

Finding Bugs

Last time

- Run-time reordering transformations

Today

- Program Analysis for finding bugs, especially security bugs
 - problem specification
 - motivation
 - approaches
 - remaining issues

Problem

What is a bug?

- a path in the code that causes a run-time exception
- a path through the code that causes incorrect results

Issues

- exponential many paths
- cannot statically determine the path a program will take
- “Program testing can be used to find the presence of bugs, but never to show their absence.” [Dijkstra 1972]

Undecidability

- soundness and completeness together is undecidable
- confusion in literature: which is which?
 - every reported error is genuine (no false positives)
 - if the program has any errors then the checker will report some error (no false negatives)

Motivation for the Automatic Detection of Bugs

Time spent in program maintenance

- most software engineers spend the majority of their time doing maintenance
- most time spent doing maintenance is time spent debugging

Costs due to bugs that allow security exploits (approximations published at CNET News.com, Jan 31 2003)

- Slammer (950 million)
- Code Red (2.6 billion productivity loss)
- LoveLetter (8.8 billion)
- Klez virus (9.0 billion)

Approaches to Finding Bugs

Approaches

- strengthening the type system
- static analysis to detect bug patterns
- automated theorem proving
- dynamic analysis
 - catch errors before they occur
 - find the cause for failures after the fact

Evaluating the different approaches

- how many false positives?
- how many false negatives?
- extent of user intervention or ease of use
- efficiency of approach

Example Bugs

null dereference

```
if (p==null) {  
    p->open()  
}
```

array bounds error

```
int a[20];  
a[20] = ...;
```

untrusted access

```
- format string vulnerability  
fgets(buffer, size, file);  
printf(buffer);
```

Type Qualifiers [Shankar, et al '01]

Idea

- Add tainted and untainted types to library function signatures

```
fgets(tainted char *buffer, int size, FILE *f);  
printf(untainted char *format, . . .);
```

- Use type constraint solver to find errors
 - Errors are type mismatches

Issues

- What is the type of `stderr()`?
- What happens when the value of strings change?

Static Analysis

FindBugs

- project at University of Maryland for finding bugs in Java
- they observe that bugs found in student programs are also found in production code
- implementation steps:
 1. think of the simplest technique that would find occurrences of the bug
 2. implement it
 3. apply it to real software. Hopefully find some real bugs. Will probably produce some false warnings.
 4. add heuristics to reduce percentage of false warnings

Their experience: new detectors can usually be implemented quickly (somewhere between a few minutes and a few days). Often, detectors find more bugs than you would expect

Kinds of analysis in implementing detectors:

- Examination of method names, signatures, class hierarchy
- Linear scan of bytecode instructions using a state machine
- Method control flow graphs, dataflow analysis
- No interprocedural flow analysis or sophisticated heap analysis

How FindBugs Handles the Example Bugs

Null pointer dereferences

- found 37 in rt.jar 1.5-b59, 55 in eclipse-3.0

Array bounds checking

- not an issue in Java

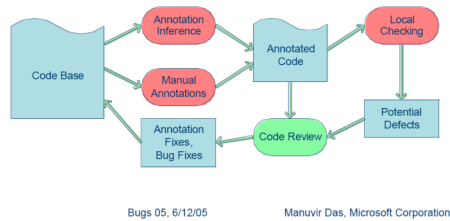
Untrusted Code

- Can static fields (or the objects they refer to) be modified by untrusted code?
 - Public, non-final static fields
 - Public static fields pointing to an array
- Warnings: 254 in rt.jar 1.5-b59, 967 in eclipse-3.0

Automated Theorem Proving

SAL at Microsoft

- Standard Annotation Language for interface pre and post conditions
- focus is on buffer overruns and pointer usage
- SALinfer is a tool that determines specifications automatically



SAL Example

```

    Requirement on foo's callers: must pass a buffer that is len elements long
    void foo(pre elementCount(len) int *buf, int len)
    {
        Assumption made by foo: buf is count elements long
        ...
        Requirement on foo: argument buf is len*4 bytes long
        memset(buf, 0, len*sizeof(int));
    }
    Requirement on memset's callers: must pass a buffer that is len bytes long
    int *memset(pre byteCount(len) void *dest, int c, size_t len);
  
```

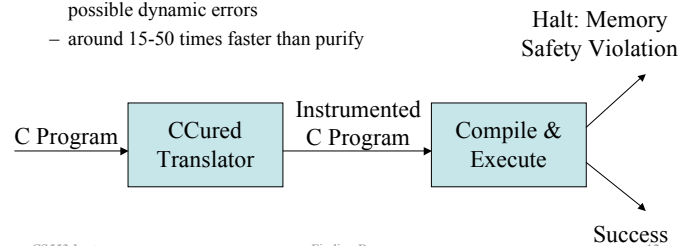
Bugs 05, 6/12/05

Manuvir Das, Microsoft Corporation

Dynamic Analysis

“Ccured: Taming C Pointers” by George Necula, Scott McPeak, and Wes Weimer, May 22, 2002

- adds run-time checks to C programs for catching memory safety errors
- requires user annotations
- the only thing that happens statically is figuring out what special type a pointer should be, want fastest possible type that still can catch any possible dynamic errors
- around 15-50 times faster than purify



How Ccured Handles the Example Bugs

New Pointer Types

- SAFE pointer: on use does a null pointer check
- SEQ pointer: on use does a null pointer check and an array bounds check
- DYN pointer: on use does a null pointer check, a bounds check, and a type check (checks type casts)

Null Pointer Dereference

- use SAFE pointer

Array Bounds

- use SEQ pointer

Untrusted Access

- has special handling for variable number of arguments

Remaining Issues

Evaluation of new techniques is tedious

- must have a human determine if problem reported is an actual bug
- getting developers to fix the bug is another battle
- how can we determine if one bug detection system is better than another?
 - might analyze different languages
 - experiments performed on different benchmarks (version of the software make a different benchmark)
 - approach: people are starting to put together bug benchmarks

Static Analysis

- whole program versus partial program analysis
- quality of alias analysis affects quality number of false positives

Concepts

Approaches to bug detection

- augmenting the type system
- static analysis
- automated theorem proving
- dynamic analysis

Comparing bug detection techniques is tricky

- what is considered a real bug?
- how can we compare false positives with false negatives? how can we determine them at all

Next Time

Lecture

- This is it!
 - review of what we covered this quarter
 - how does it all fit together?
 - any requests?