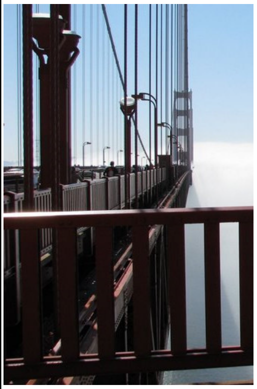


Computer Science at the University of Virginia

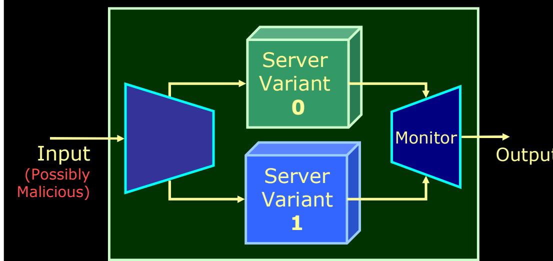
Redundant Computing for Security

David Evans
University of Virginia

Work with Ben Cox, Anh Nguyen-Tuong, Jonathan Rowanhill, John Knight, and Jack Davidson

Yahoo! Tech Talk
16 October 2008

The Basic Idea

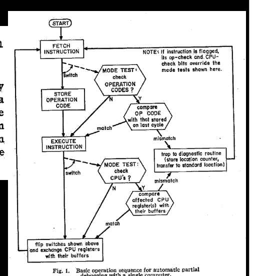


Attacker must find one input that compromises both variants

A Combination Hardware–Software Debugging System

K. C. KNOWLTON

Abstract—A scheme is proposed for automatically detecting many programming errors; in particular, those errors which can cause a program to misbehave in different ways, depending upon how the faulty program and its data are mapped into storage. Error detection is accomplished by simultaneously running two versions of a program which purport to be logically identical, with appropriate hardware checking between them.



IEEE Transactions on Computers, Jan 1968

Image: National Maritime Museum, London

Nevil Maskelyne
5th English Astronomer Royal, 1765-1811

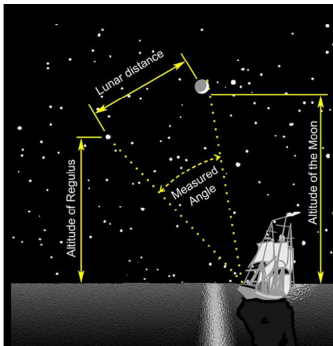


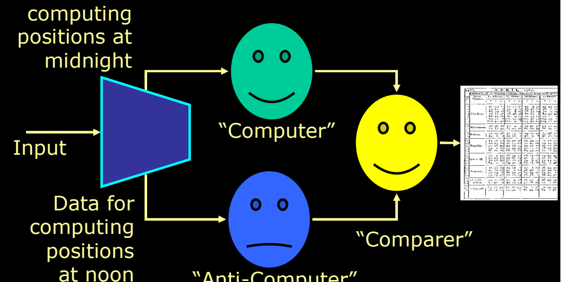
Image: Michael Daly, Wikimedia Commons

[118] OCTOBER 1772.

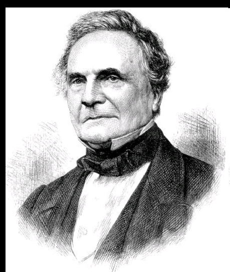
Diſtances of ☽'s Center from ☉, and from Stars weſt of her.

Days.	Stars Names.	Noon. D. M. S.	3 Hours. D. M. S.	6 Hours. D. M. S.	9 Hours. D. M. S.
1	The Sun.	62. 6. 55	63. 44. 49	65. 22. 18	66. 59. 22
2		74. 58. 25	76. 32. 59	78. 7. 10	79. 40. 56
3		87. 24. 0	88. 55. 28	90. 26. 35	91. 57. 21
4		99. 26. 2	100. 54. 47	102. 23. 14	103. 51. 23
5		111. 7. 52	112. 34. 22	114. 0. 37	115. 26. 38
3	Antares.	33. 0. 51	34. 36. 17	36. 11. 37	37. 46. 49
4		[illegible]	47. 14. 40	[illegible]	[illegible] 22. 24
5		[illegible]	59. 40. 36		[illegible]
6					

Maskelyne's Redundant Computing Data Diversity



Babbage's Review



"I wish to God these calculations had been executed by steam."
Charles Babbage, 1821

...back to the 21st century (and beyond)

- Moore's Law: number of transistors/$ increases exponentially
- Einstein's Law: speed of light isn't getting faster

CPU cycles are becoming free, but only in parallel.

- Eastwood/Turing Law: "If you want a guarantee, buy a toaster."
- Sutton's Law: "That's where the money is."

Vulnerabilities and attackers aren't going away.

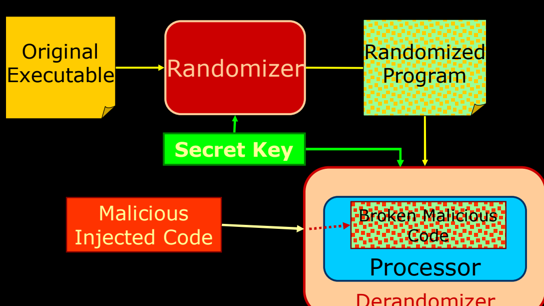
Using Extra Cores for Security

- Despite lots of effort:
  - Automatically parallelizing programs is still only possible in rare circumstances
  - Human programmers are not capable of thinking asynchronously
- Most server programs do not have fine grain parallelism and are I/O-bound
- Hence: lots of essentially free cycles for security

Security Through Diversity

- Address-Space Randomization
  - [Forest+ 1997, *PaX ALSR* 2001, Bhatkar+ 2003, *Windows Vista* 2008]
- Instruction Set Randomization
  - [Kc+ 2003, Barrantes+ 2003]
- Data Diversity

Example: Instruction Set Randomization

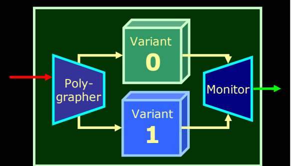


Limitations of Diversity Techniques

- Weak security assurances
  - Probabilistic guarantees
  - Uncertain what happens when it works
- Need high-entropy variations
  - Address-space may be too small [Shacham+, CCS 04]
- Need to keep secrets
  - Attacker may be able to incrementally probe system [Sovarel+, USENIX Sec 2005]
  - Side channels, weak key generation, etc.

N-Variant System Framework

- Polygrapher
  - Replicates input to all variants
- Variants
  - *N* processes that implement the same service
  - Vary property you hope attack depends on: memory locations, instruction set, system call numbers, calling convention, data representation, …



- Monitor
  - Observes variants
  - Delays external effects until all variants agree
  - Initiates recovery if variants diverge

No secrets, high assurances, no need for entropy

N-Version Programming

[Avizienis & Chen, 1977]

- Multiple teams of programmers implement same specification
- Voter compares results and selects most common
- No guarantees: teams may make same mistake

N-Variant Systems

- Transformer automatically produces diverse variants
- Monitor compares results and detects attack
- Guarantees: variants behave differently on particular input classes

Variants Requirements

- *Detection* Property

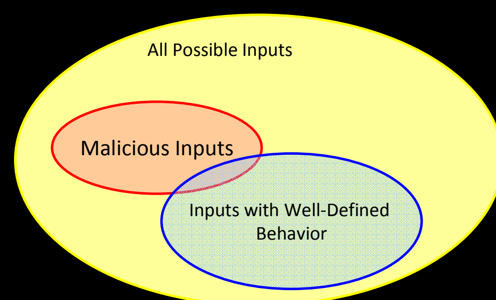
  Any attack that compromises one variant causes the other to “crash” (behave in a way that is noticeably different to the monitor)
- *Normal Equivalence* Property

  Under normal inputs, the variants stay in equivalent states:

$$\mathcal{A}_0(S_0) \equiv \mathcal{A}_1(S_1)$$

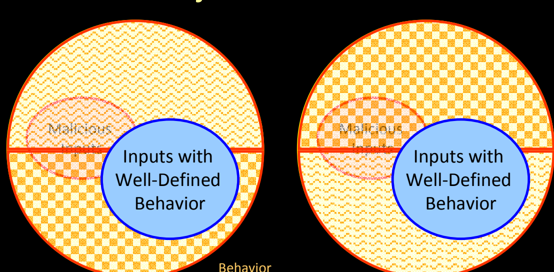
Actual states are different, but abstract states are equivalent

Opportunity for Variation



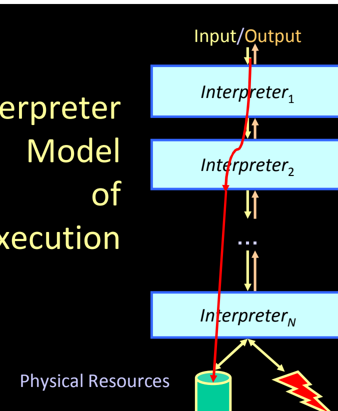
Can’t change “well-defined” behavior, but can change “undefined” behavior

Disjoint Variants



Variant 0   Variant 1

Interpreter Model of Execution



Each interpreter manipulates different data types, protecting inner interpreters.

Malicious data finds a way through protections in one interpreter to exploit functionality in lower interpreters.

Our goal: replace interpreters so malicious data is interpreted.

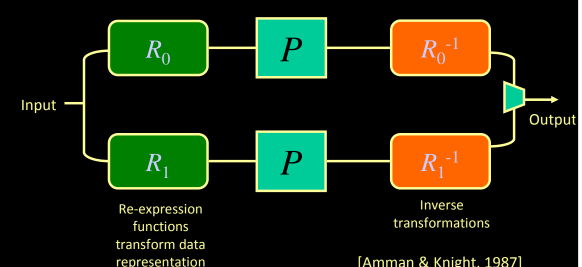
Example: Address-Space Partitioning

- Variation
  - Variant 0: addresses all start with **0**
  - Variant 1: addresses all start with **1**
- Normal Equivalence
  - Map addresses to same address space
  - Assumes normal behavior does not depend on absolute addresses
- Detection Property
  - Any injected *absolute* load/store is invalid on one of the variants

Example: Instruction Set Tagging

- Variation: add an extra bit to all opcodes
  - Variation 0: tag bit is a **0**
  - Variation 1: tag bit is a **1**
  - Run-time: check and remove bit (software dynamic translation)
- Normal Equivalence:
  - Remove the tag bits
  - Assume well-behaved program does not rely on its own instructions
- Detection Property
  - Any (tagged) opcode is invalid on one variant
  - Injected code (identical on both) cannot run on both

Data Diversity

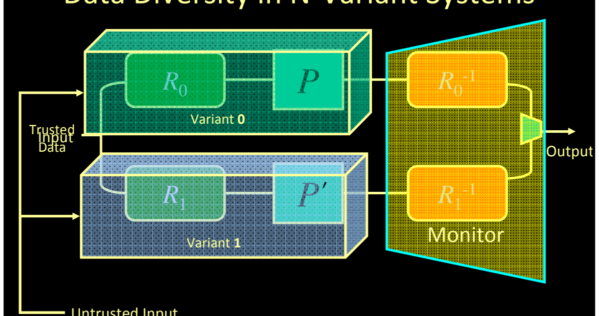


[Amman & Knight, 1987] and [Maskelyne 1767]

Variations on Interpreters

Variation	**Data Type**	**Variant 0**	**Variant 1**
Address Space Partitioning	Address	$R_0(a) = a$ $R_0^{-1}(a) = a$	$R_1(a) = a +$ 0x800… $R_1^{-1}(a) = a -$ 0x800…
Instruction Set Tagging	Instruction	$R_0(inst) = 0 \parallel inst$ $R_0^{-1}(0 \parallel inst) = inst$	$R_1(inst) = 1 \parallel inst$ $R_1^{-1}(1 \parallel inst) = inst$
…	?	?	?

Data Diversity in N-Variant Systems



UID Corruption Attacks

```
uid_t user;
...
user = authenticate();
...
setuid(user);
```

Examples in [Chen+, USENIX Sec 2005]

Attacker corrupts user

Goal: thwart attacks by changing data representation

UID Data Diversity

root:	0	root:	0x7FFFFFFF
bin:	1	bin:	0x7FFFFFFE
nobody:	99	nobody:	0x7FFFFF9C

Identity Re-expression

$R_0(u) = u$
$R_0^{-1}(u) = u$

Variant 0

Flip Bits Re-expression

$R_1(u) = u \oplus \text{0x7FFFFFFF}$
$R_1^{-1}(u) = u \oplus \text{0x7FFFFFFF}$

Variant 1

Data Transformation Requirements

- Normal equivalence:
  - $\forall x: T, R_i^{-1}(R_i(x)) = x$
  - All trusted data of type T is transformed by R
  - All instructions in P that operate on data of type T are transformed to preserve original semantics on re-expressed data
- Detection:
  - $\forall x: T, R_0^{-1}(x) \neq R_1^{-1}(x))$ (disjointedness)

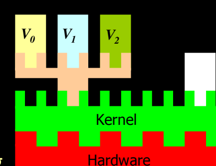
Ideal Implementation

- Polygrapher
  - Identical inputs to variants at same time
- Monitor
  - Continually examine variants completely
- Variants
  - Fully isolated, behave identically on normal inputs

Infeasible for real systems

Framework Implemention

- Modified Linux 2.6.11 kernel
- Run variants as processes
- Create 2 new system calls
  - n_variant_fork
  - n_variant_execve
- Replication and monitoring by wrapping system calls



Wrapping System Calls

- All calls: check each variant makes the same call
- I/O system calls (process interacts with external state) (e.g., open, read, write)
  - Make call once, send same result to all variants
- Reflective system calls (e.g, fork, execve, wait)
  - Make call once per variant, adjusted accordingly
- Dangerous
  - Some calls break isolation (mmap) or escape framework (execve)
  - Current solution: disallow unsafe calls


```
sys_write_wrapper(int fd, char __user * buf, int len) {
  if (!IS_VARIANT(current)) { perform system call normally }
  else {
    if (!inSystemCall(current->nv_system))  {  // First variant to reach
      Save Parameters
      Sleep
      Return Result Value
    } else if (currentSystemCall(current->nv_system) !=SYS_WRITE) {
      DIVERGENCE – different system calls
    } else if (!Parameters Match) {
      DIVERGENCE – different parameters
    } else if (!isLastVariant(current->nv_system)) {
      Sleep
      Return Result Value
    } else {
      Perform System Call
      Save Result
      Wake Up All Variants
      Return Result Value
    }
  }
}
```


30

Implementing Variants

- Address Space Partitioning
  - Specify segments' start addresses and sizes
  - OS detects injected address as SEGV
- Instruction Set Tagging
  - Use Diablo [De Sutter+ 03] to insert tags into binary
  - Use Strata [Scott+ 02] to check and remove tags at runtime

Implementing UID Variation

- Assumptions:
  - We can identify UID data (uid_t, gid_t)
  - Only certain operations are performed on it:
    - Assignments, Comparisons, Parameter passing

Program shouldn't depend on actual UID values, only the users they represent.

Code Transformation

- Re-express UID constants in code

if (!getuid()) ⇒ if (getuid() == 0)

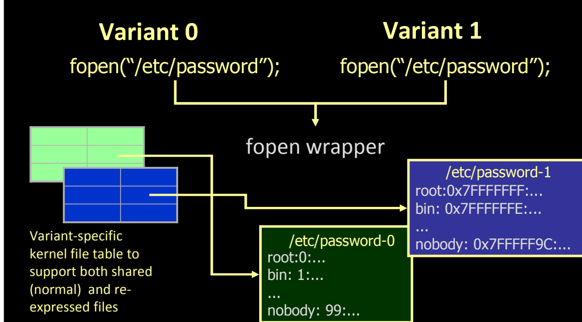
$\downarrow R_1$

⇒ if (getuid() == 0x7FFFFFFF)

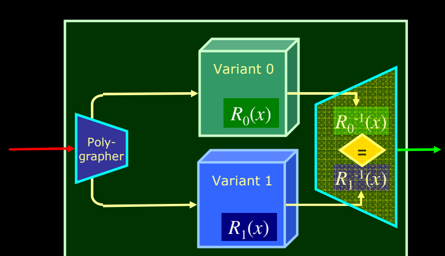
- Preserve semantics
  - Flip comparisons
- Fine-grained monitoring:

  uid_t uid_value(uid_t), bool check_cond(bool)
- External Trusted Data (e.g., /etc/passwd)

Re-expressed Files

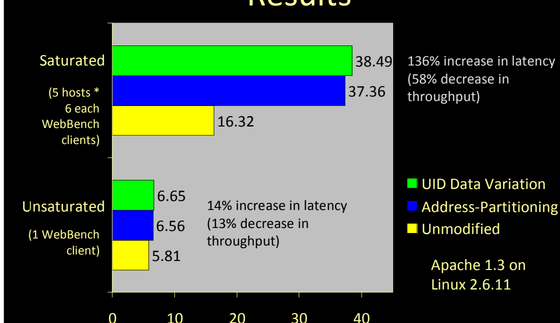


Thwarting UID Corruption



Injected UID: $\forall x$: T, $R_0^{-1}(x) \neq R_1^{-1}(x)) \Rightarrow$ detected

Results



Open Problems and Opportunities

- Dealing with non-determinism
  - Most sources addressed by wrappers
    - e.g., entropy sources
  - ...but not multi-threading [Bruschi, Cavallero & Lanzi 07]
- Finding useful higher level variations
  - Need specified behavior
  - Opportunities with higher-level languages, web application synthesizers
- Client-side uses (e.g., JavaScript interpreters)
- Giving variants different inputs
  - Character encodings

N-Variant Framework Summary

- Force attacker to simultaneously compromise all variants with same input
- Advantages
  - Enables low-entropy variations
  - High security assurance with no secrets
    - Easier to deploy and maintain than secret diversity
- Disadvantages
  - Expensive for CPU-bound applications
  - Variations limited by need to preserve application semantics

http://www.cs.virginia.edu/nvariant/

Papers: USENIX Sec 2006, DSN 2008
Collaborators: Ben Cox, Anh Nguyen-Tuong, Jonathan Rowanhill, John Knight, Jack Davidson

Supported by National Science Foundation Cyber Trust Program and MURI

Related Work

- Design Diversity
  - HACQIT [Just+, 2002], [Gao, Reiter & Song 2005]
- Probabilistic Variations
  - DieHard [Berger & Zorn, 2006]
- Other projects exploring similar frameworks
  - [Bruschi, Cavallaro & Lanzi 2007], [Salamat, Gal & Franz 2008]