Security Through Diversity

- Today’s Computing Monoculture
  - Exploit can compromise billions of machines since they are all running the same software
- Biology’s Solution: Diversity
  - Members of a species are different enough that some are immune
- Computer security research: [Cohen 92], [Forrest 97], [Cowan 2003], [Barrantes 2003], [Kc 2003], [Bhatkar 2003], [Just 2004], [Bhatkar, Sekar, DuVarney 2005]

Instruction Set Randomization
[Barrantes+, CCS 03] [Kc+, CCS 03]

- Code injection attacks depend on knowing the victim machine’s instruction set
- Defuse them all by making instruction sets different and secret
  - It is expensive to design new ISAs and build new microprocessors

ISR Defuses Attacks

Automating ISR

How secure is ISR?

**ISR Attack**

- **Attack Client** Incorrect Guess → **ISR-protected Server** Crash!
- **Attack Client** Correct Guess → **ISR-protected Server** Observable Behavior

---

**Server Requirements**

- Vulnerable: buffer overflow is fine
- Able to make repeated guesses
  - No rerandomization after crash
  - Likely if server forks requests (Apache)
- Observable: notice server crashes
- Cryptanalyzable
  - Learn key from one ciphertext-plaintext pair
  - Easy with XOR

---

**Jump Attack**

- **JMP -2 (0xEBFE): jump offset -2**
  - 2-byte instruction: up to $2^{16}$ guesses
  - Produces infinite loop
- Incorrect guess *usually* crashes server

---

**Incremental Jump Attack**

- Guessing first 2 byte masks
- Guessing next byte: < 256 attempts

---

**Guess Outcomes**

<table>
<thead>
<tr>
<th>Correct Guess</th>
<th>Success</th>
<th>False Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incorrect Guess</td>
<td>False Positive</td>
<td>Progress</td>
</tr>
</tbody>
</table>
False Positives

- Injected bytes produce an infinite loop:
  - JMP -4
  - JNZ -2
- Injected bytes are "harmless", later instruction causes infinite loop

False Positives – Good News

- Can distinguish correct mask using other instructions
- Try injecting a "harmless" one-byte instruction
  - Correct: get loop
  - Incorrect: usually crashes
- Difficulty: dense opcodes

False Positives – Better News

- False positives are not random
  - Conditional jump instructions
  - Opcodes 01110000-0111111
- All are complementary pairs: 0111xyzā not taken ⇔ 0111xyzā is!
- 32 guesses must find an infinite loop, about 8 more guesses to learn correct mask

Extended Attack

- Near jump to return location
  - Execution continues normally
  - No infinite loops
- 0xCD 0xCD is interrupt instruction guaranteed to crash

Expected Attempts

- 0xEB (JMP)
- 0x06 (offset)
- 0xE9 (Near Jump)

~ 15½ to find first jumping instruction
+ ~ 8 to determine correct mask
23½ expected attempts per byte

Experiments

- Implemented attack against constructed vulnerable server protected with RISE [Barrantes et. al, 2003]
  - Need to modify RISE to ensure child processes have same key
- Obtain correct key over 95% of the time
  - 4 byte key in 3½ minutes
  - 4096 bytes in 48 minutes
    (>100,000 guess attempts)
- Is this good enough?
How many key bytes needed?

- Inject malcode in one ISR-protected host
  - Sapphire worm = 376 bytes
- Create a worm that spreads on a network of ISR-protected servers
  - Space for our code: 34,723 bytes
  - Need to crash server ~800K times

Maybe less...?

- VMWare: 3,530,821 bytes
- Java VM: 135,328 bytes
- MicroVM: 100 bytes

Entire MicroVM Code

```
push dword ebp
mov ebp, WORM_ADDRESS + WORM_REG_OFFSET
pop dword ebp + WORM_DATA_OFFSET
xor eax, eax ; WormIP = 0 (load from ebp + eax)
read more worm: read NUM_BYTES at a time until worm is done
xor ebx, ebx ; copy next Worm block into execution buffer
add eax, NUM_BYTES ; change WormIP
pushad ; save register vals
mov edi, dword [ebp]
mov esi, dword [ebp + ESI_OFFSET]
mov ebx, dword [ebp + EBX_OFFSET]
mov edx, dword [ebp + EDX_OFFSET]
mov eax, dword [ebp + EAX_OFFSET]
beg_worm_exec: ; this is the worm execution buffer
    nop nop nop nop nop nop nop nop nop
    jmp read_more_worm
```

MicroVM Learned Key Bytes

76 bytes of code
+ 22 bytes for execution
+ 2 bytes to avoid NULL
= 100 bytes is enough
> 99% of the time

Worm code must be coded in blocks that fit into execution buffer
(pad with noops so instructions do not cross block boundaries)

Deploying a Worm

- Learn 100 key bytes to inject MicroVM
  - Median time: 311 seconds, 8422 attempts
  - Fast enough for a worm to spread effectively
- Inject pre-encrypted worm code
  - XORed with the known key at location
  - Insert NOOPs to avoid NULLs
- Inject key bytes
  - Needed to propagate worm

Preventing Attack: Break Attack Requirements

- Vulnerable: eliminate vulnerabilities
  - Rewrite all your code in a type safe language
- Able to make repeated guesses
  - Rerandomize after crash
- Observable: notice server crashes
  - Maintain client socket after crash?
- Cryptanalyzable
  - Use a strong cipher like AES instead of
Better Solution

- Avoid secrets!
  - Keeping them is hard
  - They can be broken or stolen
- Prove security properties without relying on assumptions about secrets or probabilistic arguments

N-Variant Systems: A Secretless Framework for Security through Diversity


Thomas Jefferson

- Author of “Declaration of Independence”
- 3rd President of United States
- Cryptographer, scientist, architect

Computer Science at UVA

- Strong research groups in:
  - Security (me, Jack Davidson, Anita Jones, Alf Weaver)
  - Software Engineering (me, Mary Lou Soffa, John Knight)
  - Architecture (Gurumurthi, Skadron)
  - Sensor Networks (Stankovic)
  - Theory (Mishra)
  - Graphics (Humphreys)
- 75 PhD students

University of Virginia
Charlottesville, Virginia, USA
Founded by Thomas Jefferson, 1819
N-Version Programming
[Avizienis & Chen, 1977]
- Multiple teams of programmers implement same spec
- Voter compares results and selects most common
- No guarantees: teams may make same mistake
- Transformer automatically produces diverse variants
- Monitor compares results and detects attack
- Guarantees: variants behave differently on particular input classes

N-Variant System Framework
- Polygrapher
  - Replicates input to all variants
- Variants
  - N processes implement the same service
  - Vary property you hope attack depends on: memory locations, instruction set, file names, system call numbers, scheduler, calling convention, ...
- Monitor
  - Observes variants
  - Delays effects until all variants agree
  - Starts recovery if variants diverge

Variants Requirements
- Detection Property
  Any attack that compromises Variant 0 causes Variant 1 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:
  \[ A_0(S_0) \equiv A_1(S_1) \]  Actual states are different, but abstract states are equivalent

Memory Partitioning
- Variation
  - Variant 0: addresses all start with 0
  - Variant 1: addresses all start with 1
- Normal Equivalence
  - Map addresses to same address space
- Detection Property
  - Any absolute load/store is invalid on one of the variants

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
  - At run-time check bit and remove it
    - Low-overhead software dynamic translation using Strata [Scott, et al., CGO 2003]
- Normal Equivalence: Remove the tag bits
- Detection Property
  - Any (tagged) opcode is invalid on one variant
  - Injected code (identical on both) cannot run on both
Implementing N-Variant Systems

- Competing goals:
  - Isolation: of monitor, polygrapher, variants
  - Synchronization: variants must maintain normal equivalence (nondeterminism)
  - Performance: latency (wait for all variants to finish) and throughput (increased load)
- Two implementations:
  - Divert Sockets (prioritizes isolation over others)
  - Kernel modification (sacrifices isolation for others)

Kernel Modification Implementation

- Modify process table to record variants
- Create new fork routine to launch variants
- Intercept system calls:
  - 289 calls in Linux
  - Check parameters are the same for all variants
  - Make call once

Wrapping System Calls

- I/O system calls (process interacts with external state) (e.g., open, read, write)
  - Make call once, send same result to all variants
- Process system calls (e.g., fork, execve, wait)
  - Make call once per variant, adjusted accordingly
- Dangerous:
  - mmap: each variant maps segment into own address space, only allow MAP_ANONYMOUS (shared segment not mapped to a file) and MAP_PRIVATE (writes do not go back to file)
  - execve: cannot allow

System Call Wrapper Example

```c
ssize_t sys_read(int fd, const void *buf, size_t count) {
  if (hasSibling (current)) {
    record that this variant process entered call
    if (inSystemCall (current->sibling)) { // this variant is first
      save parameters
      sleep // sibling will wake us up
      get result and copy *buf data back into address space
    } else if (currentSystemCall (current->sibling) == SYS_READ) {
      // I'm second variant, sibling is waiting
      if (parameters match) { // match depends on variation
        perform system call
        save result and data in kernel buffer
        wake up sibling
        return result;
      } else {
        DIVERGENCE ERROR!
        DIVERGENCE ERROR!
      }
    } else { // sibling is in a different system call!
      DIVERGENCE ERROR!
    }
  } else { // sibling has entered different system call
    DIVERGENCE ERROR!
  }
  ...
}
```

Summary

- Producing artificial diversity is easy
  - Defeats undetermined adversaries
- Keeping secrets is hard
  - Remote attacker can break ISR-protected server in < 6 minutes
- N-variant systems framework offers provable (but expensive) defense
  - Effectiveness depends on whether variations vary things that matter to attack

Overhead

<table>
<thead>
<tr>
<th>Description</th>
<th>Unmodified Apache, unmodified kernel</th>
<th>2-variant system, address space partitioning</th>
<th>2-variant system, instruction tagging</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput (MB/s)</td>
<td>2.36</td>
<td>2.04</td>
<td>1.80</td>
</tr>
<tr>
<td>Latency (ms)</td>
<td>2.35</td>
<td>2.77</td>
<td>3.02</td>
</tr>
<tr>
<td>Throughput (MB/s)</td>
<td>9.70</td>
<td>5.06</td>
<td>3.55</td>
</tr>
<tr>
<td>Latency (ms)</td>
<td>17.65</td>
<td>34.20</td>
<td>48.30</td>
</tr>
</tbody>
</table>

Latency increases ~18% Throughput 36% of original
Diversity depends on your perspective

From my USENIX Security 2004 Talk, What Biology Can (and Can't) Teach us about Security

Links: http://www.cs.virginia.edu/nvariant
Contributors: Ben Cox, Jack Davidson, Adrian Filipi, Jason Hiser, Wei Hu, John Knight, Ana Nora Sovarel, Anh Nguyen-Tuong, Nate Paul, Jonathan Rowanhill

Funding: National Science Foundation, DARPA