The N-Variant Systems Framework
Polygraphing Processes for Sensible Security

David Evans
http://www.cs.virginia.edu/~dave
University of Virginia
Computer Science

University of Texas at San Antonio
4 October 2005

Security Through Diversity
• Today’s Computing Monoculture
  – Exploit can compromise billions of machines since they are all running the same software
• Biological Diversity
  – All successful species use very expensive mechanism (sex) to maintain diversity
• Computer security research: [Cohen 92], [Forrest97], [Cowan+2003], [Barrantes+2003], [Kc+2003], [Bhaktar+2003], [Just+2004], [Bhaktar, 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’s expensive to design new ISAs and build new microprocessors

Automating ISR

ISR Defuses Attacks

ISR Designs

Columbia [Kc 03]  RISE [Barrantes 03]

Randomization Function
XOR or 32-bit transposition  XOR

Key Size
32 bits (same key used for all locations)  program length (each location XORed with different byte)

Transformation Time
Compile Time  Load Time

Derandomization
Hardware  Software (Valgrind)
How secure is ISR?
Slows down an attack about 6 minutes!

Under the right circumstances...

Memory Randomization Attack
- Brute force attack on memory address space randomization (Shacham et. al. [CCS 2004]): 24-bit effective key space
- Can a similar attack work against ISR?
  - Larger key space: must attack in fragments
  - Need to tell if partial guess is correct

ISR Attack
- Attack Client
- Incorrect Guess
  - ISR-protected Server
  - Crash!
- 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

Two Attack Ideas
- RET (0xC3): return from procedure
  - 1-byte instruction: up to 256 guesses
  - Returns, leaves stack inconsistent
    - Only works if server does something observable before crashing
- JMP -2 (0xEBFE): jump offset -2
  - 2-byte instruction: up to $2^{16}$ guesses
  - Produces infinite loop
- Incorrect guess usually crashes server
Jump Attack

- 2^{16} possible guesses for 2-byte instruction
- Correct guess produces infinite loop

Incremental Jump Attack

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

Guess Outcomes

<table>
<thead>
<tr>
<th>Correct Guess Behavior</th>
<th>Observe &quot;Correct&quot; Behavior</th>
<th>Observe &quot;Incorrect&quot; Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Correct Guess</td>
<td>Success</td>
</tr>
<tr>
<td>False Negative</td>
<td>Incorrect Guess</td>
<td>False Positive</td>
</tr>
<tr>
<td>False Positive</td>
<td></td>
<td>Progress</td>
</tr>
</tbody>
</table>

False Positives

- Injected bytes produce an infinite loop:
  - JMP -4
  - JNZ -2
- Injected bytes are “harmless”, later executed instruction causes infinite loop
- Injected guess causes crash, but timeout expires before remote attacker observes

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
  - No pair that differs in only last bit are reliably different in harmfulness

False Positives – Better News

- False positives are not random
  - Conditional jump instructions
  - Opcodes 01110000-0111111
- All are complementary pairs: 0111_{xyz}a not taken \iff 0111_{xyz} \bar{a} is!
- 32 guesses always find an infinite loop
- About 8 additional guesses to determine correct mask
Experiments

- Implemented attack against constructed vulnerable server protected with RISE
  - Memory space randomization works!
  - Turned off Fedora’s address space randomization
  - Needed to modify RISE
  - Ensure forked processes use same randomization key (other proposed ISR implementations wouldn’t need this)

- Obtain correct key over 95% of the time
  - Sometimes can’t because unable to inject NULLs

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 FEEB attack code: 34,723 bytes
  - Need to crash server ~800K times
Maybe less...?

- VMWare: 3,530,821 bytes
- Java VM: 135,328 bytes
- Minsky’s UTM: 7 states, 4 colors
- MicroVM: 100 bytes

Entire MicroVM Code

```assembly
push dword ebp  mov ebp, WORM_ADDRESS + WORM_REG_OFFSET
pop dword [ebp + WORM_DATA_OFFSET]
lea ecx, eax  ; WORM_P = 0 (load from ebp + eax)
read more worm: ; read NUM_BYTES at a time until worm is done
lea eax, ecx  ; save ecx/edx
mov edx, ecx  ; WORM_ADDRESS
mov edx, [ebp + eax]  ; get saved WormIP
add edx, eax  ; WORM_P = 0 (load from ebp + eax)
add eax, NUM_BYTES ; change WormIP
push eax       ; save register val
mov edx, [ebp] ; restore WormIP
mov ebx, [ebp + EBX_OFFSET] ; restore worm registers
mov edx, [ebp + EAX_OFFSET] ; restore worm registers
mov esi, [ebp + ECX_OFFSET] ; restore worm registers
mov edi, [ebp + EDX_OFFSET] ; restore worm registers
mov ecx, [ebp + EDX_OFFSET] ; restore worm registers
mov edx, [ebp + EBX_OFFSET] ; restore worm registers
add edx, eax  ; set new WormIP
mov ecx, [ebp + EAX_OFFSET] ; set new WormIP
mov ebx, [ebp + ECX_OFFSET] ; set new WormIP
mov esi, [ebp + EDX_OFFSET] ; set new WormIP
pushad        ; save register vals
mov edi, [ebp] ; restore worm registers
mov esi, [ebp + ESI_OFFSET] ; restore worm registers
mov ebx, [ebp + EBX_OFFSET] ; restore worm registers
mov edx, [ebp + EDX_OFFSET] ; restore worm registers
mov ecx, [ebp + ECX_OFFSET] ; restore worm registers
mov eax, [ebp + EAX_OFFSET] ; restore worm registers
popad         ; restore reg vals
jmp read_more_worm
```

Making Jumps

- Within a block - short relative jump is fine
- Between worm blocks
  - From end of block, to beginning of block
  - Update the WormIP stored on the stack
  - Code conditional jump, JZ target in worm as:
    ```assembly
    JNZ +5 ; if opposite condition, skip
    MOV [ebp + WORMIP_OFFSET] target
    ```

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 when necessary 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
  - Resecond a crash
- Observed: notice server crashes
  - Maintain client socket after crash?
- Cryptanalyzable
  - Use a strong cipher like AES instead of XOR
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

Polygraphing Processes: N-Variant Systems for Secretless Security
work with Ben Cox, Jack Davidson, Adrian Filipi, Jason Hiser, Wei Hu, John Knight, Anh Nguyen-Tuong, Jonathan Rowanhill

2-Variant System

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

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

N-Variant System Framework

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.
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., CACM 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

Indirect Code Injection Attack

- Inject bytes into data buffer
- Original code transforms contents of that buffer (XORing every byte with a different value on $P_1$ and $P_2$)
- Relative jump to execute injected, transformed code

- **What went wrong?**
  - Normal Equivalence property violated: need to know that data manipulated differently is never used as code

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)
Implementation: Divert Sockets [Adrian Filipi]

- Process intercepts traffic (nvpd)
- Uses divert sockets to send copies to isolated variants (can be on different machines)
- Waits until all variants respond to request before returning to client
- Adjusts TCP sequence numbers to each variant appears to have normal connection

Implementation: Kernel Modification [Ben Cox]

- 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
- Low overhead, lack of isolation

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
- Special:
  - 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)

System Call Wrapper Example

```c
ssize_t sys_read(int fd, const void *buf, size_t count) {
  if (hasSibling (current)) {
    if (hasSibling (current->sibling)) {
      // this variant is first
      save parameters
      sleep // sibling will wake us up
      get result and copy *buf data back into address space
      return result;
    } else if (currentSystemCall (current->sibling) == SYS_READ) {
      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 { // DIVERGENCE ERROR!
      ... // DIVERGENCE ERROR!
    } else { // sibling is in a different system call!
      DIVERGENCE ERROR!
    }
  } else { // DIVERGENCE ERROR!
    ... // DIVERGENCE ERROR!
  }

  return -1; // DIVERGENCE ERROR!
}
```

Current Status

- Can run apache with address and instruction tag variations
  - Thwarts any attack that depends on referencing an absolute address or executing injected code
- Open problems
  - Non-determinism, persistent state
  - Establishing normal equivalence
- Cost
  - nvpd implementation, https, 4x machines: Latency x 2.3
  - Kernel modification (hopefully better, no numbers yet)
Diversity depends on your perspective.

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

Questions?


Contributors: Ben Cox, Jack Davidson, Adrian Filipi, Jason Hiser, Wei Hu, John Knight, Anh N. Nguyen, Nata Paul, Anh Nguyen-Tuong, Jonathan Rowanhill

Funding: NSF CyberTrust, DARPA SRS