Where's the FEEB?
Effectiveness of Instruction Set Randomization

CERIAS Security Seminar
Purdue University
9 March 2005

David Evans
University of Virginia
work with Nora Sovarel, Nate Paul
and the UVA/CMU Genesis Project

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 (i.e., sex) to maintain diversity
• Computer security research: [Cohen 92],
  [Forrest + 97], [Cowan + 2003], [Barrantes + 2003], [Kc + 2003], [Bhatkar + 2003], [Just + 2004]

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

<table>
<thead>
<tr>
<th></th>
<th>Columbia [Kc 03]</th>
<th>RISE [Barrantes 03]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Randomization Function</td>
<td>XOR or 32-bit transposition</td>
<td>XOR</td>
</tr>
<tr>
<td>Key Size</td>
<td>32 bits (same key used for all locations)</td>
<td>program length (each location XORed with different byte)</td>
</tr>
<tr>
<td>Transformation Time</td>
<td>Compile Time</td>
<td>Load Time</td>
</tr>
<tr>
<td>Derandomization</td>
<td>Hardware</td>
<td>Software (Valgrind)</td>
</tr>
</tbody>
</table>
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

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
**Incremental Jump Attack**

- Vulnerable Buffer
  - Guessing first 2 byte masks
  - Overwritten Return Address
  - Guessed Masks
- Overwritten Return Address
- 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 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:
  0111xyzα not taken ⇔ 0111xyzβ is!
- 32 guesses always find an infinite loop
- About 8 additional guesses to determine 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

<table>
<thead>
<tr>
<th>Jumping Instruction</th>
<th>Attempts</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xEB (JMP)</td>
<td>~ 15½</td>
</tr>
<tr>
<td>0x69 (Near Jump)</td>
<td>~ 8</td>
</tr>
<tr>
<td>0xCD (INT)</td>
<td>23½</td>
</tr>
</tbody>
</table>

- "Crash Zone"
- Expected attempts per byte = 23½

Experiments

- Implemented attack against constructed vulnerable server protected with RISE [Barrantes et. al, 2003]
  - Memory space randomization works!
  - Turned of 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

Attempts Required

- 4339 attempts to get first 2 bytes
- 101,551 attempts to get 4096 bytes

Attempts per Byte

- Drops to below 24 average attempts per byte

Total Time

- 4-byte key (Columbia implementation) in < 3½ minutes
- 4096-byte key in 48 minutes

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]
read more worm: read NUM_BYTES at a time until worm is done
add eax, ecx
mov eax, NUM_BYTES
mov dword esi, WORM_ADDRESS
mov edx, dword [eax]
add edx, eax
mov edx, dword [WORM_ADDRESS + WORM_REG_OFFSET]
add eax, ecx
mov eax, ecx
xor eax, eax
; WormIP = 0 (load from ebp + eax)
read_more_worm:
; read NUM_BYTES at a time until worm is done
mov byte cl, NUM_BYTES
mov dword esi, WORM_ADDRESS
add dword esi, eax
mov edi, [esi + eax]
mov edi, [edi + EBX_OFFSET]
rep movsb
add eax, NUM_BYTES
; change WormIP
pushad
mov edi, [ebp]
mov edi, [ebp + ESI_OFFSET]
mov edi, [ebp + EBX_OFFSET]
mov edi, [ebp + EDX_OFFSET]
mov edi, [ebp + ECX_OFFSET]
mov edi, [ebp + EAX_OFFSET]
begin_worm_exec:
; this is the worm execution buffer
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
mov [ebp], edi
mov [ebp + ESI_OFFSET], esi
mov [ebp + EBX_OFFSET], ebx
mov [ebp + EDX_OFFSET], edx
mov [ebp + ECX_OFFSET], ecx
mov [ebp + EAX_OFFSET], eax
popad
jmp read_more_worm
```

Deployment 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

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
    ```

Preventing Attack: Break Requirement

- 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 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

Disjoint Variants

- Any attack that succeeds against Variant A must cause Variant B to crash
- Monitor observes crash and recovers

Examples:
- Instruction Sets
- Memory Addresses
- Schedule Interleaving

Challenges

- Engineering
  - Input replicator and monitor
  - Shared state (databases, files)
  - Nondeterminism (session state)
- Security
  - Proving variants are disjoint
  - Multi-stage attacks
  - Achieving high-level disjoint properties

Secretless Security Structure

work with Jack Davidson, Jonathan Hill, John Knight & Anh Nguyen-Tuong

Input (Possibly Malicious)

Server Variant A

Input Replicator

Server Variant B

Monitor

Output

Making Disjoint Variants

JMP
CALL
JO
JNO
JB
JNB
JZ
JNZ

Variant A

...

Variant B

Diversity depends on your perspective

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

- Diversity defenses defeat undetermined adversaries
- Determined adversaries may be able to determine secrets
  - Break ISR-protected server in < 6 minutes
- Secretless diversity designs promise provable security against classes of attack

Credits

Nora "NORandomizer" Sovarel
Nate "Byte Annihilator" Paul

Genesis Project: Jack Davidson, Adrian Filipi, Jonathan Hill, John Knight, Anh Nguyen-Tuong, Chenxi Wang (CMU)
Thanks: Gabriela Barrantes (RISE code), Stephanie Forrest, Fred Schneider
Sponsor: DARPA SRS (Lee Badger), NSF CAREER/ITR

Questions?