The Dimensionality Model for Characterizing Software Robustness

Jiantao Pan
ECE Department
Carnegie Mellon University
jpan@cmu.edu

Institute for Complex Engineered Systems
ECE
Carnegie Mellon
DARPA
Introduction

◆ Software robustness matters

◆ The Ballista project
  • Testing & hardening COTS/legacy software modules
  • 1.1 million data points on 15 POSIX OSes
  • The Dimensionality Model: for finding failure patterns

◆ Potential uses of the model:
  • Guiding robustness testing
  • Guiding robustness failure protection
The Dimensionality Model

◆ Definitions:
  • Parameter dimensionality: number of arguments accepted by a software module
  • Robustness failure dimensionality: number of parameters contributing to the failure

◆ Examples
  • 3-D parameter dimensionality:
    - read(file_descriptor, buffer, bytes_to_read)
  
  • 1-D failure: read(NULL, —, —)
    - NULL file_descriptor
  
  • 2-D failure: read(—, 16K, 64K)
    - buffer smaller than bytes_to_read
Low Dimensionality Failures Prevail

- If we can eliminate 1-D failures, average failure rate drops from:
  - 15.2% \(\downarrow\) 2.8%

Impact on Robustness after Guarding against 1-Dimensional Failures

<table>
<thead>
<tr>
<th>15 POSIX OS Versions from Ten Vendors</th>
<th>Not 1-D Failures</th>
<th>1-D Failures</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX 4.1</td>
<td>9.99%</td>
<td>9.99%</td>
</tr>
<tr>
<td>HP-UX A.09.05</td>
<td>11.39%</td>
<td>11.39%</td>
</tr>
<tr>
<td>LynxOS 2.4.0</td>
<td>11.89%</td>
<td>11.89%</td>
</tr>
<tr>
<td>LINUX 2.0.18</td>
<td>12.54%</td>
<td>12.54%</td>
</tr>
<tr>
<td>IRIX 6.2</td>
<td>12.62%</td>
<td>12.62%</td>
</tr>
<tr>
<td>HP-UX B.10.20</td>
<td>13.05%</td>
<td>13.05%</td>
</tr>
<tr>
<td>IRIX 5.3</td>
<td>14.45%</td>
<td>14.45%</td>
</tr>
<tr>
<td>SunOS 5.5</td>
<td>14.55%</td>
<td>14.55%</td>
</tr>
<tr>
<td>Digital Unix 4.0</td>
<td>15.07%</td>
<td>15.07%</td>
</tr>
<tr>
<td>Digital Unix 3.2</td>
<td>15.63%</td>
<td>15.63%</td>
</tr>
<tr>
<td>SunOS 4.1.3</td>
<td>15.84%</td>
<td>15.84%</td>
</tr>
<tr>
<td>NetBSD 1.3</td>
<td>16.39%</td>
<td>16.39%</td>
</tr>
<tr>
<td>FreeBSD 2.2</td>
<td>20.28%</td>
<td>20.28%</td>
</tr>
<tr>
<td>QNX 4.22</td>
<td>21.00%</td>
<td>21.00%</td>
</tr>
<tr>
<td>QNX 4.24</td>
<td>22.69%</td>
<td>22.69%</td>
</tr>
</tbody>
</table>
Low Dimensionality is Common

- All operating systems tested exhibit similar phenomena
  - Average 82% (standard deviation 3.24%) failure rate is attributed to 1-D
Conclusions

- Most failures we have seen are 1-Dimensional
  - Prevalent across a wide range of POSIX OS APIs
  - Confirms hypotheses of testers (AETG, etc.)

- The Dimensionality Model
  - Analysis method for API level robustness failures
    - Generic analysis method for other applications?
  - Might be used to guide automated testing
    - Potentially cost-effective
  - Good for robustness hardening?
    - Automated robustness hardening guided by dimensionality analysis
Testing Methodology [backup]

- **Feed combinations of valid and invalid inputs to POSIX calls**
  - Assume no access to source code (black box)
  - Single call per test for simplicity, ignore interactions and timing
  - Testing method intended to work on other Commercial Off-The-Shelf (COTS) software

**Function call:** `read(file_descriptor, buffer, bytes_to_read)`

**Parameter value data base**

- **File descriptor (13 cases)**
  - valid file, closed
  - valid file, read only
  - valid file, read-write
  - empty file

- **Buffer pointer (15 cases)**
  - 1
  - 4K
  - freed buffer
  - NULL
  - very large buffer

- **Integer (16 cases)**
  - 0
  - 1
  - 4K
  - -1
  - -64K

3120 combinations
3120 tests

Example test instances: `read(-1, very_large_buf, 4K)`
Patterns of Testing Result [backup]

◆ `fprintf(File_Pointer, STRing)` in HP-UX

◆ 1-D failures:
  - They form a line in a 2-D function (function that parameter dimensionality=2)
  - They form a hyperplane in a n-D function

---

**Results for fprintf() [HP-UX B.10.20]**

- **Pass** (Success, Error code)
- **Robustness Failure** (Catastrophic, Restart, Abort)