BALLISTA

Automated Robustness Testing of Off-the-Shelf Software Components

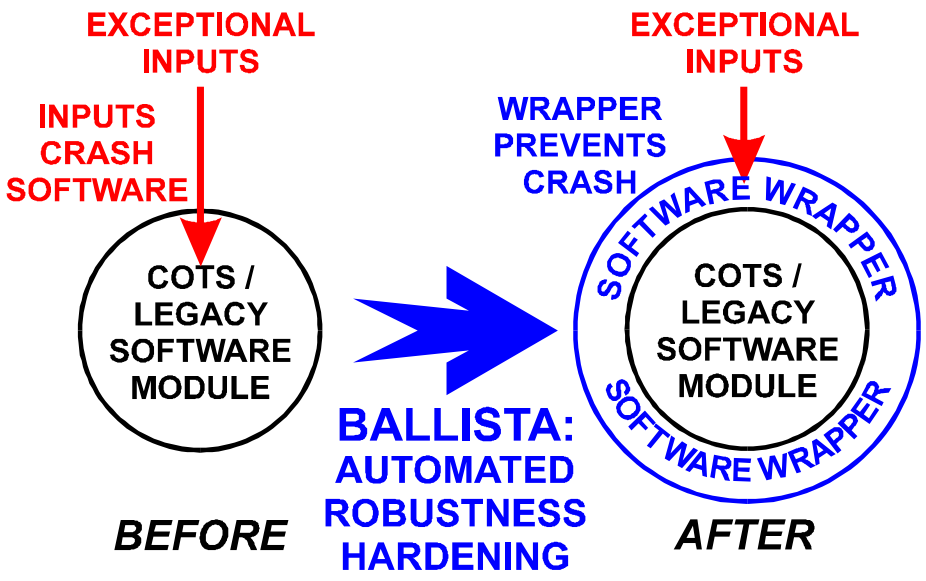
Nathan Kropp • Phil Koopman • Dan Siewiorek
Carnegie Mellon University
http://www.ices.cmu.edu/ballista

Institute for Complex Engineered Systems

Carnegie Mellon

DARPA

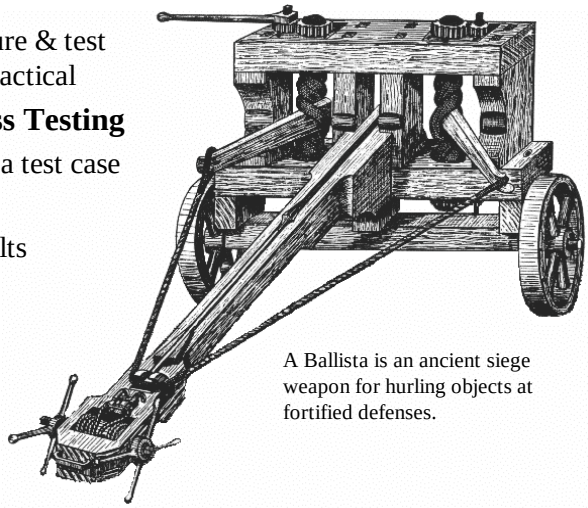
The Big Picture: Ballista Project



2

Overview: Ballista Robustness Testing

- **System Robustness**
  - Must be able to measure & test before hardening is practical
- **Automated Robustness Testing**
  - Operating Systems as a test case
  - Need scalability
  - Full-scale testing results
- **Conclusions**



A Ballista is an ancient siege weapon for hurling objects at fortified defenses.

BALLISTA

3

System Robustness

A) Graceful behavior in the presence of exceptional conditions

- Unexpected operating conditions
- Activation of latent design defects
- Focus of the current research

B) Operation under extraordinary loads

- The *other* half of robustness -- but not covered in this work

- **Current test case -- Operating Systems (POSIX API)**
  - Goal -- metric for comparative evaluation of OS robustness
  - If a mature OS isn't "bullet-proof", what hope is there for application software?

4

Measuring Robustness

- **Software testing heritage:**
  - "Dirty" test cases -- see if correct error response is generated
    - Can significantly out-number "clean" test cases (4:1 or 5:1) Þ *expensive!*
- **Fault tolerance heritage: fault injection**
  - Insert an intentional defect and observe how gracefully the system responds
    - Potentially automated (potentially *cheap*)
  - But, there are challenges
    - Creating a non-intrusive injection mechanism
    - Combinational explosion of potential interactions
    - Repeatability / determinism
    - Portability to compare systems / requirement for special hardware

5

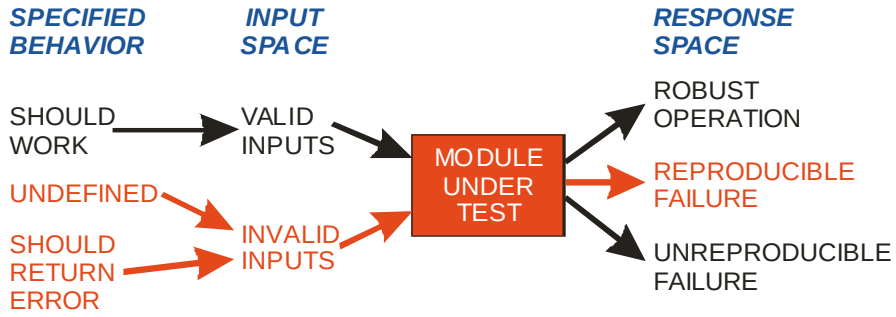
Ballista Automated Testing Goals

- **No functional specification**
  - Generically applicable to modules having argument lists
  - No source code, no reverse compilation, … no "peeking"
- **Highly scalable**
  - Automated operation from test case generation to hardening
  - Effort to create tests sub-linear with number of functions tested
- **Repeatable results**
  - Robustness failures repeatable on demand
  - Single-function-call fault model
    - Enables creation of very simple "bug report" code
    - Makes it possible to create reasonably simple wrappers
    - Only addresses a subset of problems (but, a big subset?)

6

Ballista Software Testing Heritage

- **SW Testing requires:**
  - Test case
  - Module under test
  - *Oracle* (a "specification")
- **Ballista uses:**
  - "Bad" value combinations
  - Module under Test
  - *Watchdog timer/core dumps*



7

Ballista Fault Injection Heritage

Name	Method	Level	Repeatability
FIAT	Binary Image Changes	Low	High
FERRARI	Software Traps	Low	High
Crashme	Jump to Random Data	Low	Low
FTAPE	Memory/Register Alteration	Low	Medium
FAUST	Source Code Alteration	Middle	High
CMU-Crashme	Random Calls and Random Parameters	High	Low
Fuzz	Middleware/Drivers	High	Medium
Ballista	Specific Calls with Specific Parameters	High	High

8

Ballista: "High Level" + "Repeatable"

- **Example test:**

```
read(bad_fd, NULL_buffer, neg_one_length);
```

- **High level fault injection**
  - Send exceptional values into a component set through the API
- **Repeatable: single function call for each test:**
  - System state initialized & cleaned up for each single-call test
  - Combinations of valid and invalid parameters tried in turn
  - A "simplistic" model, but it does in fact work...
    - Crashes several commercial operating systems

9

CRASH Severity Scale

- **Catastrophic**
  - Test computer crashes (both Benchmark and Starter abort or hang)
- **Restart**
  - Benchmark process hangs, requiring restart
- **Abort**
  - Benchmark process aborts (*e.g.*, "core dump")
- **Silent**
  - No error code generated, when one should have been (*e.g.*, de-referencing null pointer produces no error)
- **Hindering**
  - Incorrect error code generated

10

A Challenge: Scalability

- **Precursors to Ballista achieved high level repeatability**
  - But, they didn't scale without significant effort
- **Scaffolding**
  - Software testing in general requires scaffolding to be erected for every function to be tested
  - But, this makes it expensive to test a significant API
- **Specification/oracle creation**
  - Software testing in general requires a specification for each function
  - But, specification (or even source code) may be unavailable

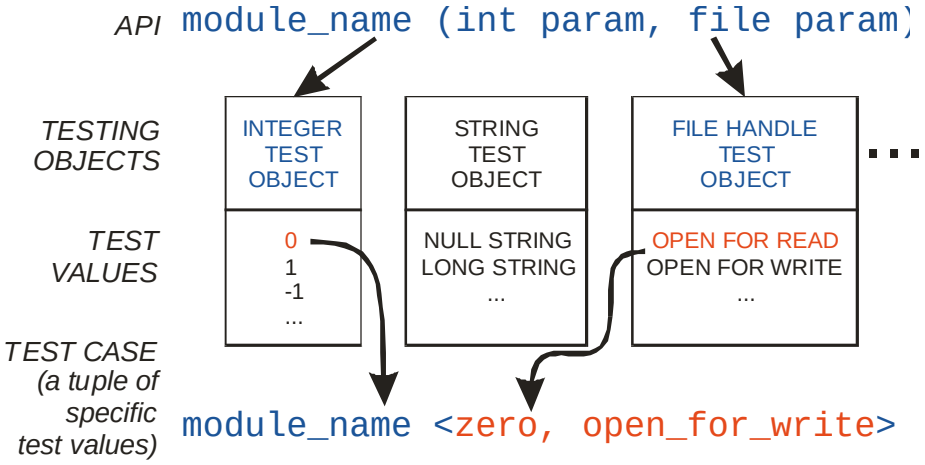
11

Scalable Test Generation -- Scaffolding

- **Problem 1: Avoid per-function work for test scaffolding**
  - Scaffolding required to set appropriate state for each function
  - ***Insight:*** Fewer data types than functions
  - ***Solution:*** Encapsulate scaffolding in data types alone -- no per-function scaffolding.
- **Each test value instance has a constructor & destructor**
  - Constructor creates state required for a particular test value
    - *e.g.*, create a file, put data in it, open it for read, return that file handle
  - Destructor cleans up any remaining state after the test
    - *e.g.*, close & delete a file that had been created by constructor
  - Scaffolding based on data type *regardless of function*

12

Ballista: Scalable Test Generation



- **Only 20 data types for 233 POSIX function calls**

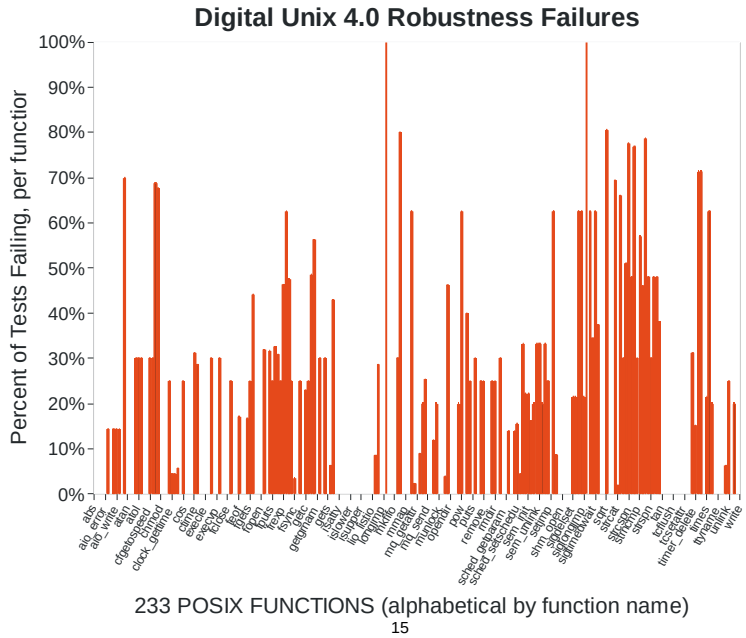
13

Scalable Test Generation -- An Oracle

- **Problem 2: Avoid per-test work to determine pass *vs.* fail**
  - Understanding of functionality required for "pass" vs. "fail"
  - ***Insight:*** Ignore functionality -- use "doesn't crash; doesn't hang"
  - ***Solution:*** Test them all and let the watchdog timer/core dumps sort them out.
    - Test only Catastrophic - Restart - Abort failures (for now, anyway)
    - Ignore pass/fail in terms of return code; just look for robustness failures
- **Example:**
  - `read()` succeeds but `write()` (hopefully) returns an error when accessing a read-only file
  - But we can ignore any return codes and just look for an Abort or Restart in either case

14

So, Did It Find Anything?



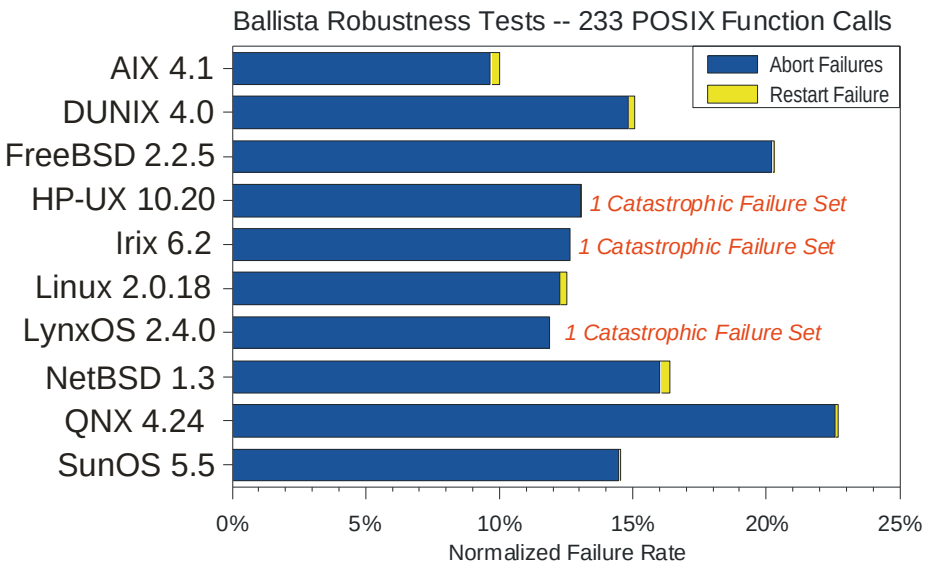
15

What We Measured

- **233 POSIX Calls (including real-time extensions)**
  - That take at least one parameter
  - That don't intentionally hang or generate signals
  - 92,658 tests per OS if all 233 functions are supported
- **"Single-number" summary metric**
  - Failure rate computed for each function and then averaged
    - Should weight by usage frequency for any particular application environment
  - *Gives a portable comparative metric for robustness(!)*

16

Was It Portable?



17

Was It Repeatable + Scalable?

- **http://www.ices.cmu.edu/ballista -- Digital Unix demo**
  - Generates single-test "bug report" programs
  - Reproduces results by executing a program from the command line
- **Yes, it's scalable**
  - Generates ~100,000 test cases for 233 functions
  - ~2000 lines of "easy" C code to test 20 data types
    - (plus Ballista test harness)
  - A reasonable amount of system state is tested without per-test scaffolding
    - e.g., files, memory arrays, data structures
    - The encapsulation of system state within test cases really worked
  - Work on a simulation backplane API for looks promising

18

Conclusions

- **Ballista testing quantifies one aspect of robustness**
  - Scalable -- base scaffolding on data types, not functions
  - Repeatable -- single-call approach is simple, but effective
  - Portable -- use API for fault injection
- **But, it is only a start**
  - Tests one aspect of system robustness
  - Currently uses only heuristic tests (want broader coverage in future)

19

Anybody can build a system that works when it works, but it's how it works when it doesn't work that counts.