SAFECODE: Static Analysis For safe Execution of Code

Aman Bhatia, Indian Institute of Technology, Kanpur, India
ADVISOR: Vikram Adve

Background
Unsafe languages provide very weak semantic guarantees due to the possibility of undetected memory errors:

- Dangling pointer references
- Array bounds overflow
- Arbitrary casting between types

Systems written in these languages are vulnerable to security attacks.

SAFECode - Compiler and Runtime System to enable development of reliable software

SAFECode Goals
- Guarantee Memory Safety: No uninitialized pointer uses, no array overflows
- Make dangling pointers safe
- Completely Automatic: No Wrapper, No GC
- Work for arbitrary C programs
- Very low overhead for C programs

 Fundamental Questions/Challenges
- Array Bounds checking for C: Why is it difficult? Don’t know the start of target array (referrent) of a pointer

    foo(&p[i], ...);
    void foo(int *q, ...)
    {
        ...
    }

    &ref=lookup(q);
    r = q[40];
    boundscheck(ref, r);
    ...

- Given a pointer how can we get start and size of its target array?

  Fat Pointers:
  Represent pointer logically as (p, target)
  Fast but Incompatible solution [SafeC, CCured, Cyclone, ...]

  Maintain search tree of allocated objects:
  [Jones-Kelly]

  Backwards Compatible but Slow: 4x-11x slowdowns
  SAFECode uses an improved Jones-Kelly approach for bounds checking: Split memory into distinct partitions, use separate trees per partition
  [Dhurjati, Adve, ICSE 2006]

Research Plan
- Identify bottlenecks during the runtime performance by profiling.
- Investigate lookup overhead.
- Reduce number of lookups.
- Increase speed of lookups.

Research Results
- Split lookup and check methods to eliminate redundant lookups in loops.
- Used constraint solver to reduce number of runtime checks.

Interaction with Other Projects
SVA: Secure Virtual Architecture
http://sva.cs.uiuc.edu
[Criswell et al, SOSP 2007]

Automatic Pool Allocation
[Lattner, Adve, PLDI 2005]

LLVM: Low Level Virtual Machine
http://llvm.org