Redundant Computing for Security

David Evans
University of Virginia

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

TRUST Seminar
UC Berkeley
25 September 2008

The Basic Idea

Input (Possibly Malicious)

Server Variant 0

Monitor

Output

Server Variant 1

Attacker must find one input that compromises both variants

A Combination Hardware-Software Debugging System

K. C. KNOWLTON

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

IEEE Transactions on Computers, Jan 1968

Nevil Maskelyne
5th English Astronomer Royal, 1765-1811


Maskelyne’s Redundant Computing Data Diversity

Data for computing positions at midnight

Input

“Computer”

Data for computing positions at noon

“Anti-Computer”

“Comparer”
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 any faster
- Eastwood/Turing Law: “If you want a guarantee, buy a toaster.”
- Sutton’s Law: “Because that’s where the money is.”

Conclusion: CPU cycles are becoming free, but vulnerabilities and attackers aren’t going away

Security Through Diversity

- Address-Space Randomization
- Instruction Set Randomization
  - [Kc+ 2003, Barrantes+ 2003]
- DNS Port Randomization
- Data Diversity

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 attacks
- 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:
  \[ A_i(S_0) \equiv A_i(S_1) \]

  Actual states are different, but abstract states are equivalent.

Disjoint Variants

- **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

Opportunity for Variation

- **All Possible Inputs**
- **Malicious Inputs**
- **Inputs with Well-Defined Behavior**

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

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

Disjoint 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

- **Re-expression functions** transform data representation
- **Inverse transformations**

[Amman & Knight, 1987] and [Maskelyne 1767]
Data Diversity in N-Variant Systems

UID Corruption Attacks

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

Goal: thwart attacks by changing data representation

UID Data Diversity

<table>
<thead>
<tr>
<th>Variant 0</th>
<th>Variant 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>root: 0</td>
<td>root: 0x7FFFFFFF</td>
</tr>
<tr>
<td>bin: 1</td>
<td>bin: 0x7FFFFFFE</td>
</tr>
<tr>
<td>nobody: 99</td>
<td>nobody: 0x7FFFFF9C</td>
</tr>
</tbody>
</table>

Identity Re-expression

\( R_i(u) = u \)
\( R_i^{-1}(u) = u \)

Flip Bits Re-expression

\( R_i(u) = u \oplus 0x7FFFFFFF \)
\( R_i^{-1}(u) = u \oplus 0x7FFFFFFF \)

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:
  \( \forall x : T, R_i^{-1}(x) \neq R_i^{-1}(x) \) (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 Implementation

- Modified Linux 2.6.11 kernel
- Run variants as processes
- Create 2 new system calls
  - \( n\text{-variant\_fork} \)
  - \( n\text{-variant\_execve} \)
- 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

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)
  - 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

```
Variant 0
fopen("/etc/password");
```
```
Variant 1
fopen("/etc/password");
```
```
Thwarting UID Corruption

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

Results

- Saturated
  - [5 hosts, 4 each, 10 clients]
  - [5 hosts, 10 clients]
- $5.65^{\text{%, decrease in latency}}$
- $6.81^{\text{%, decrease in throughput}}$
- $136^{\text{%, increase in latency}}$
- $138^{\text{%, decrease in throughput}}$

Open Problems and Opportunities

- Dealing with non-determinism
  - Most sources addressed by wrappers
    - e.g., entropy sources
    - Just 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
- Giving variants different inputs
  - Character encodings

Related Work

- Design Diversity
  - HACCIIT [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]

Backup Slides

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

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

Supported by National Science Foundation Cyber Trust Program and NAVY
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