Smashing the Gadgets: Hindering the Return-Oriented Programming Using In-Place Code Randomization

Ben Niu @ Aug 31, 2012
Return-Oriented Programming (ROP) (1/2)
Return-Oriented Programming (ROP) (1/2)

- ROP is based on buffer overrun.
  - The gadgets are small pieces of code performing simple computations and ending with “ret”s.
  - For a large piece of code (libc), ROP is Turing-complete.

- ROP is immune to
  - Non-Executable Data protection.
  - Instruction Set Randomization.
  - ASLR (position-dependent code cannot be randomly located in memory)

- ROP can be prevented by
  - CFI.
  - Function location permutation.
  - Dynamic prevention and others.
Motivation

- Third-party applications often come without the source code.

- Existing ROP prevention techniques (prior to this paper) have drawbacks:
  - Relying on accurate disassembly.*
  - Incurring high overhead.

- The authors argued that in-place randomization can overcome the shortcomings of existing techniques (prior to this paper)
Steps of In-Place Code Randomization.

- The code is statically analyzed by IDA Pro.
  - Part of the code is extracted from the original binary with high confidence.
  - The remaining code is not dealt with.

- The extracted code is analyzed and randomized applying the following approaches:
  - Atomic instruction substitution.
  - Instruction reordering.
  - Register reassignment.
### Atomic Instruction Substitution.

#### Program Code

<table>
<thead>
<tr>
<th>B0</th>
<th>01</th>
<th>3A</th>
<th>C3</th>
<th>8D</th>
<th>45</th>
<th>80</th>
<th>50</th>
<th>68</th>
</tr>
</thead>
</table>

- `4A8223C7`: `mov al,0x1`
- `4A8223C9`: `cmp al,bl`
- `4A8223CB`: `lea eax,[ebp-0x80]`

#### gadget

| 4A8223C8 | add [edx],edi |
| 4A8223CA | ret            |

---

#### Program Code

<table>
<thead>
<tr>
<th>B0</th>
<th>01</th>
<th>38</th>
<th>D8</th>
<th>8D</th>
<th>45</th>
<th>80</th>
<th>50</th>
<th>68</th>
</tr>
</thead>
</table>

- `4A8223C7`: `mov al,0x1`
- `4A8223C9`: `cmp bl,al`
- `4A8223CB`: `lea eax,[ebp-0x80]`

#### gadget

| 4A8223C8 | add [eax],edi |
| 4A8223CA | fmul [ebp+0x68508045] |
### Instruction Reordering

#### Intra-Basic Block Reordering

**Program Code**

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>4A83D176</td>
<td><code>mov eax,[ecx+0x10]</code></td>
</tr>
<tr>
<td>4A83D179</td>
<td><code>push ebx</code></td>
</tr>
<tr>
<td>4A83D17A</td>
<td><code>mov ebx,[ecx+0xc]</code></td>
</tr>
<tr>
<td>4A83D17D</td>
<td><code>cmp eax,ebx</code></td>
</tr>
<tr>
<td>4A83D17F</td>
<td><code>mov [ecx+0x8],eax</code></td>
</tr>
<tr>
<td>4A83D182</td>
<td><code>jle 0x5c</code></td>
</tr>
</tbody>
</table>

**Gadget**

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>4A83D17B</td>
<td><code>pop ecx</code></td>
</tr>
<tr>
<td>4A83D17C</td>
<td><code>or al,0x3b</code></td>
</tr>
<tr>
<td>4A83D17E</td>
<td><code>ret</code></td>
</tr>
</tbody>
</table>

**Program Code**

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>4A83D176</td>
<td><code>push ebx</code></td>
</tr>
<tr>
<td>4A83D177</td>
<td><code>mov ebx,[ecx+0xc]</code></td>
</tr>
<tr>
<td>4A83D17A</td>
<td><code>mov eax,[ecx+0x10]</code></td>
</tr>
<tr>
<td>4A83D17D</td>
<td><code>mov [ecx+0x8],eax</code></td>
</tr>
<tr>
<td>4A83D180</td>
<td><code>cmp eax,ebx</code></td>
</tr>
<tr>
<td>4A83D182</td>
<td><code>jle 0x5c</code></td>
</tr>
</tbody>
</table>

**Gadget**

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>4A83D17B</td>
<td><code>inc ecx</code></td>
</tr>
<tr>
<td>4A83D17C</td>
<td><code>adc [ecx-0x3cc4f7bf],cl</code></td>
</tr>
</tbody>
</table>

**Diagram**

```
  push ebx
  mov eax,[ecx+0x10]
  mov ebx,[ecx+0xc]
  cmp eax,ebx
  mov [ecx+0x8],eax
  jle 0x5c
```
Instruction Reordering
Register Preservation Code Reordering

4A834B3B  0  push ebx
4A834B3C -4  push esi
4A834B3D -8  mov ebx,ecx
4A834B3F -8  push edi
4A834B40  -C  mov esi,edx

4A834B3B  push edi
4A834B3C  push ebx
4A834B3D  push esi
4A834B3E  mov ebx,ecx
4A834B40  mov esi,edx

4A834B7C  -C  pop edi
4A834B7D  -8  pop esi
4A834B7E  -4  pop ebx
4A834B7F  0  ret

4A834B7C  pop esi
4A834B7D  pop ebx
4A834B7E  pop edi
4A834B7F  ret
Register Reassignment

1. 4A8063BF push esi
2. 4A8063C0 push edi
3. 4A8063C1 mov edi,[ebp+0x8]
4. 4A8063C4 mov eax,[edi+0x14]
5. 4A8063C7 test eax,eax
6. 4A8063C9 jz 0x4a80640b

7. 4A8063CB mov ebx,[ebp+0x10]
8. 4A8063CE push ebx
9. 4A8063CF lea ecx,[ebp-0x4]
10. 4A8063D2 push ecx
11. 4A8063D3 push edi
12. 4A8063D4 call eax

13. 4A806414 mov eax,[ebp+0xc]
14. 4A806417 test eax,eax
15. 4A806419 pop edi
16. 4A80641A pop esi
17. 4A80641B pop ebx
18. 4A80641C jz 0x4a806423
Evaluation

- Over all modified Windows PE and DLL files,
  - 76.9% gadgets are modifiable.
  - 9.5% gadgets are eliminated.
  - 67.4% gadgets are broken.

- The more the randomization alternatives are, the harder attackers can perform attacks.

- The experiments show that in-place randomization is a probabilistic prevention tool without any guarantee.
Critique

- Well-written and easy to understand.
- One of the motivations seems weak.
- No protection guarantee.
- The effectiveness is doubted.
  - To perform an attack, not many gadgets are needed.
  - For each gadget, there exist only several alternatives.
  - The gadget search space is not enlarged greatly.