Bear: A Framework for Understanding Application Sensitivity to OS (Mis)Behavior

Ruimin Sun

Florida Institute of Cyber Security
University of Florida
Motivation

- Unpredictabilities at the OS level are **more common than once thought**.

- Developers are not equipped to write robust applications facing unpredictable or even adversarial OSes.
Source of Unpredictabilities

- OS handles network events and protocols differently
- Subtle and undocumented differences in common APIs across different platforms
- OS changes over time
- A buggy or malicious OS
Fuzz Testing

- An effective way to discover coding errors and security loopholes
- To test applications against invalid, unexpected, or random data inputs.
- Trinity, KLEE, BALLISTA...
Fault Injection

- An important method for generating test cases in fuzz testing
- Example
  - Memory, CPU, and communication faults.
  - Hardware-induced software errors and kernel software faults
  - library-calls error injection
Failures Oblivious Computing

- Allows a system or program to continue execution in spite of errors
- Example
  - Rinard et al. [42] a C compiler to insert checks to dynamically detect invalid memory accesses
  - discard invalid writes
  - return manufactured values to for invalid reads
A Linux-based framework for statistical analysis of application sensitivity to OS unpredictability.

- Analyzes a program using a set of unpredictability strategies on a set of commonly used systems calls
- Discovers the most sensitive system calls/strategies
Bear’s Goal

- Discover bugs that hard to be reproduced
- Target end-to-end checks, time-consuming tests and verification procedures
- Equip developers to design more resilient applications
## Perturbation Strategies

<table>
<thead>
<tr>
<th>Category</th>
<th>System Call Example</th>
<th>Strategies</th>
<th>Common Related Bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Management</td>
<td>sys_unmap</td>
<td>Fail to deallocate system call (failMem)</td>
<td>Memory leak</td>
</tr>
<tr>
<td></td>
<td>sys_mmap</td>
<td>Empty buffer in memory system call (nullMem)</td>
<td>Null dereferencing</td>
</tr>
<tr>
<td>Signal Control</td>
<td>sys_mlock</td>
<td>Failure to lock related system call (failLock)</td>
<td>Synchronization error</td>
</tr>
<tr>
<td></td>
<td>sys_kill</td>
<td>Failure to signal control system call (failSig)</td>
<td>Signal delivery error</td>
</tr>
<tr>
<td>File Operation</td>
<td>sys_read</td>
<td>Different data type to buffer parameter (diffType)</td>
<td>Value outside domain</td>
</tr>
<tr>
<td></td>
<td>sys_write</td>
<td>Injection of bytes to system call with a buffer(bufOf)</td>
<td>Buffer overflow</td>
</tr>
<tr>
<td>Network Communication</td>
<td>sys_access</td>
<td>Failure to access system call (failAcc)</td>
<td>Dealing with volatile objects</td>
</tr>
<tr>
<td></td>
<td>sys_sendto</td>
<td>Reduction of buffer size/length parameter (redLen)</td>
<td>Buffer return not checked</td>
</tr>
<tr>
<td>Process Scheduling</td>
<td>sys_mq_notify</td>
<td>Fail to notify system call (failNoti)</td>
<td>Naked notify in method</td>
</tr>
<tr>
<td></td>
<td>sys_setuid</td>
<td>Fail to change user id (chUid)</td>
<td>Privilege degradation</td>
</tr>
</tbody>
</table>
Research questions:

1. which system calls are the most sensitive to OS unpredictability and by what degree?

2. which strategies cause the most impact in program execution and by what degree?

3. do program type and execution workloads affect the strategy impact or system call sensitivity?

Fig. 4: Statistical tests used in this study.
There is an correlation between a program execution outcome and a perturbation system call, and likewise a perturbation strategy.

Software samples
53 CPU bound programs, and 47 I/O bound programs from GNU projects, SPEC CPU2006 and Phoronix-test-suite.
The impact of perturbation strategies in predicting abnormal program outcome (IO bound).

**Normal** referred to a correct execution or a graceful exit of a program and result
**Abnormal** referred to all the other results.

**Odds ratio** shows how more or less likely a strategy is to cause an abnormal program outcome compared to the reference strategy.

The impact of perturbation strategies in predicting abnormal program outcome (CPU bound).
Impact of System Call

IO-bound

CPU-bound
Findings

- The impact of *buffer overflow* and *wrong parameter type* doubled when workload is heavy.
- Network related system calls didn’t show high impact.
- *Null dereferencing* is a severe problem and almost the hardest to debug too.
  - *failure-oblivious computing* can be a promising way for memory errors.
Findings

- Generic system calls are more sensitive than specialized system calls
  - `write` and `sendto` can both be used to send data through a socket, but the sensitivity of `write` is twice that of `sendto`.
- System calls with an array parameter of a buffer are more sensitive to perturbations than those having a struct parameter
  - `read` v.s. `readv`
- The fewer parameters a system call has, the more sensitive it is
Questions?

Ruimin Sun
gracesrm@ufl.edu