Understanding Application Sensitivity to OS (Mis)Behavior

Ruimin Sun, Andrew Lee, Aokun Chen, Matt Bishop, Donald E. Porter, Daniela Oliveira

Electrical and Computer Engineering, University of Florida

Email: gracesrm@ufl.edu

Motivation

Unpredictability at the OS level is **more common than once thought**.

- ❑ Different OS handles network events and protocols differently
- ❑ Subtle differences in common APIs across different platforms
- ❑ OS changes over time
- ❑ OS has bugs, vulnerabilities

Developers are not equipped to write robust applications facing unpredictable or even adversarial Operating Systems.

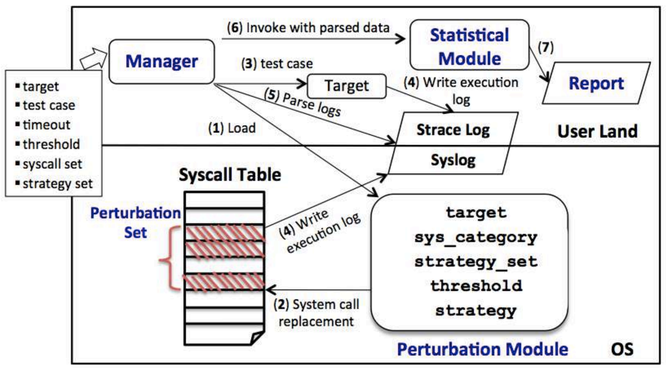
Theory and Background

- ❑ Fuzz Testing
  - ❑ An effective way to discover coding errors and security loopholes
  - ❑ To test applications against invalid, unexpected, or random data inputs.
- ❑ 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

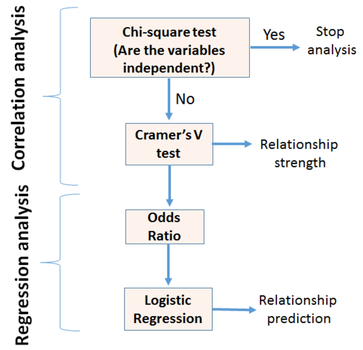
Design Goal

- ❑ 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

Architecture



Statistical Method



Experiment

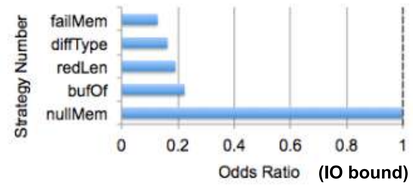
Strategies	Common Related Bugs
Fail to deallocate system call (failMem)	Memory leak
Empty buffer in memory system call (nullMem)	Null dereferencing
Failure to lock related system call (failLock)	Synchronization error
Failure to signal control system call (failSig)	Signal delivery error
Different data type to buffer parameter (diffType)	Value outside domain
Injection of bytes to system call with a buffer(bufOf)	Buffer overflow
Failure to access system call (failAcc)	Dealing with volatile objects
Reduction of buffer size/length parameter (redLen)	Buffer return not checked
Fail to notify system call (failNoti)	Naked notify in method
Fail to change user id (chUid)	Privilege degradation

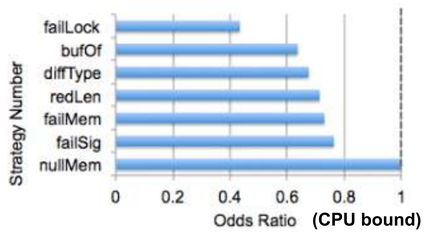
Research questions:

1. which system calls are the most sensitive to OS unpredictability and by what degree?
2. which strategies cause the most impact?
3. do program type and execution workloads affect?

Results and Analysis

Impact of perturbation **strategies**





Tested on 53 CPU bound programs, and 47 I/O bound programs from GNU projects, SPEC CPU2006 and Phoronix-test-suite.

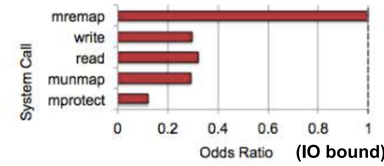
Normal referred to a correct execution or a graceful exit of a program and result **Abnormal** referred to all the other results.

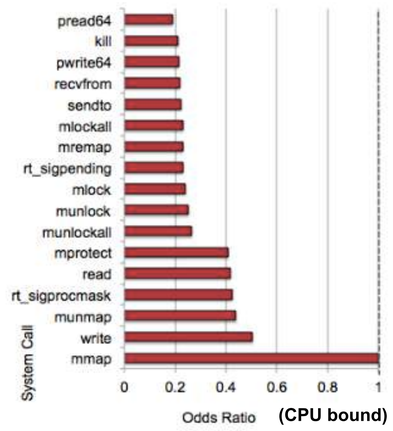
Odds ratio shows how more or less likely a strategy is to cause an abnormal program outcome compared to the **reference strategy.**

- ❑ The impact of *buffer overflow* and *wrong parameter type* doubled when workload is heavy.
- ❑ *Null dereferencing* is a severe problem and almost the hardest to debug too.
  - ❑ failure-oblivious computing can be a promising way for memory errors.

Results and Analysis (Cont'd)

Impact of perturbation **system calls**





- ❑ Network related system calls are not highly impactful.
- ❑ Generic system calls are more sensitive than specialized system calls
  - ❑ *write* v.s. *sendto*
- ❑ System calls with an array parameter of a buffer are more sensitive than those with a *struct* parameter
  - ❑ *read* and *write* v.s. *readv* and *writev*
- ❑ The fewer parameters a system call has, the more sensitive it is
  - ❑ *access* v.s. *faccessat*

Conclusion

- ❑ Bear - a framework for statistical analysis on program sensitivity to OS unpredictability
- ❑ Bear's evaluation produced important insights for developers.
- ❑ More: Ruimin Sun, Andrew Lee, Aokun Chen, Don Porter, Matt Bishop and Daniela Oliveira. Bear: A Framework for Understanding Application Sensitivity to OS (Mis)Behavior. ISSRE 2016. Ottawa, Canada, Oct. 23-27.