Just-in-Time Code Reuse

The more things change, the more they stay the same

Kevin Z. Snow¹
Luca Davi²

&

A. Dmitrienko²  C. Liebchen²  F. Monrose¹  A.-R. Sadeghi²

¹ Department of Computer Science
University of North Carolina at Chapel Hill

² CASED/Technische Universität
Darmstadt, Germany
The Big Picture
The Big Picture

- Scripting facilitates attacks
The Big Picture

- Scripting facilitates attacks

Large attack surface
The Big Picture

- Scripting facilitates attacks
- Exploit packs automate increasingly complex attacks
- Adversary must apply a code-reuse strategy

Large attack surface
The Big Picture

Daily Blog Tips awarded the

Last week Darren Rowse, the Daily Blog Tips is
from the famous Ren
ProBlogger blog, attracting a vast audience
announced the winners of following blogs. When asked about
his latest Group Writing looking to improve their
Project called "Reviews blogs. When asked about
and Predictions". Among the success of his blog
Daniel commented that
the
The Big Picture

The New York Times

Saturday, January 6, 2007

Daily Blog Tips awarded the

Last week Darren Rowse, from the famous
ProBlogger blog, announced the winners of
his latest Group Writing Project called "Reviews
and Predictions". Among

Daniel commented that

The Daily Blog Tips is...
The Big Picture

Last week Darren Rowse, the Daily Blog Tips is Ren from the famous attracting a vast audience from the blog of ProBlogger blog of bloggers who are announced the winners of looking to improve their Project called ’Reviews the success of his blog that and Predictions’. Among Daniel commented that the return oriented programming
Basic ROP Attack Technique
Basic ROP Attack Technique

Adversary
Basic ROP Attack Technique

Adversary

Stack

Heap

Code
Basic ROP Attack Technique

Adversary

Stack

Heap

Code

- Stack Pivot
- LOAD Gadget
- ADD Gadget

RET
Basic ROP Attack Technique

Adversary

Stack
- Stack Var 1
- Stack Var 2

Heap

Code
- Stack Pivot RET
- LOAD Gadget RET
- ADD Gadget RET
Basic ROP Attack Technique

Adversary

Stack
- Stack Var 1
- Stack Var 2

SP

Heap

Code
- Stack Pivot
- LOAD Gadget
- ADD Gadget

RET

Thursday, August 1, 13
Basic ROP Attack Technique

Adversary

Inject ROP Payload

Stack

SP

Stack Var 1
Stack Var 2

Heap

RET Address 3
RET Address 2
RET Address 1

Code

Stack Pivot RET
LOAD Gadget RET
ADD Gadget RET
Basic ROP Attack Technique

Adversary

Inject ROP Payload

Stack
- Stack Var 1
- Stack Var 2

SP

Heap
- Heap Vulnerability
- RET Address 3
- RET Address 2
- RET Address 1

Code
- Stack Pivot
- LOAD Gadget
- ADD Gadget
- RET
Basic ROP Attack Technique

Adversary

Exploit Vulnerability to Launch ROP Payload

Stack

Stack Var 1
Stack Var 2

SP

Heap

Heap Vulnerability

RET Address 3
RET Address 2
RET Address 1

Code

Stack Pivot
LOAD Gadget
ADD Gadget

RET
RET
RET
Basic ROP Attack Technique

Adversary

Exploit Vulnerability to Launch ROP Payload

Stack
- Stack Var 1
- Stack Var 2

Heap
- Heap Vulnerability
- RET Address 3
- RET Address 2
- RET Address 1

Code
- Stack Pivot
- LOAD Gadget
- ADD Gadget
Basic ROP Attack Technique

Adversary

Exploit Vulnerability to Launch ROP Payload

Stack
- Stack Var 1
- Stack Var 2

Heap
- Heap Vulnerability
  - RET Address 3
  - RET Address 2
  - RET Address 1

Code
- Stack Pivot
- LOAD Gadget
- ADD Gadget

RET
Basic ROP Attack Technique

Adversary

Exploit Vulnerability to Launch ROP Payload

Stack
- Stack Var 1
- Stack Var 2

Heap
- Heap Vulnerability
- RET Address 3
- RET Address 2
- RET Address 1

SP

Code
- Stack Pivot
- LOAD Gadget
- ADD Gadget

RET Address 1
RET Address 2
RET Address 3
Basic ROP Attack Technique

Adversary

Exploit Vulnerability to Launch ROP Payload

Stack

- Stack Var 1
- Stack Var 2

Heap

- Heap Vulnerability

SP

- RET Address 3
- RET Address 2
- RET Address 1

Code

- Stack Pivot
- LOAD Gadget
- ADD Gadget
Basic ROP Attack Technique

Adversary

Stack
- Stack Var 1
- Stack Var 2

Heap
- Heap Vulnerability
- RET Address 3
- RET Address 2
- RET Address 1

SP

Code
- Stack Pivot
- LOAD Gadget
- ADD Gadget

Exploit Vulnerability to Launch ROP Payload
Code Reuse Attacks History

*selected not exhaustive*

1997

2001

2005

2007

2008

2009

2010
## Code Reuse Attacks History

*selected not exhaustive*

<table>
<thead>
<tr>
<th>Year</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1997</td>
<td>ret2libc, Solar Designer</td>
</tr>
<tr>
<td>2001</td>
<td></td>
</tr>
<tr>
<td>2005</td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td></td>
</tr>
<tr>
<td>2008</td>
<td></td>
</tr>
<tr>
<td>2009</td>
<td></td>
</tr>
<tr>
<td>2010</td>
<td></td>
</tr>
</tbody>
</table>
# Code Reuse Attacks History

*selected not exhaustive*

<table>
<thead>
<tr>
<th>Year</th>
<th>Attack Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1997</td>
<td>ret2libc&lt;br&gt;Solar Designer</td>
</tr>
<tr>
<td>2001</td>
<td>Advanced ret2libc&lt;br&gt;Nergal</td>
</tr>
<tr>
<td>2005</td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td></td>
</tr>
<tr>
<td>2008</td>
<td></td>
</tr>
<tr>
<td>2009</td>
<td></td>
</tr>
<tr>
<td>2010</td>
<td></td>
</tr>
</tbody>
</table>
Code Reuse Attacks History

*selected not exhaustive*

- **1997**
  - *ret2libc*
  - Solar Designer

- **2001**
  - *Advanced ret2libc*
  - Nergal

- **2005**
  - *Borrowed Code Chunks Exploitation*
  - Krahmer

- **2007**

- **2008**

- **2009**

- **2010**
Code Reuse Attacks History

selected not exhaustive

- 1997
  - ret2libc
  - Solar Designer

- 2001
  - Advanced ret2libc
  - Nergal

- 2005
  - Borrowed Code Chunks Exploitation
  - Krahmer

- 2007
  - ROP
  - Shacham (CCS)

- 2008

- 2009

- 2010
Code Reuse Attacks History

selected not exhaustive

1997
- **ret2libc**
  - Solar Designer

2001
- **Advanced ret2libc**
  - Nergal

2005
- **Borrowed Code Chunks Exploitation**
  - Krahmer

2007
- **ROP**
  - Shacham (CCS)

2008
- **ROP on SPARC**
  - Buchanan et al (CCS)
- **ROP on Atmel**
  - Francillon et al (CCS)
- **ROP**
  - Shacham (BlackHat USA)

2009

2010

1997

2001

2005

2007

2008

2009

2010
Code Reuse Attacks History

**selected not exhaustive**

1997
- ret2libc
  - Solar Designer

2001
- Advanced ret2libc
  - Nergal

2005
- Borrowed Code Chunks Exploitation
  - Krahmer

2007
- ROP
  - Shacham (CCS)

2008
- ROP on SPARC
  - Buchanan et al (CCS)
- ROP on Atmel
  - Francillon et al (CCS)
- ROP
  - Shacham (BlackHat USA)

2009
- ROP Rootkits
  - Hund et al (USENIX)
- ROP on PowerPC
  - FX Lindner (BlackHat USA)
- ROP on ARM/iOS
  - Miller et al (BlackHat USA)

2010
Code Reuse Attacks History

selected not exhaustive

1997
- **ret2libc**
  - Solar Designer

2001
- **Advanced ret2libc**
  - Nergal

2005
- **Borrowed Code Chunks Exploitation**
  - Krahmer

2007
- **ROP**
  - Shacham (CCS)

2008
- **ROP on SPARC**
  - Buchanan et al (CCS)
- **ROP on Atmel**
  - Francillon et al (CCS)
- **ROP**
  - Shacham (BlackHat USA)

2009
- **ROP Rootkits**
  - Hund et al (USENIX)
- **ROP on PowerPC**
  - FX Lindner (BlackHat USA)
- **ROP on ARM/iOS**
  - Miller et al (BlackHat USA)

2010
- **ROP without Returns**
  - Checkoway et al (CCS)
- **Ropportunity**
  - Iozzo et al (BlackHat USA)
- **Payload already inside**
  - Long Le (BlackHat USA)
- **Pwn2Own iPhone**
  - Weinmann & Iozzo
- **Pwn2Own IE**
  - Nils
- **Practical ROP**
  - Zovi (RSA Conference)
ASLR – Address Space Layout Randomization
Basics of ASLR

- ASLR randomizes the base address of code/data segments
Basics of ASLR

- ASLR randomizes the base address of code/data segments
Basics of ASLR

• ASLR randomizes the base address of code/data segments

1. Exploit disclosure vulnerability
2. Leak function pointer
3. Adjust instruction sequence pointers

Disclosure Attack e.g., [Sotirov et al., Blackhat 2008]
Example Memory Disclosure

See [Serna, Blackhat USA 2012] for more memory disclosure tactics.
Example Memory Disclosure

See [Serna, Blackhat USA 2012] for more memory disclosure tactics.
Example Memory Disclosure

Vulnerable Object | JavaScript String | Object
---|---|---
AAAAAAAAAAAAAAAAAAAAAAA | MAX_SIZE | string data
| | funcPointer | funcPointer

Library (e.g., user32.dll)

Program Memory (abstract)

See [Serna, Blackhat USA 2012] for more memory disclosure tactics.
Example Memory Disclosure

See [Serna, Blackhat USA 2012] for more memory disclosure tactics.
Tackling the Problems of ASLR via

*Fine-Grained ASLR*
### Basics of Fine-grained ASLR

#### Application Run 1

<table>
<thead>
<tr>
<th>Library (e.g., user32.dll)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction Sequence 1</td>
</tr>
<tr>
<td>RET</td>
</tr>
<tr>
<td>Instruction Sequence 2</td>
</tr>
<tr>
<td>RET</td>
</tr>
<tr>
<td>Instruction Sequence 3</td>
</tr>
<tr>
<td>RET</td>
</tr>
</tbody>
</table>
Basics of Fine-grained ASLR

• Different fine-grained ASLR approaches have been proposed recently
  • ORP [Pappas et al., IEEE Security & Privacy 2012]
  • ILR [Hiser et al., IEEE Security & Privacy 2012]
  • STIR [Wartell et al., ACM CCS 2012]
  • XIFER [Davi et al., ASIACCS 2013]
  • All mitigate single memory disclosure attacks
Inner Basic Block Randomization

[Pappas et al., IEEE S&P 2012]

• Instruction Reordering

Original

```plaintext
MOV EBX, &ptr
MOV EAX, &string
```

Randomized
Inner Basic Block Randomization

[Pappas et al., IEEE S&P 2012]

- Instruction Reordering

**Original**

- `MOV EBX, &ptr`
- `MOV EAX, &string`

**Randomized**

- `MOV EAX, &string`
- `MOV EBX, &ptr`
Inner Basic Block Randomization

[Pappas et al., IEEE S&P 2012]

• Instruction Reordering

Original

- MOV EBX, &ptr
- MOV EAX, &string

Randomized

- MOV EAX, &string
- MOV EBX, &ptr

• Instruction Substitution

Original

- MOV EBX, $0

Randomized
Inner Basic Block Randomization

[Pappas et al., IEEE S&P 2012]

- Instruction Reordering

Original

$\textbf{MOV} \ EBX, \ &\text{ptr}$  
$\textbf{MOV} \ EAX, \ &\text{string}$

Randomized

$\textbf{MOV} \ EAX, \ &\text{string}$  
$\textbf{MOV} \ EBX, \ &\text{ptr}$

- Instruction Substitution

Original

$\textbf{MOV} \ EBX, \$0$

Randomized

$\textbf{XOR} \ EBX,EBX$
Inner Basic Block Randomization

[Pappas et al., IEEE S&P 2012]

- Instruction Reordering

Original

- MOV EBX, &ptr
- MOV EAX, &string

Randomized

- MOV EAX, &string
- MOV EBX, &ptr

- Instruction Substitution

Original

- MOV EBX, $0

Randomized

- XOR EBX, EBX

- Register Re-Allocation (in case another register is free to use)

Original

- MOV EAX, &ptr
- CALL *EAX

Randomized


Inner Basic Block Randomization

[Pappas et al., IEEE S&P 2012]

- Instruction Reordering

Original

\[
\begin{align*}
\text{MOV} & \text{ EBX, } \&\text{ptr} \\
\text{MOV} & \text{ EAX, } \&\text{string}
\end{align*}
\]

Randomized

\[
\begin{align*}
\text{MOV} & \text{ EAX, } \&\text{string} \\
\text{MOV} & \text{ EBX, } \&\text{ptr}
\end{align*}
\]

- Instruction Substitution

Original

\[
\begin{align*}
\text{MOV} & \text{ EBX, } \$0
\end{align*}
\]

Randomized

\[
\begin{align*}
\text{XOR} & \text{ EBX,EBX}
\end{align*}
\]

- Register Re-Allocation (in case another register is free to use)

Original

\[
\begin{align*}
\text{MOV} & \text{ EAX, } \&\text{ptr} \\
\text{CALL} & \text{ *EAX}
\end{align*}
\]

Randomized

\[
\begin{align*}
\text{MOV} & \text{ EBX, } \&\text{ptr} \\
\text{CALL} & \text{ *EBX}
\end{align*}
\]
Basic Block Randomization

[Wartell et al., ACM CCS 2012; Davi et al. AsiaCCS 2013]
Basic Block Randomization

[Wartell et al., ACM CCS 2012; Davi et al. AsiaCCS 2013]

Original

BBL_1

BBL_2

BBL_3
Basic Block Randomization
[Wartell et al., ACM CCS 2012; Davi et al. AsiaCCS 2013]

Original

BBL_1

**MOV** EBX, EAX
**CALL** 0x10FF

BBL_2

**MOV** (ESP), EAX
**RET**

BBL_3

**ADD** EAX, ECX
**RET**

0x10FF:
Basic Block Randomization

[Wartell et al., ACM CCS 2012; Davi et al. AsiaCCS 2013]

Original

BBL_1

**MOV** EBX, EAX
**CALL** 0x10FF

BBL_2

**MOV** (ESP), EAX
**RET**

BBL_3

**ADD** EAX, ECX
**RET**

0x10FF:
Basic Block Randomization

[Wartell et al., ACM CCS 2012; Davi et al. AsiaCCS 2013]

Original

- BBL_1
  - MOV EBX, EAX
  - CALL 0x10FF

- BBL_2
  - MOV (ESP), EAX
  - RET

- BBL_3
  - ADD EAX, ECX
  - RET

Randomized

0x10FF:
Basic Block Randomization
[Wartell et al., ACM CCS 2012; Davi et al. AsiaCCS 2013]

Original

BBL_1

MOV EBX, EAX
CALL 0x10FF

BBL_2

MOV (ESP), EAX
RET

BBL_3

ADD EAX, ECX
RET

Randomized

BBL_2

0x1000:

BBL_3

0x10A0:

BBL_1

0x10FF:
Basic Block Randomization
[Wartell et al., ACM CCS 2012; Davi et al. AsiaCCS 2013]

Original

BBL_1

MOV EBX, EAX
CALL 0x10FF

BBL_2

MOV (ESP), EAX
RET

BBL_3

ADD EAX, ECX
RET

Randomized

BBL_2

0x1000:

BBL_3

0x10A0:

BBL_1

0x10FF:

MOV EBX, EAX
CALL 0x10A0
Basic Block Randomization
[Wartell et al., ACM CCS 2012; Davi et al. AsiaCCS 2013]

Original

- BBL_1
  - MOV EBX, EAX
  - CALL 0x10FF

- BBL_2
  - MOV (ESP), EAX
  - RET

- BBL_3
  - ADD EAX, ECX
  - RET

Randomized

- BBL_2

- BBL_3
  - ADD EAX, ECX
  - RET

- BBL_1
  - MOV EBX, EAX
  - CALL 0x10A0

0x10FF:

- BBL_1
  - MOV EBX, EAX
  - CALL 0x10A0

0x1000:

- BBL_2

0x10A0:

- BBL_3
Basic Block Randomization
[Wartell et al., ACM CCS 2012; Davi et al. AsiaCCS 2013]

Original

BBL_1

MOV EBX, EAX
CALL 0x10FF

BBL_2

MOV (ESP), EAX
RET

BBL_3

ADD EAX, ECX
RET

Randomized

BBL_2

BBL_3

ADD EAX, ECX
RET

BBL_1

MOV EBX, EAX
CALL 0x10A0
JMP 0x1000

0x10FF:

0x1000:

0x10A0:
Basic Block Randomization
[Wartell et al., ACM CCS 2012; Davi et al. AsiaCCS 2013]

Original

- BBL_1
  - MOV EBX, EAX
  - CALL 0x10FF

- BBL_2
  - MOV (ESP), EAX
  - RET

- BBL_3
  - ADD EAX, ECX
  - RET

Randomized

- BBL_2
  - MOV (ESP), EAX
  - RET

- BBL_3
  - ADD EAX, ECX
  - RET

- BBL_1
  - MOV EBX, EAX
  - CALL 0x10A0
  - JMP 0x1000

0x10FF:

0x1000:

0x10A0:
Instruction Location Randomization
[Hiser et al., IEEE S&P 2012]

Original

MOV EBX, EAX
CALL 0x10FF
MOV (ESP), EAX
RET

0x10FF:
ADD EAX, ECX
RET
Instruction Location Randomization

[Hiser et al., IEEE S&P 2012]

**Original**

- MOV EBX, EAX
- CALL 0x10FF
- MOV (ESP), EAX
- RET

0x10FF:

- ADD EAX, ECX
- RET

**Randomized**

0x1000:

- MOV (ESP), EAX

0x12A0:

- RET

0x1F00:

- RET

0x2000:

- CALL 0x3000

0x2500:

- MOV EBX, EAX

0x3000:

- ADD EAX, ECX
**Instruction Location Randomization**

[Hiser et al., IEEE S&P 2012]

**Original**

- **MOV EBX, EAX**
- **CALL 0x10FF**
- **MOV (ESP), EAX**
- **RET**

**Randomized**

- **MOV (ESP), EAX**
- **RET**
- **RET**
- **CALL 0x3000**
- **MOV EBX, EAX**
- **ADD EAX, ECX**

Execution is driven by a fall-through map and a binary translation framework (Strata)
Does Fine-Grained ASLR Provide a Viable Defense in the Long Run?
A novel attack class that undermines fine-grained ASLR, dubbed *just-in-time code reuse*
Contributions

1. A novel attack class that undermines fine-grained ASLR, dubbed *just-in-time code reuse*.

2. We show that *memory disclosures* are far more damaging than previously believed.
Contributions

1. A novel attack class that undermines fine-grained ASLR, dubbed *just-in-time code reuse*

2. We show that *memory disclosures* are far more damaging than previously believed

3. A prototype exploit framework that demonstrates one instantiation of our idea, called *JIT-ROP*
Assumptions

Adversary

Defender
Assumptions

Adversary

Defender

Non-Executable Stack and Heap
Assumptions

Adversary

Defender

- Fine-Grained ASLR
- Non-Executable Stack and Heap
Assumptions

Adversary

- Memory Disclosure Vulnerability

Defender

- Fine-Grained ASLR
- Non-Executable Stack and Heap
Assumptions

Adversary

- Control-Flow Vulnerability
- Memory Disclosure Vulnerability
- Fine-Grained ASLR
- Non-Executable Stack and Heap

Defender
Workflow of Just-In-Time Code Reuse

Adversary
Workflow of Just-In-Time Code Reuse

Adversary

Leak Code Pointer
Workflow of Just-In-Time Code Reuse

Adversary

→

Leak Code Pointer

→

Exploit Description
   (High-Level Language)
Workflow of Just-In-Time Code Reuse

Adversary

Leak Code Pointer

Exploit Description
(High-Level Language)

Vulnerable Application
Workflow of Just-In-Time Code Reuse

Adversary

Leak Code Pointer

Exploit Description (High-Level Language)

Vulnerable Application

JIT-ROP Framework

Map Memory
Workflow of Just-In-Time Code Reuse

Adversary

Leak Code Pointer

Exploit Description (High-Level Language)

Vulnerable Application

JIT-ROP Framework

Map Memory

Find ROP Sequences (Gadgets)
Workflow of Just-In-Time Code Reuse

Adversary

Leak Code Pointer

Exploit Description (High-Level Language)

Vulnerable Application

JIT-ROP Framework

Map Memory

Find ROP Sequences (Gadgets)

Find API Functions
Workflow of Just-In-Time Code Reuse

Adversary

Leak Code Pointer

Exploit Description (High-Level Language)

Vulnerable Application

JIT-ROP Framework

Map Memory

Find ROP Sequences (Gadgets)

Compile ROP Program

Find API Functions
Workflow of Just-In-Time Code Reuse

Adversary

Leak Code Pointer

Exploit Description (High-Level Language)

Vulnerable Application

JIT-ROP Framework

Map Memory

Find ROP Sequences (Gadgets)

Compile ROP Program

Find API Functions

17 Thursday, August 1, 13
Challenges

Map memory without crashing
Challenges

Map memory without crashing

Find gadgets, APIs, and compile payload dynamically at runtime
Challenges

- Map memory without crashing
- Find gadgets, APIs, and compile payload dynamically at runtime
- Fully automated
Challenges

Map memory without crashing

Find gadgets, APIs, and compile payload dynamically at runtime

Fully automated

Demonstrate efficient, practical exploit
Our Approach

- Map Memory
- Find API Calls
- Find Gadgets
- JIT Compile

**Observation:**

single leaked function pointer $\Rightarrow$ an entire code page is present
Our Approach

Map Memory  Find API Calls  Find Gadgets  JIT Compile

observation:

single leaked function pointer $\Rightarrow$ an entire code page is present

f295afcad42b43
638b2bbf6381ff
72efc88bda4cc0
0732bba1575ccb
eb7c025e6b8ad3
0c283baa9f03e4
7464fc814176cd
546bcee28e4232

initial code page
Our Approach

Map Memory  Find API Calls  Find Gadgets  JIT Compile

**observation:**

single leaked function pointer $\Rightarrow$ an entire code page is present

```
... 
push 0x1
\textbf{call} [-0xFEED]
mov ebx, eax
\textbf{jmp} +0xBEEF
dec ecx
xor ebx, ebx
... 
```

initial code page
Our Approach

Map Memory  Find API Calls  Find Gadgets  JIT Compile

**observation:**

single leaked function pointer $\Rightarrow$ an entire code page is present

... push 0x1
**call** [-0xFEED]
mov ebx, eax
**jmp** +0xBEEF
dec ecx
xor ebx, ebx
...

initial code page
Our Approach

Map Memory  Find API Calls  Find Gadgets  JIT Compile

**observation:**

single leaked function pointer  \(\Rightarrow\) an entire code page is present

... push 0x1 call [-0xFEED] mov ebx, eax jmp +0xBEEF dec ecx xor ebx, ebx ...

initial code page
Our Approach

- Map Memory
- Find API Calls
- Find Gadgets
- JIT Compile

Thursday, August 1, 13
Our Approach

Map Memory Find API Calls Find Gadgets JIT Compile

Desired Payload

```
URLDownloadToFile("http://...", "bot.exe");
WinExec("bot.exe");
ExitProcess(1);
```
Our Approach

Map Memory
Find API Calls
Find Gadgets
JIT Compile

Desired Payload

```
URLDownloadToFile("http://...", "bot.exe");
WinExec("bot.exe");
ExitProcess(1);
```
Our Approach

Desired Payload

```c
URLDownloadToFile("http://...", "bot.exe");
WinExec("bot.exe");
ExitProcess(1);
```

- needed APIs often not referenced by program

Vulnerable Application

- Code Page Previously Found
  - Sleep("
  - FindWindow("
  - GetActiveWindow("

Map Memory

Find API Calls

Find Gadgets

JIT Compile
Our Approach

- Map Memory
- Find API Calls
- Find Gadgets
- JIT Compile

Desired Payload

```c
URLDownloadToFile("http://...", "bot.exe");
WinExec("bot.exe");
ExitProcess(1);
```

- needed APIs often not referenced by program
- dynamic library and function loading is common
- solution: scan for `LoadLibrary` and `GetProcAddress` references instead

Vulnerable Application

- Code Page Previously Found
- `LoadLibrary("library.dll")`
- `GetProcAddress("func1")`
- `GetProcAddress("func2")`
Our Approach

Desired Payload

```
URLDownloadToFile("http://...", "bot.exe");
WinExec("bot.exe");
ExitProcess(1);
```

- needed APIs often not referenced by program
- dynamic library and function loading is common
- solution: scan for `LoadLibrary` and `GetProcAddress` references instead

With Dynamic Loading

```
LoadLibrary("urlmon.dll");
GetProcAddress(@, "URLDownloadToFile");
@("http://...", "bot.exe");
LoadLibrary("kernel32.dll");
GetProcAddress(@, "WinExec");
@("bot.exe");
...
```
Our Approach

- Map Memory
- Find API Calls
- Find Gadgets
- JIT Compile
Our Approach

Map Memory  Find API Calls  Find Gadgets  JIT Compile

code pages

Find Gadgets

Map Memory  JIT Compile

JIT Compile

Find API Calls

...
Our Approach

Map Memory
Find API Calls
Find Gadgets
JIT Compile

code pages

code sequences

mov ebx, eax
ret

pop eax
mov [ecx], eax
ret

pop eax
mov ebx, edx
ret

pop eax
mov ebx, edx
ret

mov eax, 0x14
...

Galileo Algorithm
[Schacham, ACM CCS 2007]

...
Our Approach

Map Memory Find API Calls Find Gadgets JIT Compile

code pages code sequences gadget types

mov ebx, eax ret
mov [ecx], eax ret
mov ebx, edx pop eax ret
mov eax, 0x14 ret
... MovRegG JumpG ArithmeticG LoadRegG ...

push 0x1 call [-0xFEED]
mov ebx, eax jmp +0xBEEF
dec ecx xor ebx, ebx
... Galileo Algorithm [Schacham, ACM CCS 2007]

... gadgets found

Find Gadgets
Our Approach

Map Memory
Find API Calls
Find Gadgets
JIT Compile

code pages

code sequences

mov ebx, eax
ret

pop eax
mov [ecx], eax
ret

pop eax
mov ebx, edx
pop eax
ret

mov eax, 0x14

MovRegG
JumpG
ArithmeticG
LoadRegG

gadget types

Galileo Algorithm
[Schacham, ACM CCS 2007]

gadgets found
Our Approach

Map Memory  Find API Calls  Find Gadgets  JIT Compile

code pages

code sequences

gadget types

mov ebx, eax  ret

mov [ecx], eax  ret

mov ebx, edx  pop eax  pop eax  pop eax

mov ebx, eax  jmp

call

MovRegG  JumpG  ArithmeticG  LoadRegG

... 

galileo Algorithm

[Schacham, ACM CCS 2007]
Our Approach

Map Memory  Find API Calls  Find Gadgets  JIT Compile

code pages  code sequences  gadget types

mov ebx, eax  ret

pop eax

mov [ecx], eax  ret

mov ebx, edx  pop eax  pop eax

mov ebx, edx  ret

push 0x1  call [-0xFEED]  mov ebx, eax  jmp

dec ecx  xor ebx, ebx

Galileo Algorithm
[Schacham, ACM CCS 2007]

gadgets found
Our Approach

Map Memory  Find API Calls  Find Gadgets  JIT Compile

code pages  code sequences  gadget types

mov ebx, eax  ret
mov ebx, edx  pop eax

mov ebx, edx  ret

load 0x1  call [-0xFEED]
mov ebx, eax  jmp +0x0BEEF
dec ecx  xor ebx, ebx

Galileo Algorithm
[Schacham, ACM CCS 2007]

mov ebx, eax  jmp +0xBEEF  dec ecx  xor ebx, ebx

gadgets found

MovRegG  JumpG  ArithmeticG  LoadRegG
...
Our Approach

Map Memory  Find API Calls  Find Gadgets  JIT Compile

code pages  code sequences  gadget types

mov ebx, eax  ret

mov ebx, edx  pop eax

...  was used for gadget types

push 0x1  call -0xFEED  mov ebx, eax  jmp +0xBEEF  dec ecx  xor ebx, ebx  ...

MovRegG  JumpG  ArithmeticG  LoadRegG

...  Galileo Algorithm

[Schacham, ACM CCS 2007]

gadgets found
Compiling the ROP program
Compiling the ROP program

our high-level language

LoadLibrary("kernel32");
GetProcAddress(@, "WinExec");
@("calc", SW_SHOWNORMAL);
LoadLibrary("kernel32");
GetProcAddress(@, "ExitProcess");
@(1);

gadgets available

LoadRegG
MovRegG
JumpG
Compiling the ROP program

our high-level language

`LoadLibrary("kernel32");
GetProcAddress(@, "WinExec");
@("calc", SW_SHOWNORMAL);
LoadLibrary("kernel32");
GetProcAddress(@, "ExitProcess");
@();

generate possible gadget arrangements

Gadget 1  Gadget 2  Gadget 3  Gadget 4  Gadget 5  Gadget 6  ...

gadgets available

LoadRegG  MovRegG  JumpG
Compiling the ROP program

our high-level language

\texttt{LoadLibrary(“kernel32”);}
\texttt{GetProcAddress(@, “WinExec”;)}
\texttt{@(“calc”, SW_SHOWNORMAL);}
\texttt{LoadLibrary(“kernel32”);}
\texttt{GetProcAddress(@, “ExitProcess”;)}
\texttt{@(1);}

gadgets available

fullfill with available gadgets

Reimplementation of Q gadget compiler algorithms [Schwartz et al., USENIX 2011] extended for multiple program statements and function parameters
Compiling the ROP program

Reimplementation of Q gadget compiler algorithms [Schwartz et al., USENIX 2011] extended for multiple program statements and function parameters.

our high-level language

\begin{verbatim}
LoadLibrary("kernel32");
GetProcAddress(@, "WinExec"); @("calc", SW_SHOWNORMAL);
LoadLibrary("kernel32");
GetProcAddress(@, "ExitProcess"); @(1);
\end{verbatim}

gadgets available

fullfill with available gadgets

generate possible gadget arrangements

Gadget 1 Gadget 2 Gadget 3 Gadget 4 Gadget 5 Gadget 6 ...
Compiling the ROP program

Reimplementation of Q gadget compiler algorithms [Schwartz et al., USENIX 2011] extended for multiple program statements and function parameters
JIT-ROP is only our initial prototype of **just-in-time code reuse**.

**Potential Improvements:**

<table>
<thead>
<tr>
<th>Map Memory</th>
<th>Improve ability to discern code from embedded data.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find API Calls</td>
<td>Explore direct use of system calls.</td>
</tr>
<tr>
<td>Find Gadgets</td>
<td>Lower conservativeness at expense of complexity.</td>
</tr>
<tr>
<td>Compile</td>
<td>Define more composite gadgets implementing an operation.</td>
</tr>
<tr>
<td>Run Time</td>
<td>Optimize code throughout.</td>
</tr>
</tbody>
</table>

Bigger changes: apply JIT code reuse to jump-oriented programming, return-less ROP, or ret-to-libc styles of code reuse.
Page Mapping Considerations

All other steps depend on the ability to map code pages well.

Are there enough function pointers on the heap?
Page Mapping Considerations

All other steps depend on the ability to map code pages well.

Are there enough function pointers on the heap?

Assume only **one code pointer** initially accessible.
(e.g. from a virtual table entry, callback, or event handler)
Page Mapping Considerations

All other steps depend on the ability to map code pages well.

- Are there enough function pointers on the heap?
- Assume only **one code pointer** initially accessible.
  (e.g. from a virtual table entry, callback, or event handler)
- Are code pages interconnected enough?
Page Mapping Considerations

All other steps depend on the ability to map code pages well.

- Are there enough function pointers on the heap?
- Are code pages interconnected enough?

Assume only one code pointer initially accessible.
(e.g. from a virtual table entry, callback, or event handler)

Tested on 7 Applications:
Experiment Design

For each application:

1. Open Application with Blank Document
2. Save Snapshots of Program Memory
3. Use only one initial code pointer to kick-off memory mapping, repeat for all possible initializations
4. Map Memory
5. Find API Calls
6. Find Gadgets
7. Compile

Build Native x86 Version of JIT-ROP
On average, 300 pages of code harvested.

Map Memory: Pages harvested from a single initial code pointer

Find API Calls

Find Gadgets

Run Time

Pages harvested:
- Median: 300
- Lower quartile: 100
- Upper quartile: 500

Thursday, August 1, 13
Experimental Results

Map Memory

Using the LoadLibrary() and GetProcAddress() APIs, the generated ROP payload can lookup any other APIs needed.

Find API Calls

Find 9 to 12 on average, but only one needed.

Find Gadgets

Run Time

![Graph showing the comparison between GetProcAddress() and LoadLibrary() with ASCII and UNICODE]

- GetProcAddress(): median 15, upper quartile 10
- LoadLibrary(): median 5

similar results for all applications
Experimental Results

Map Memory

Find API Calls

Find Gadgets

Run Time

We only consider ‘xchg eax, esp’ for a stack pivot, this could be improved.

Usually find one or more of each gadget type.

Also tested against ‘gadget elimination’, e.g. ORP [Pappas et al., IEEE S&P 2012], which had little benefit. Some gadgets vanished, while new gadgets appeared.
Experimental Results

Map Memory

Find API Calls

Find Gadgets

Run Time

Varies, but viable for real-world exploitation.

End-to-end live exploitation experiments with different environments and vulnerabilities.

CVE 2012 1876

string size overwrite

string size overwrite

format string disclosure

Run Time

seconds to exploit

Windows 7

Windows 8

V8

E10
Live Demo

CVE-2013-2551 on Windows 8 Internet Explorer 10

Credits

Vulnerability Discovery: Nicolas Joly
Metasploit Module for Win7/IE8: Juan Vazquez
Conclusion

Fine-grained ASLR

- not sufficient against adversary with ability to bypass *standard* ASLR via memory disclosure
Conclusion

Fine-grained ASLR

• not sufficient against adversary with ability to bypass *standard* ASLR via memory disclosure

Quick Fix?

• re-randomize periodically [Giuffrida et al., USENIX 2012]
• performance trade-off is impractical
Conclusion

- **Fine-grained ASLR**
  - not sufficient against adversary with ability to bypass *standard* ASLR via memory disclosure

- **Quick Fix?**
  - re-randomize periodically [Giuffrida et al., USENIX 2012]
  - performance trade-off is impractical

- **Towards More Comprehensive Mitigations**
  - control-flow integrity
    - [Abadi et al., CCS 2005]


## Conclusion

**Fine-grained ASLR**
- not sufficient against adversary with ability to bypass *standard* ASLR via memory disclosure

**Quick Fix?**
- re-randomize periodically [Giuffrida et al., USENIX 2012]
- performance trade-off is impractical

**Towards More Comprehensive Mitigations**
- control-flow integrity
  - [Abadi et al., CCS 2005]

**Need for Practical Solutions**
- work towards efficient fine-grained CFI/DFI