Software Robustness Testing Service

http://www.ices.cmu.edu/ballista

John P. DeVale
devale@cmu.edu - (412) 268-4264 - http://www.ece.cmu.edu/~jdevale
Overview: Ballista Automated Robustness Testing

- **System Robustness**
  - Motivation
  - Ballista automatic robustness testing tool

- **OS Robustness Testing**
  - Raw results for 15 Operating Systems

- **Testing Service**

- **Conclusions**

A Ballista is an ancient siege weapon for hurling objects at fortified defenses.
System Robustness
Ariane 5 Flight 501 Robustness Failure

- June, 1996 loss of inaugural flight
  - Lost $400 million scientific payload (the rocket was extra)

- Efforts to reduce system costs led to the failure
  - Re-use of Inertial Reference System software from Ariane 4
  - Improperly handled exception caused by variable overflow during new flight profile (that wasn’t simulated because of cost/schedule)
    - 64-bit float converted to 16-bit int *assumed* not to overflow
    - Exception caused dual hardware shutdown (because it was assumed software doesn’t fail)

- What really happened here?
  - **The narrow view:** it was a software bug -- fix it
    - Things like this have been happening for decades -- Apollo 11 LEM computer crashed during lunar descent
  - **The broad view:** the loss was caused by a lack of system robustness in an exceptional (unanticipated) situation

- Our research goal: *improved system robustness*
System Robustness -- Improves Dependability

◆ **Graceful behavior in the presence of exceptional conditions**
  - Unexpected operating conditions
  - Activation of latent design defects

◆ **Robustness definition also includes operation in overloads**
  - Not in current research, but is set as an eventual goal
  - We conjecture overload robustness also hinges on exception handling

◆ **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?
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*

---

**Ballista combines:**
- Domain testing ideas / Syntax testing ideas
- In general, “dirty” testing
# Ballista Fault Injection Heritage

<table>
<thead>
<tr>
<th>Name</th>
<th>Method</th>
<th>Level</th>
<th>Repeatability</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIAT</td>
<td>Binary Image Changes</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>FERRARI</td>
<td>Software Traps</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Crashme</td>
<td>Jump to Random Data</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>FTAPE</td>
<td>Memory/Register Alteration</td>
<td>Low</td>
<td>Medium</td>
</tr>
<tr>
<td>FAUST</td>
<td>Source Code Alteration</td>
<td>Middle</td>
<td>High</td>
</tr>
<tr>
<td>CMU-Crashme</td>
<td>Random Calls and Random Parameters</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Fuzz</td>
<td>Middleware/Drivers</td>
<td>High</td>
<td>Medium</td>
</tr>
<tr>
<td>Ballista</td>
<td>Specific Calls with Specific Parameters</td>
<td>High</td>
<td>High</td>
</tr>
</tbody>
</table>
CRASH Severity Scale

- **Catastrophic**
  - Test computer crashes (both Benchmark and Starter abort or hang)
  - Irix 6.2: `munmap( malloc((1<<30)+1), ((1<<31)-1))` ;

- **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
**Ballista: Scalable Test Generation**

**API**

`inttrap(double a, double b, int N)`

**Testing Objects**

- Double Precision Floating Point Test Object
  - ZERO
  - ONE
  - NEGONE
  - TWO
  - PI
  - PIBYTWO
  - TWOPI
  - E
  - DBLMAX
  - DBLMIN
  - SMALLNOTZERO
  - NEGSMLALLNOTZERO

- Double Precision Floating Point Test Object
  - ZERO
  - ONE
  - NEGONE
  - TWO
  - PI
  - PIBYTWO
  - TWOPI
  - E
  - DBLMAX
  - DBLMIN
  - SMALLNOTZERO
  - NEGSMLALLNOTZERO

- Integer Value Test Object
  - MAXINT
  - MININT
  - ZERO
  - ONE
  - NEGONE
  - 2
  - 4
  - 8
  - 16
  - 32
  - 64
  - 1K
  - 64K
  - ...

**Test Values**

- DOUBLE PRECISION FLOATING POINT TEST OBJECT
- DOUBLE PRECISION FLOATING POINT TEST OBJECT
- INTEGER VALUE TEST OBJECT

**Test Case**

`inttrap(ONE, DBLMAX, 64K)`
## Test Value Inheritance

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Generic String</td>
<td>BIG STRING</td>
<td>STRINGLEN1</td>
<td>ALLASCII</td>
<td>NONPRINTABLE</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generic Pointer</td>
<td>NULL</td>
<td>DELETED</td>
<td>1K</td>
<td>PAGESIZE</td>
<td>MAXSIZE</td>
<td>SIZE1</td>
<td>INVALID</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Date string inherits test cases from all parents
Ballista: “High Level” + “Repeatable”

- High level testing is done using API to perform fault injection
  - Send exceptional values into a system through the API
    - Requires no modification to code -- only linkable object files needed
    - Can be used with any function that takes a parameter list
  - Direct testing instead of middleware injection simplifies usage

- Each test is a specific function call with a specific set of parameters
  - 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...

- Early results were encouraging:
  - Found a significant percentage of functions with robustness failures
  - Crashed systems from user mode
OS Robustness Testing
Comparing Fifteen Operating Systems

Ballista Robustness Tests for 233 Posix Function Calls

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Normalized Failure Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX 4.1</td>
<td>10%</td>
</tr>
<tr>
<td>FreeBSD 2.2.5</td>
<td>15%</td>
</tr>
<tr>
<td>HP-UX 9.05</td>
<td>10%</td>
</tr>
<tr>
<td>HP-UX 10.20</td>
<td>10%</td>
</tr>
<tr>
<td>Irix 5.3</td>
<td>15%</td>
</tr>
<tr>
<td>Irix 6.2</td>
<td>15%</td>
</tr>
<tr>
<td>Linux 2.0.18</td>
<td>15%</td>
</tr>
<tr>
<td>LynxOS 2.4.0</td>
<td>15%</td>
</tr>
<tr>
<td>NetBSD 1.3</td>
<td>15%</td>
</tr>
<tr>
<td>OSF 1.3.2</td>
<td>15%</td>
</tr>
<tr>
<td>OSF 1.4.0</td>
<td>15%</td>
</tr>
<tr>
<td>QNX 4.22</td>
<td>20%</td>
</tr>
<tr>
<td>QNX 4.24</td>
<td>15%</td>
</tr>
<tr>
<td>SunOS 4.1.3</td>
<td>15%</td>
</tr>
<tr>
<td>SunOS 5.5</td>
<td>15%</td>
</tr>
</tbody>
</table>

Abort Failures: 1 Catastrophic
Restart Failure: 1 Catastrophic

2 Catastrophics
C Library Is A Potential Robustness Bottleneck

Portions of Failure Rates Due To System/C-Library

- AIX 4.1
- Free BSD 2.2.5
- HP-UX 9.05
- HP-UX 10.20
- Irix 5.3
- Irix 6.2
- Linux 2.0.18
- LynxOS 2.4.0
- NetBSD 1.3
- OSF 1.3.2
- OSF 1.4.0
- QNX 4.22
- QNX 4.24
- SunOS 4.1.3
- SunOS 5.5

Normalized Failure Rate

0% 5% 10% 15% 20% 25%

1 Catastrophic
2 Catastrophics

System Calls
C Library
Common Failure Sources

- Based on correlation of failures to data values, not traced to causality in code

- Associated with a robustness failure were:
  - 94.0% of invalid file pointers (excluding NULL)
  - 82.5% of NULL file pointers
  - 49.8% of invalid buffer pointers (excluding NULL)
  - 46.0% of NULL buffer pointers
  - 44.3% of MININT integer values
  - 36.3% of MAXINT integer values
Testing Service
Robustness Testing Service

◆ Ballista Server
  • Selects tests
  • Performs pattern Analysis
  • Generates “bug reports”
  • Never sees user’s code

◆ Ballista Client
  • Links to user’s SW under test
  • Can “teach” new data types to server (defn. language)
Ballista Capability Summary

◆ Automated testing of software components
  • Generically applicable to modules having parameter lists

◆ Minimal knowledge of component
  • Interface specification is typically available (data types)
  • No source code, no reverse compilation, no functional specification

◆ Highly scalable
  • Effort to create tests sub-linear with number of functions tested
  • No per-function test scaffolding

◆ Repeatable results
  • Robustness failures that are identified are repeatable on demand
  • Single-function-call failure generation
    – 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?)
Conclusions

◆ Ballista robustness testing approach
  • Scalable, portable, reproducible
  • Can include considerable state information (although that’s not obvious)

◆ Also applied to DoD HLA/RTI simulation backplane
  • C++, call-backs, client/server, throws signals for exception handling
  • Specifically written for robustness; has lower failure rates than OS code

◆ Internet-based testing service available
http://www.ices.cmu.edu/ballista