_o, m-n-2)$
- Value of c during i'th iteration (closed form of c) at line a[c]=0 is

  $c_i = c_o + i$
- Value of c in final iteration is

  $c = c_o + (b - c_o)$

  $= b$
- Hence statement a[c]=0

  when c=b causes buffer overflow

  in program path where $m-n-2 >= b-c_o$

$c_o$ is value of c at the start of the program

# Problem in General

- Undecidable
- Enumerate program paths
  - State explosion problem
  - How to do it for
    - General programs with GoTos
    - Programs with Loops
- Symbolic execution of a Loop
  - Unknown repetitions
  - State explosion
  - Can’t be solved for every case using one algorithms

# Enumerating Program Paths

- Gulwani et al.
  - non-deterministic semantics - no GoTos
- Burgstaller et al.
  - Path expressions  algebra – with GoTos
  - Loops as black boxes
- Extend non-deterministic semantics to control flow graphs
- Loop paths analysis
- Algorithm to
  - Enumerate Disjoint Program paths for any program

# Symbolic execution

- Algorithm to
  - Do symbolic execution
  - Compute path condition
- Eliminate invalid paths
- For each loop in a program path, solve
  - Closed form of loop induction variables
  - Loop counter

# Solving Program Loops

```
i=0
array a[b-1]
while c < b and i>m and i<n
   i=i+1
   j=j+b
   d=2*d
   c=c+1
   a[c]=0 /* buffer overflow */
   if  e  >  0
      f=f+2
   else
      e=g*e
```

Recurrence System (for j) : j(i+1)=j(i)+b

Solution is : j(i)= j(0)+i*b

Recurrence system for d : d(i+1)=2*d(i)

Solution is : $d(i)=2^{i}*d_{0}$

# Solving Program Loops ….

```
i=0
array a[b-1]
while c < b and i>m and i<n
   i=i+1
   j=j+b
   d=2*d
   c=c+1
   a[c]=0 /* buffer overflow */
   if   e  >  0
      f=f+2
   else
      e=g*e
```

Loop continue condition :
1.i  > m  :: i is in range [m+1,infinity]
2.i  < n  :: i is in range [-infinity,n-1]
And'ing the two we get : i is in range [m+1,n-1]

Loop non-entry condition is : m+1>n-1
Loop entry condition is : m+1<=n-1
Loop counter is : (n-1)-(m+1)=n-m-2

Values at end of loop :

$j = j_0 + (n-m-2)*b$

$d = d_0 * 2^{(n-m-2)}$

# Solving Program Loops ….

- Computer Algebra algorithms to
  - Solve recurrence systems
  - Solve loop exit condition
  - Solve loop counters as symbolic expressions
- Use skolmisation techniques  for unsolvable cases

# Path State Explosion

- Extend the notion of Burgstaller's path expressions
- Extend Gulwani's semantics
- Combine them together defining new
  - non-deterministic domains

# Path enumeration

```
if  e > 0
    f=f+2
  Else
    e=g*e
```

Two program paths :
(1) Assume(e>0);f=f+2
(2) Assume (not e>0); e=g*e

```
i=0
array a[b-1]
while c < b and i>m and i<n
   i=i+1
   j=j+b
   d=2*d
   c=c+1
   a[c]=0 /* buffer overflow */
   if  e > 0
       f=f+2
   else
       e=g*e
```

Program Path : i=0; $x^{mu}$

- x is loop body and
- mu is number of loop iterations.
- $2^{mu}$ deterministic program paths in
  - concrete domain as well as in
  - symbolic domain.
- Only 3 non-deterministic program paths
  - c>=b
  - c<b & $e_i$>0 & $x_k$
  - b<d & $e_i$<=0 & $x_k$

# Experimental work

We develop program to

- Enumerate  disjoint program paths
- path conditions
- Interface with a Computer Algebra System to
  - Obtain closed form for loop induction variables
  - Solve loop exit condition
  - Solve loop counter
- Perform symbolic execution
- Array bound comparison
- Reporting array bound violation (Static analysis)

# Experimental work ….

- Analyse a small program
  - for all possible paths
  - to crash it

Or

- Analyse an open source system
  - for Buffer Overflow reporting
  - Prioritised as
    - Definite
    - May be
  - With full set of counter examples

# Summary

- Array  access  in loops causes buffer overflow error
- Enumerate disjoint program paths  & conditions
- Non-deterministic semantics for path expressions
- Symbolic execution of program paths to identify buffer overflows
- Program loops cause path enumeration state explosion
- Non-deterministic domains to reduce state explosion

# References


References

[1] Johann Blieberger Bernd Burgstaller, Bernhard Scholz. Symbolic Analysis An Algebra-based approach.

[2] G. Canfora, A. Cimitile, and A. De Lucia. Conditioned program slicing. Information and Software Technology, 40(11-12):595607, 1998.

[3] T. Fahringer and B. Scholz. Advanced symbolic analysis for compilers. Springer-Verlag New York, Inc. Secaucus, NJ, USA, 2003.

[4] Michael P. Gerlek, Eric Stoltz, and Michael Wolfe. Beyond induction variables: detecting and classifying sequences using a demand-driven ssa form. ACM Trans. Program. Lang. Syst., 17(1):85122, 1995.

[5] S. Gulwani, S. Jain, and E. Koskinen. Control-refinement and progress invariants for bound analysis. PLDI, 2009.

[6] M.R. Haghighat and C.D. Polychronopoulos. Symbolic analysis for parallelizing compilers. ACM Transactions on Programming Languages and Systems (TOPLAS), 18(4): 477518, 1996.