Juggling the Gadgets: Binary-level Code Randomization using Instruction Displacement

Hyungjoon Koo and Michalis Polychronakis
Memory Corruption: Injection \(\rightarrow\) Reuse

- **Attack goal:** *Divert control flow*

  ![Code Injection](image)

  Run arbitrary code!
Memory Corruption: Injection $\rightarrow$ Reuse

- **Attack goal:** *Divert control flow*

![Diagram showing Code Injection and Code Reuse processes]

- $W \otimes X$
- Run arbitrary code!
- Execute existing code!
Memory Corruption: ROP Concept

```
[Z]
mov [ebx], eax
ret

[Y]
add eax, ebx
ret

[X]
pop eax
pop ebx
ret
```
Memory Corruption: ROP Concept

![Diagram of ROP concept]

- **param_0**
- **param_1**
- **ret addr**
- **frame pointer**
- **buf**

**Exploit Payload**

- **[Z]**
  - `mov [ebx], eax`
  - `ret`

- **[Y]**
  - `add eax, ebx`
  - `ret`

- **[X]**
  - `pop eax`
  - `pop ebx`
  - `ret`
### Memory Corruption: ROP Concept

#### Juggling the Gadgets

<table>
<thead>
<tr>
<th>High</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

- **Z:** Gadget #2
- **Y:** Gadget #1

- **X:** Return Addr
  - **0x2**
  - **0x1**

- **Exploit Payload**
  - **0xdeadbeef**
  - **0xdeadbeef**

#### Assembly Code

```
[Z]
  mov [ebx], eax
  ret

[Y]
  add eax, ebx
  ret

[X]
  pop eax
  pop ebx
  ret
```

---

- **ESP**
- **Low**
Memory Corruption: ROP Concept

![Diagram of ROP concept]

- **ESP**
  - **Low**: `0xdeadbeef`
  - **High**:...

- **X: Return Addr**
  - `0x1`
  - `0x2`

- **Y: Gadget #1**
  - `add eax, ebx`
  - `ret`

- **Z: Gadget #2**
  - `mov [ebx], eax`
  - `ret`

- **Exploit Payload**
  - `pop eax`
  - `pop ebx`
  - `ret`
Memory Corruption: ROP Concept

Juggling the Gadgets

Exploit Payload

[X]
pop eax
pop ebx
ret

[Y]
add eax, ebx
ret

[Z]
mov [ebx], eax
ret

0x2 0x1
X:Return Addr

0xdeadbeef

Y:Gadget #1
Z:Gadget #2

Low

0xdeadbeef

High

...
Memory Corruption: ROP Concept

```
...  
...  
...  
...  
...  
...  
Z:Gadget #2
Y:Gadget #1
0x2
0x1
X:Return Addr
0xdeadbeef
0xdeadbeef
```

```
[Z] mov [ebx], eax
    ret

[Y] add eax, ebx
    ret

[X] pop eax
    pop ebx
    ret
```

Exploit Payload
Memory Corruption: ROP Concept

High

Z: Gadget #2
Y: Gadget #1
0x2
0x1
X: Return Addr
0xdeadbeef
0xdeadbeef

Low

ESP

[Z]
mov [ebx], eax
ret

[Y]
add eax, ebx
ret

[X]
pop eax
pop ebx
ret

Exploit Payload

Juggling the Gadgets
Memory Corruption: ROP Concept

Juggling the Gadgets

ESP

High

... ...

... ...

... ...

Z: Gadget #2

Y: Gadget #1

0x2

0x1

X: Return Addr

0xdeadbeef

0xdeadbeef

[Z] mov [ebx], eax

ret

[Y] add eax, ebx

ret

[X] pop eax

pop ebx

ret

Exploit Payload
ROP Defenses

Two main approaches

- **Address Space Predictability**
  - **Randomization**
    - Breaks the knowledge of code layout by introducing artificial diversity

- **Control Flow Diversion**
  - **Control Flow Integrity**
    - Restricts the use of indirect branches against control flow hijacking

Address Space Layout Randomization
ROP Defenses

Two main approaches

- **Address Space Predictability**
  - **Randomization**
    - Breaks the knowledge of code layout by introducing artificial diversity

- **Control Flow Diversion**
  - **Control Flow Integrity**
    - Restricts the use of indirect branches against control flow hijacking
Two main approaches

- **Address Space Predictability**
  - **Randomization**
    - Breaks the knowledge of code layout by introducing artificial diversity

- **Control Flow Diversion**
  - **Control Flow Integrity**
    - Restricts the use of indirect branches against control flow hijacking
Previous Work: In-Place Randomization (IPR)

**Techniques**
- Instruction substitution
- Instruction reordering
- Register reassignment

**Advantages**
- Stripped binaries
- Practical for real apps
- Almost no overhead

**Assumptions**
- Incomplete CFG
- Inaccurate disassembly
- No code resizing

Can break 80% of the discovered gadgets!
## Previous Work: In-Place Randomization (IPR)

<table>
<thead>
<tr>
<th>Techniques</th>
<th>Advantages</th>
<th>Assumptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction substitution</td>
<td>Stripped binaries</td>
<td>Incomplete CFG</td>
</tr>
<tr>
<td>Instruction reordering</td>
<td>Practical for real apps</td>
<td>Inaccurate disassembly</td>
</tr>
<tr>
<td>Register reassignment</td>
<td>Almost no overhead</td>
<td>No code resizing</td>
</tr>
</tbody>
</table>

- Can break 80% of the discovered gadgets!
- The remaining gadgets (20%) may still be enough for the construction of a functional ROP payload!
Our Work

- Idea: breaking gadgets by displacing them
- Goal: maximize the gadget coverage on top of IPR
- Highly practical: can be applied on stripped binaries

Assume an adversary has the power of ROP:
- Functional payload with initial hijacking and memory disclosure
- Existing protections (DEP/ASLR) are enabled
- Attacker does not have access to the randomized binary
High Level View of Gadget Displacement
High Level View of Gadget Displacement

Pre-discovered Gadgets

Juggling the Gadgets
High Level View of Gadget Displacement

Pre-discovered Gadgets

Displaced Regions
High Level View of Gadget Displacement

Pre-discovered Gadgets

Basic Block (BBK)
Need *jmp* instructions

Displaced regions (>=5B)
*jmp* [rel-addr]
*int* 3
Intended vs. Unintended Gadgets
### Intended vs. Unintended Gadgets

**Basic Block**

**Gadgets for Displacement**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Assembly Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x82D6</td>
<td>E8 D2 FF FF FF</td>
<td>call sub_70082AD</td>
</tr>
<tr>
<td>0x82DB</td>
<td>C7 06 88 09 01 07</td>
<td>mov dword ptr [esi], offset 7010988</td>
</tr>
<tr>
<td>0x82E1</td>
<td>8B C6</td>
<td>mov eax, esi</td>
</tr>
<tr>
<td>0x82E3</td>
<td>5E</td>
<td>pop esi</td>
</tr>
<tr>
<td>0x82E4</td>
<td>C3</td>
<td>retn</td>
</tr>
</tbody>
</table>

---

**Juggling the Gadgets**
Intended vs. Unintended Gadgets

Basic Block
Gadgets for Displacement

.text:070082D6 E8 D2 FF FF FF call sub_70082AD
.text:070082DB C7 06 88 09 01 07 mov dword ptr [esi], offset 7010988
.text:070082E1 8B C6 mov eax, esi
.text:070082E3 5E pop esi
.text:070082E4 C3 retn

The first byte of each instruction
### Intended vs. Unintended Gadgets

**Basic Block**

**Gadgets for Displacement**

```
.text:070082D6  E8  D2  FF  FF  FF   call   sub_70082AD
.text:070082DB  C7  06  88  09  01  07  mov   dword ptr [esi], offset 7010988
.text:070082E1  8B  C6              mov   eax, esi
.text:070082E3  5E              pop   esi
.text:070082E4  C3              retn
```

**Pre-discovered Six Gadgets (2~10B in size)**

- The first byte of each instruction

---

**Juggling the Gadgets**

8
Intended vs. Unintended Gadgets

Basic Block
Gadgets for Displacement

```plaintext
.text:070082D6 E8 D2 FF FF FF  call  sub_70082AD
.text:070082DB C7 06 88 09 01 07  mov  dword ptr [esi], offset 7010988
.text:070082E1 8B C6  mov  eax, esi
.text:070082E3 5E  pop  esi
.text:070082E4 C3  ret
```

Pre-discovered Six Gadgets (2~10B in size)

The first byte of each instruction

Intended

G1
G5
G6

Juggling the Gadgets
Intended vs. Unintended Gadgets

### Basic Block

**Gadgets for Displacement**

- `.text:070082D6 E8 D2 FF FF FF`  
  `call sub 70082AD`
- `.text:070082DB C7 06 88 09 01 07`  
  `mov dword ptr [esi], offset 7010988`
- `.text:070082E1 8B C6`  
  `mov eax, esi`
- `.text:070082E3 5E`  
  `pop esi`
- `.text:070082E4 C3`  
  `retn`

### Pre-discovered Six Gadgets (2~10B in size)

- **E8 D2 FF FF FF**
  - The first byte of each instruction

### Intended

- **G1**
- **G2**
- **G3**
- **G4**
- **G5**
- **G6**

### Intended

- **C7 06 88 09 01 07 8B C6 5E C3**

### Unintended

- **js 0x32**
- `xor [edi], eax`
- `mov eax, esi`
- `pop esi`
- `retn`
Intended vs. Unintended Gadgets

Basic Block
Gadgets for Displacement

.text:070082D6 E8 D2 FF FF FF  call  sub 70082AD
.text:070082DB C7 06 88 09 01 07  mov  dword ptr [esi], offset 7010988
.text:070082E1 8B C6  mov  eax, esi
.text:070082E3 5E  pop  esi
.text:070082E4 C3  ret

Pre-discovered Six Gadgets (2~10B in size)

E8 D2 FF FF FF

The first byte of each instruction

C7 06 88 09 01 07 8B C6 5E C3

Intended

G1
G2
G3
G4
G5
G6

Unintended

✓ Nested in nature
Requirements for Displacement

- Maintain the original code semantics

  ✓ 5-byte long space to insert *jmp* instruction
  ✓ Recalculate code references
    - *branches* and *calls* with relative addresses
  ✓ Update all relocation entries
Displacement Strategy

- Paired jump instructions for every displacement?

- Keep the number of displaced regions low

- For unintended gadgets

- For intended gadgets

- Avoid generating the same binary
Displacement Strategy

✓ Paired jump instructions for every displacement?
   No ( Needless for unconditional “JMP” and “RET”)

✓ Keep the number of displaced regions low

✓ For unintended gadgets

✓ For intended gadgets

✓ Avoid generating the same binary
Displacement Strategy

✓ Paired jump instructions for every displacement?
  No (Needless for unconditional “JMP” and “RET”)

✓ Keep the number of displaced regions low
  Select the largest gadget to break the nested ones

✓ For unintended gadgets

✓ For intended gadgets

✓ Avoid generating the same binary
Displacement Strategy

✓ Paired jump instructions for every displacement?
  No (Needless for unconditional “JMP” and “RET”)

✓ Keep the number of displaced regions low
  Select the largest gadget to break the nested ones

✓ For unintended gadgets
  Find the starting byte of the first intended instruction of the gadget

✓ For intended gadgets

✓ Avoid generating the same binary
Displacement Strategy

✓ Paired jump instructions for every displacement?
  No (Needless for unconditional “JMP” and “RET”)

✓ Keep the number of displaced regions low
  Select the largest gadget to break the nested ones

✓ For unintended gadgets
  Find the starting byte of the first intended instruction of the gadget

✓ For intended gadgets
  Find the instruction all the way back in the same BBK

✓ Avoid generating the same binary
Displacement Strategy

✓ Paired jump instructions for every displacement?
  No (Needless for unconditional “JMP” and “RET”)

✓ Keep the number of displaced regions low
  Select the largest gadget to break the nested ones

✓ For unintended gadgets
  Find the starting byte of the first intended instruction of the gadget

✓ For intended gadgets
  Find the instruction all the way back in the same BBK

✓ Avoid generating the same binary
  Randomized placement of the displaced instructions
Displacement Algorithm

1. For Each Gadget
2. Apply IPR?
   - NO
   - YES
     - Displaced
       - NO
       - YES
         - BBK >= 5B
           - NO
           - YES
             - Select displacing regions
               - Recalculate references
               - Check relocation entries
               - Check for paired jumps
             - Intended
               - NO
               - YES
                 - Previous Instruction
                   - YES
                   - NO
                     - Intact
                   - NO
                     - Displace
11

Juggling the Gadgets
Displacement Example

Original

```
.text section
07002806 53  push ebx
07002807 FF1504000107 call ds:LeaveCriticalSection
0700280D 8D4704  lea eax,[edi+0x4]
07002810 5F  pop edi
07002811 5E  pop esi
07002812 5B  pop ebx
07002813 C3  ret
```

Displaced

```
.text section
07002806 E91CA00100  jmp loc_0701C827
07002808 CC  int3
...
07002813 CC  int3

.ropf section
0701C827 53  push ebx
0701C828 FF1504000107 call ds:LeaveCriticalSection
0701C82E 8D4704  lea eax,[edi+0x4]
0701C831 5F  pop edi
0701C832 5E  pop esi
0701C833 5B  pop ebx
0701C834 C3  ret
```

Nested Gadget
Displacement Example

Original

Displaced

Nested Gadget

Displaced Area

Juggling the Gadgets
PE adjustment: headers and sections

- NT HEADER
  - Optional Header
- SECTION HEADERS
  - ...
- SECTION DATA
  - ...
  - .reloc
PE adjustment: headers and sections

- NT HEADER
- Optional Header
- SECTION HEADERS
  - .ropf Section Header
- SECTION DATA
  - .reloc

VirtualSize, RVA, ...
Binary Instrumentation (1/2)

- PE adjustment: headers and sections

- NT HEADER
  - Optional Header

- SECTION HEADERS

- .ropf Section Header
  - VirtualSize, RVA, ...

- SECTION DATA

- .reloc

- .ropf

- Displaced Code Snippets
PE adjustment: headers and sections

- NT HEADER
  - Optional Header

- SECTION HEADERS
  - .ropf Section Header
    - VirtualSize, RVA, ...

- SECTION DATA
  - .reloc
  - .ropf

  - All Relocation Entries Updated
  - Displaced Code Snippets
PE adjustment: headers and sections

- NT HEADER
  - Optional Header
  - SizeOfCode, Checksum, ...
- SECTION HEADERS
  - VirtualSize, RVA, ...
- .ropf Section Header
  - All Relocation Entries Updated
- SECTION DATA
  - Displaced Code Snippets
Binary Instrumentation (1/2)

- PE adjustment: headers and sections

- NT HEADER
  - Optional Header
  - SizeOfCode, Checksum, ...

- SECTION HEADERS
  - Preserved Area
  - VirtualSize, RVA, ...

- .ropf Section Header
  - Preserved Area

- SECTION DATA
  - All Relocation Entries Updated
  - Displaced Code Snippets
- Rebuild the relocation table

<table>
<thead>
<tr>
<th>RVA of Block</th>
<th>Size of Block</th>
<th>Type</th>
<th>RVA</th>
<th>Type</th>
<th>RVA</th>
</tr>
</thead>
<tbody>
<tr>
<td>3FC7</td>
<td></td>
<td>Type RVA</td>
<td>00001FC7</td>
<td>IMAGE_REL_BASED_HIGHLOW</td>
<td></td>
</tr>
<tr>
<td>3FE5</td>
<td></td>
<td>Type RVA</td>
<td>00001FE5</td>
<td>IMAGE_REL_BASED_HIGHLOW</td>
<td></td>
</tr>
<tr>
<td>00000</td>
<td></td>
<td>Type RVA</td>
<td>00000000</td>
<td>IMAGE_REL_BASED_HIGHLOW</td>
<td></td>
</tr>
<tr>
<td>00002000</td>
<td></td>
<td>RVA of Block</td>
<td>00002000</td>
<td>IMAGE_REL_BASED_HIGHLOW</td>
<td></td>
</tr>
<tr>
<td>00000D8</td>
<td></td>
<td>Size of Block</td>
<td>00000D8</td>
<td>IMAGE_REL_BASED_HIGHLOW</td>
<td></td>
</tr>
<tr>
<td>3046</td>
<td></td>
<td>Type RVA</td>
<td>00002046</td>
<td>IMAGE_REL_BASED_HIGHLOW</td>
<td></td>
</tr>
<tr>
<td>3066</td>
<td></td>
<td>Type RVA</td>
<td>00002066</td>
<td>IMAGE_REL_BASED_HIGHLOW</td>
<td></td>
</tr>
<tr>
<td>3071</td>
<td></td>
<td>Type RVA</td>
<td>00002071</td>
<td>IMAGE_REL_BASED_HIGHLOW</td>
<td></td>
</tr>
<tr>
<td>3076</td>
<td></td>
<td>Type RVA</td>
<td>00002076</td>
<td>IMAGE_REL_BASED_HIGHLOW</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3870</td>
<td></td>
<td>Type RVA</td>
<td>00018870</td>
<td>IMAGE_REL_BASED_HIGHLOW</td>
<td></td>
</tr>
<tr>
<td>0001C000</td>
<td></td>
<td>RVA of Block</td>
<td>0001C000</td>
<td>IMAGE_REL_BASED_HIGHLOW</td>
<td></td>
</tr>
<tr>
<td>00000B4</td>
<td></td>
<td>Size of Block</td>
<td>00000B4</td>
<td>IMAGE_REL_BASED_HIGHLOW</td>
<td></td>
</tr>
<tr>
<td>3002</td>
<td></td>
<td>Type RVA</td>
<td>0001C002</td>
<td>IMAGE_REL_BASED_HIGHLOW</td>
<td></td>
</tr>
<tr>
<td>304C</td>
<td></td>
<td>Type RVA</td>
<td>0001C04C</td>
<td>IMAGE_REL_BASED_HIGHLOW</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Multiple relocation blocks
- Total number of all entries should be identical
Binary Instrumentation (2/2)

- Rebuild the relocation table

<table>
<thead>
<tr>
<th>Type RVA</th>
<th>RVA of Block</th>
<th>Size of Block</th>
<th>Type RVA</th>
<th>RVA of Block</th>
<th>Size of Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>3FC7</td>
<td>000001FC7 IMAGE_REL_BASED_HIGHLOW</td>
<td>000000D8</td>
<td>Type RVA</td>
<td>00002046 IMAGE_REL_BASED_HIGHLOW</td>
<td>3046</td>
</tr>
<tr>
<td>3FE5</td>
<td>000001FE5 IMAGE_REL_BASED_HIGHLOW</td>
<td>3066</td>
<td>Type RVA</td>
<td>00002066 IMAGE_REL_BASED_HIGHLOW</td>
<td>3071</td>
</tr>
<tr>
<td>0000</td>
<td>Type RVA</td>
<td>00002071 IMAGE_REL_BASED_HIGHLOW</td>
<td>3076</td>
<td>Type RVA</td>
<td>00002076 IMAGE_REL_BASED_HIGHLOW</td>
</tr>
<tr>
<td>3870</td>
<td>Type RVA</td>
<td>000018870 IMAGE_REL_BASED_HIGHLOW</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>0001C000</td>
<td>RVA of Block</td>
<td>000000B4</td>
<td>Type RVA</td>
<td>0001C002 IMAGE_REL_BASED_HIGHLOW</td>
<td>3002</td>
</tr>
<tr>
<td>304C</td>
<td>Type RVA</td>
<td>0001C040 IMAGE_REL_BASED_HIGHLOW</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

- Multiple relocation blocks
- Total number of all entries should be identical
Evaluation – Dataset

- 2,695 samples from Windows 7, 8.1 and benign apps

<table>
<thead>
<tr>
<th>Applications</th>
<th>Files</th>
<th>Total</th>
<th>Unintended</th>
<th>Unreachable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adobe Reader</td>
<td>50</td>
<td>677,689</td>
<td>55.24%</td>
<td>4.61%</td>
</tr>
<tr>
<td>MS Office 2013</td>
<td>18</td>
<td>195,774</td>
<td>55.04%</td>
<td>4.93%</td>
</tr>
<tr>
<td>Windows 7</td>
<td>1,224</td>
<td>5,595,031</td>
<td>53.97%</td>
<td>6.11%</td>
</tr>
<tr>
<td>Windows 8.1</td>
<td>1,341</td>
<td>6,077,543</td>
<td>63.46%</td>
<td>6.90%</td>
</tr>
<tr>
<td>Various</td>
<td>62</td>
<td>496,749</td>
<td>55.15%</td>
<td>5.79%</td>
</tr>
<tr>
<td>Total</td>
<td>2,695</td>
<td>13,042,786</td>
<td>58.52%</td>
<td>6.37%</td>
</tr>
</tbody>
</table>
Evaluation – Gadget Coverage (1/2)

- **Broken gadgets by displacement and IPR**

  ![Diagram showing gadget coverage with percentages](image)

  - **Displacement**
    - M-R: 12.27% (11.49%)
    - M+R: 90.04% (84.31%)
    - R-M: 77.78% (72.82%)
    - Unreachable: 6.37%
    - Unbroken: 2.59%
  - **In-Place Rand.**
    - M-R: 84.96% (79.55%)
    - M+R: 97.23% (91.04%)
    - R-M: 7.19% (6.73%)

  **All Gadgets**

  - 97.23% (91.04%)
Evaluation – Gadget Coverage (1/2)

- **Broken gadgets by displacement and IPR**

![Venn Diagram]

- **All Gadgets**
  - Displacement
    - M-R: 12.27% (11.49%)
    - M+R: 90.04% (84.31%)
  - In-Place Rand.
    - M&R: 77.78% (72.82%)
    - R-M: 7.19% (6.73%)
  - Unreachable: 6.37%
  - Unbroken: 2.59%

- **Coverage:** 97.23% (91.04%)
Evaluation – Gadget Coverage (2/2)

❖ Cumulative distribution of randomized gadgets

![Graph showing cumulative distribution of randomized gadgets](image)
Evaluation – Runtime Overhead

- **SPEC2006**: 0.36% average overhead
- Statistical *t*-test shows no significant difference for negative overheads
Limitations

✓ Number of gadgets that can be displaced still depends on the coverage of disassembly and CFG extraction

✓ Gadget displacement needs at least 5 bytes

✓ Cannot defend against JIT-ROP

✓ Cannot break entry-point gadgets (less than 1%)
Wrap-up

✓ Presented a novel approach: gadget displacement

✓ Broken gadget coverage: 85% → 97%

✓ Practical: no source code or debug symbols requirement

✓ Negligible overhead: 0.36%

Code available:
https://github.com/kevinkoo001/ropf