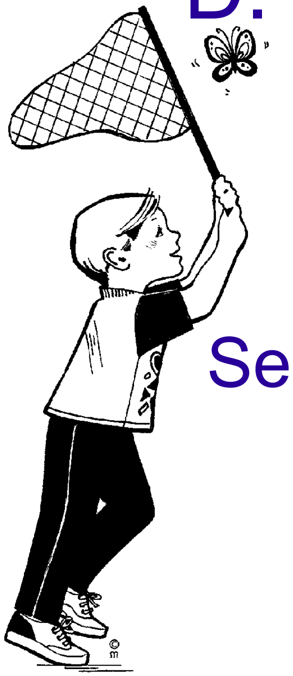
SmartFuzz

Dynamic Test Generation To Find Integer Bugs in x86 Binary Linux Programs

D. Molnar and X. C. Li and D. A. Wagner



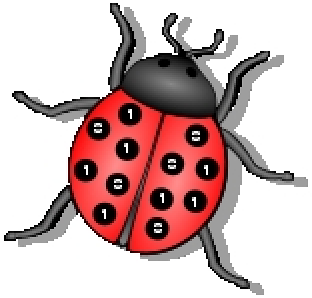
Seminar in Program Analysis for Cyber-Security
(236804) – Spring 2011
Karine Even - Technion

Outline

- Integer Bugs
- Motivation
- Test Generation for Finding Bugs
- SmartFuzz – Describing the method
- SmartFuzz – Algorithm
- Results
- Advantages/Disadvantages
- Conclusion & Discussion

Integer Bugs

- A common cause for serious security vulnerabilities
- Result from a mismatch between machine arithmetic and mathematical arithmetic
- Can lead to a buffer overflow: using too small/large number than expected



Integer Bugs - Types

- Overflow/Underflow
- Width Conversions
- Signed/Unsigned Conversion

```
char *badalloc(int sz, int n) {
  return (char *) malloc(sz * n);
}
void badcpy(Int16 n, char *p, char *q) {
  UInt32 m = n;
  memcpy(p, q, m);
}
void badcpy2(int n, char *p, char *q) {
  if (n > 800)
     return;
  memcpy(p, q, n);
}
```

Motivation

- Integer overflow bugs recently became the second most common bug type in security advisories from OS vendors
- Eliminates such bugs is important for improving software security
- Consider a large legacy code: we want to find and fix (manually) these bugs

Static Analysis

- Generate many false positives/negatives

- **Wrong values**: Since it is difficult to statically reason about integer values with sufficient precision (false-positive)
- **Wrong location**: Intent overflow semantics (false-positive)
- **Missed bug**: under approximation (false-negative)

Runtime Checks

- Inserts into the code, runtime checks for integer bugs and raises an exception if they occur

- Generate many false positives/negatives

- Benign and harmless overflows (false-positive)
- Intent overflow semantics (false-positive)
- Missed bug: imprecise checks (false-negative)

Uncover Bugs - Motivation

- ***False positives***: time-consuming
  (waste the programmer’s/end user’s time)
- Reducing the false positive rate is important
- How?
  - Automated process that checks these paths as part of the tool
  - Throws **no** exception
    $\Rightarrow$ Creates a report of all real-bugs instead
  - Done via <u>dynamic</u> test generation

Dynamic Test Generation

- A technique for generating test cases that expose specifically targeted behaviors of the program

- **A test case:** an input
- For multi-threaded programs: input+schedule

- Uses a symbolic execution of a test case to synthesize more test cases

Dynamic Test Generation

- Create an initial set of test cases
  - Randomly or by using a valid known one
- For each test case in the set:
  - Executes the program both concretely and symbolically $\Rightarrow$ extracts a path condition
  - Generates new test cases by solving symbolic constraints and add them to the set
- **Path condition**: a conjunction of all constraints over the symbolic values at each branch point of the concrete execution

Dynamic Test Generation

- **Constraints:** can represent
  - A specific path
  - A specific behavior
- For **example**: a constraints that satisfied once an assertion is violated
- Feeds to a solver: a path condition + an assertion violation constraint
- A satisfied assignment ≡ There is an input that violate a particular assertion and can cause the program to follow this path

SmartFuzz

- Performs symbolic execution and dynamic test generation on Linux x86 applications
- Discovers integer bugs in single threaded programs with untrusted data
- Reports real bugs: use common tools to check for buggy behavior (no false alarms)
- Reporting service: metafuzz.com, a web service for tracking test cases and bugs

SmartFuzz

- Constructs test cases that trigger:
  - Arithmetic overflows
  - Non-value-preserving width conversions
  - Dangerous signed/unsigned conversions

  (Via symbolic execution)

- **Signed/unsigned conversions**
  Type inference approach: detects values that are used as **both** signed and unsigned integers

SmartFuzz

- Online constraint generation: generates constraints while the program is running
- Using Valgrind intermediate representation: Translate the underlying x86 code on-the-fly into VEX

SmartFuzz

- Concertize the memory address before accessing the symbolic heap for each memory access instruction
- Stores symbolic information only for taint data (data that depends on untrusted inputs)

Algorithm

- Add test cases to a pool
  - Usually, starts with valid inputs
- Each test case in the pool receives a score
- **A score of a run:** according to the number of new basic blocks seen
- Iteratively creates more test cases
- Reports bugs via Metafuzz framework

Algorithm

- In each iteration of test generation:
  - **Chooses** a high scoring test case
  - **Executes** concretely and symbolically the program on that test case (Via the Valgrind binary analysis framework)
  - **Generates** a path condition (coverage&bug-seeking constraints)
  - **Solves** a path constraints via STP (a solver)

  - A solution: many new test cases
  - Reports a bug if a test case exhibits it
  - Add the pool test cases with no bug

Path Constraints

- **Constraints for coverage**
  - Add for each symbolic branch, a constraint that tries to force the program down a different path
  - A solution: a new “real” test case
- **Constraints for bug-seeking**
  - Add a constraint that is satisfied if an integer bug condition is satisfied
  - e.g., force an arithmetic calculation overflow

Path Constraints

- Add a constraint that is satisfied if:

- **Overflow/Underflow**: overflow/underflow occurs
- **Width Conversions**: source value can be outside the range of target value
- **Signed/Unsigned Conversions**: reconstructs signed/unsigned type information
  - Form a four-point lattice: {"Top", "Signed", "Unsigned", "Bottom"}
  - "Bottom": value has been used inconsistently as both a signed/unsigned

Results

- Compares:
  Dynamic test generation vs. black-box fuzz testing (different authors)

- ***metafuzz.com*** site has recorded more than 2,614 test runs, comprising 2,361,595 test cases

- Experiments: found approximately 77 total distinct bugs in 864 compute hours

Results

metafuzz.com

Current metafuzz stats, last updated Saturday 12th of March 2011 08:34:01 AM:
3196 runs in database, total of 2557818 test files created, with 6932 distinct bug stack hashes over all runs.

See the Premade VM Page for instructions on how to contribute your own results!

Current bugs with uploaded test cases (NOTE: this list is regenerated periodically and does not reflect an up to the minute list of test cases. See http://www.metafuzz.com/testcases for the most current list):

Run UUID	Seq. No	Stack Hash	Kind	Program	Status	Fuzz Type	Test Case	Submitter
730310	0	2774392970	UninitValue	test	Not Yet	catchconv	Download?	premade@metafuzz.com
730310	0	2762996045	UninitCondition	test	Not Yet	catchconv	Download?	premade@metafuzz.com
730310	0	2760969925	UninitCondition	test	Not Yet	catchconv	Download?	premade@metafuzz.com
730310	0	2756715073	UninitCondition	test	Not Yet	catchconv	Download?	premade@metafuzz.com
730310	0	2756107237	UninitCondition	test	Not Yet	catchconv	Download?	premade@metafuzz.com
730310	0	2755195483	UninitCondition	test	Not Yet	catchconv	Download?	premade@metafuzz.com
730310	0	2754739606	UninitCondition	test	Not Yet	catchconv	Download?	premade@metafuzz.com
152372	0	2752409568	UninitValue	test	Not Yet	catchconv	Download?	premade@metafuzz.com
693699	0	164958722	SyscallParam	mplayer	Not Yet	catchconv	Download?	premade@metafuzz.com
693699	0	3369391252	InvalidRead	mplayer	Not Yet	catchconv	Download?	premade@metafuzz.com
693699	0	2950937132	InvalidRead	mplayer	Not Yet	catchconv	Download?	premade@metafuzz.com
693699	0	678196151	UninitCondition	mplayer	Not Yet	catchconv	Download?	premade@metafuzz.com
693699	0	2909276546	UninitCondition	mplayer	Not Yet	catchconv	Download?	premade@metafuzz.com
693699	0	1370454836	InvalidRead	mplayer	Not Yet	catchconv	Download?	premade@metafuzz.com
433513	0	816552632	UninitCondition	a.out	Not Yet	catchconv	Download?	premade@metafuzz.com
433513	0	816552040	UninitCondition	a.out	Not Yet	catchconv	Download?	premade@metafuzz.com
433513	0	595185012	UninitCondition	a.out	Not Yet	catchconv	Download?	premade@metafuzz.com
433513	0	608968104	UninitCondition	a.out	Not Yet	catchconv	Download?	premade@metafuzz.com
433513	0	594955020	UninitCondition	a.out	Not Yet	catchconv	Download?	premade@metafuzz.com
433513	0	2096305996	UninitCondition	a.out	Not Yet	catchconv	Download?	premade@metafuzz.com
433513	0	2095900772	UninitValue	a.out	Not Yet	catchconv	Download?	premade@metafuzz.com

Results

- SmartFuzz finds bugs missed by zzuf (and vice versa)
- Interesting case: a program where SmartFuzz finds bugs but zzuf does not
- The zzuf tool: a simple and effective fuzz testing program
- **Fuzzing**: A method of finding software holes by feeding purposely invalid data as input to the program

Advantages

- Automated process(till the final report creation)

- Generate tests directly from shipping binaries
- No need or use of source code
- No need to modify the build process for a program under test

- Tests and analyze the whole-program:
  Can find bugs that arise due to interactions between the application and libraries it uses

Advantages

- Use different techniques for scaling dynamic test generation
(e.g., saves only necessary variables data)
- Address the problem of type inference for integer types in binary traces
- Efficient way for reporting bugs via Metafuzz

Disadvantages

- One thread - no concurrency
  Cannot test multi-threaded and network-facing programs
- Uses *Valgrind* binary analysis framework
  Results in long traces and correspondingly longer symbolic formulas
- Online constraint generation
  Instead of offline constraint generation - that is better

Disadvantages

- Cannot generate any test case
  The input of test cases is limited by size
- Repeats many sub-expression optimizations
  Sends to the solver an expression that is “close” as possible to the intermediate representation
- Needs a powerful solver
  Not all expression are simple/easy to solve

Conclusions & Discussion

- SmartFuzz: cannot guarantee full coverage
  $\Rightarrow$ Can use more than one testing tool

- ***metafuzz.com***: presents a long list of bugs
  So what's next?

Any Questions?

Thank You!