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

Current Techniques

- **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…

Current Techniques

- **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

Current Techniques

- **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

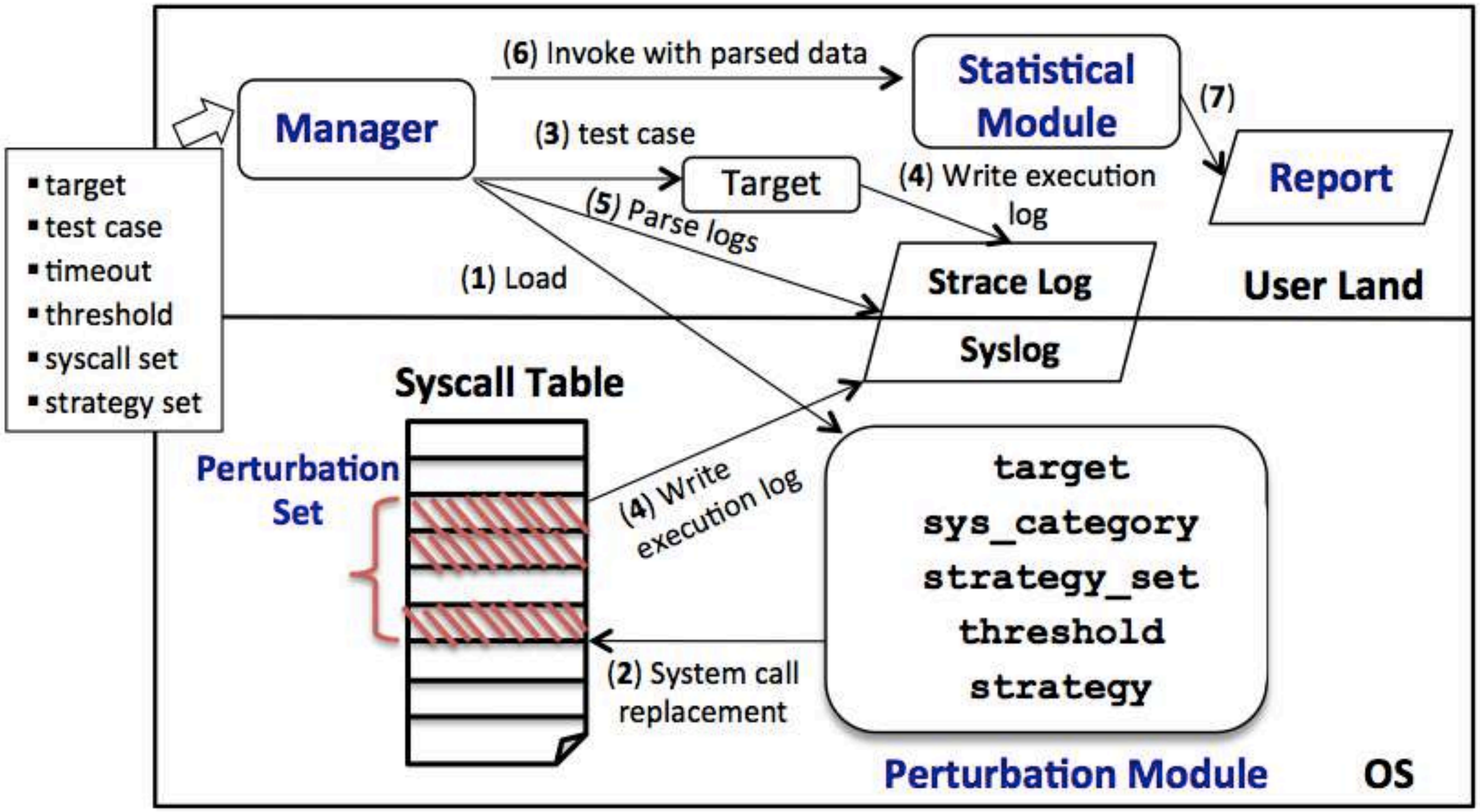
Bear

- 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

Bear Architecture



Bear's Architecture

Perturbation Strategies

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

Statistical Methods

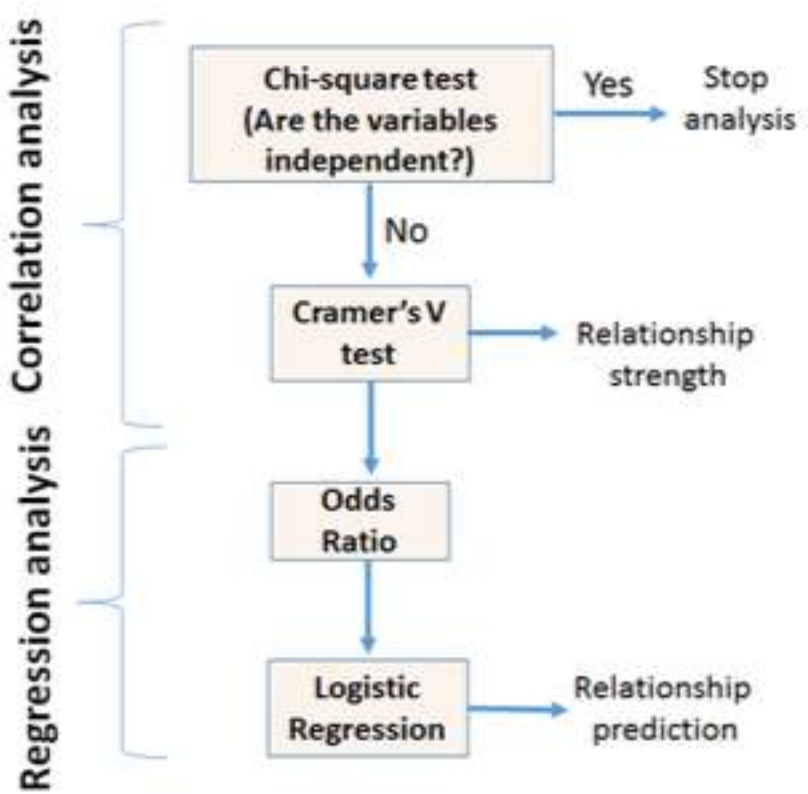


Fig. 4: Statistical tests used in this study.

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?

Evaluation Results

	X-squared value	df	p-value
All programs	262.39	20	<0.0001
CPU bound	48.732	20	0.0003355
IO bound	486.21	20	<0.0001

TABLE II: Chi-square result for *system call* and program execution outcome.

	X-squared value	df	p-value
All programs	66.428	9	<0.0001
CPU bound	42.667	9	<0.0001
IO bound	112.02	9	0.0017

TABLE III: Chi-square result for *strategy* and program execution outcome.

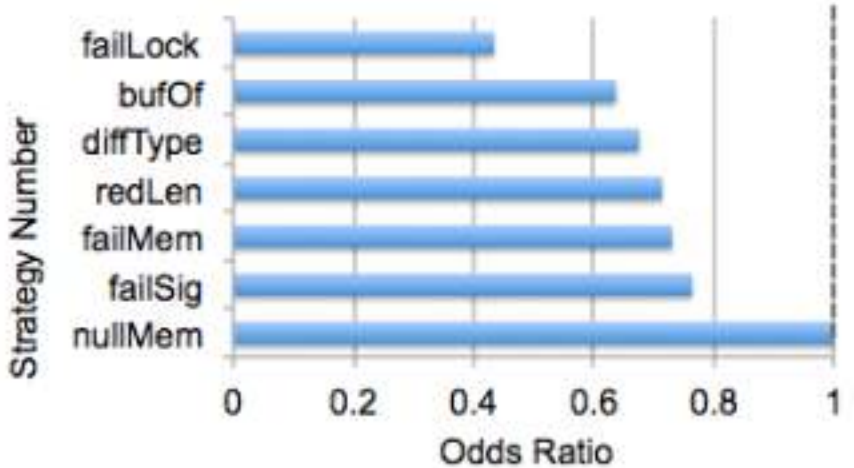
Software samples

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

There is an correlation between a program execution outcome and a perturbation system call, and likewise a perturbation strategy.

Impact of Strategy

The impact of perturbation strategies in predicting ***abnormal*** program outcome (**IO** bound).



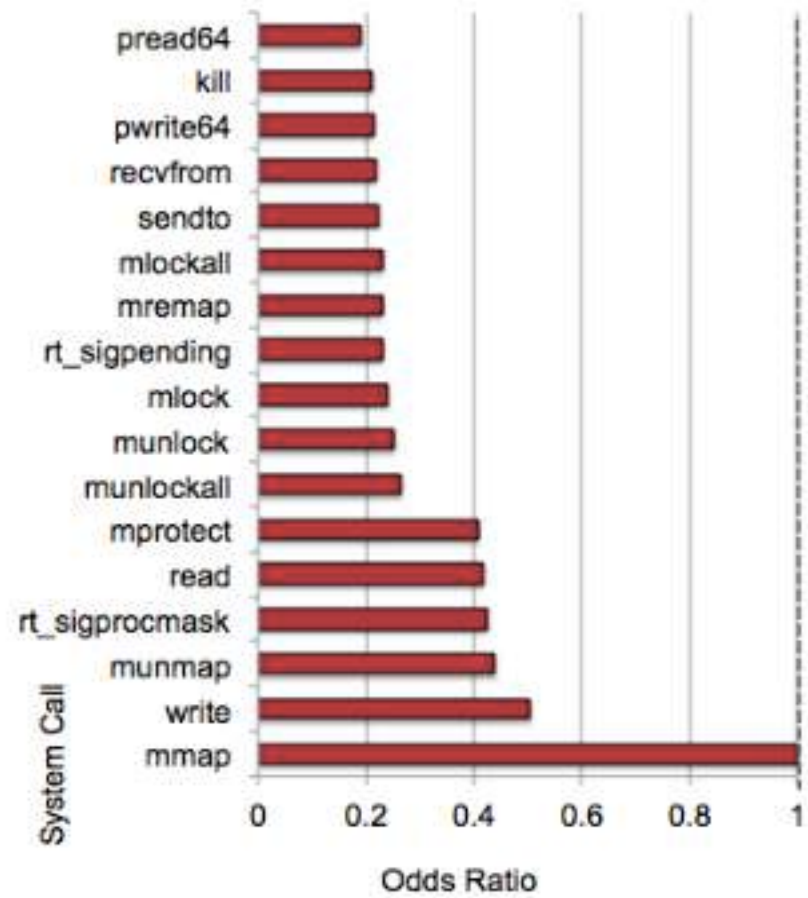
The impact of perturbation strategies in predicting ***abnormal*** program outcome (**CPU** bound).

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**.

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

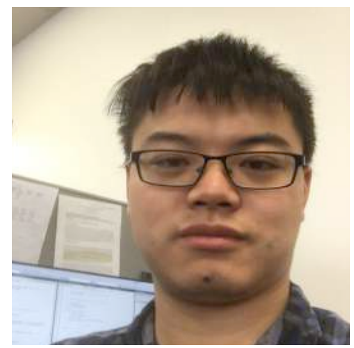
- Generic system calls are more sensitive than specialized system calls
  - *write* and *sendto* can both be used to send data through a socket, but the sensitivity of *write* is twice that of *sendto*.
- System calls with an array parameter of a buffer are more sensitive to perturbations than those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

Acknowledgement

Prof. Daniela Oliveira
Advisor

Andrew Lee
Co-Author

Dr. Donald E. Porter
Co-Author



Aokun Chen
Co-Author

Dr. Matt Bishop
Co-Author

Thank you!

Questions?

Ruimin Sun
gracesrm@ufl.edu