Evaluation of the Executional Power in Windows using Return Oriented Programming

Daniel Uroz, Ricardo J. Rodríguez*

© All wrongs reversed – under CC-BY-NC-SA 4.0 license

Universidad
Zaragoza

Dept. of Computer Science and Systems Engineering
University of Zaragoza, Spain

May 27, 2021

15th IEEE Workshop on Offensive Technologies
(online)

*Corresponding author: rjrodriguez@unizar.es
Outline

1. Introduction
2. Definition of the Virtual Language: ROPLANG
3. Evaluation
4. Related Work
5. Conclusions and Future Work
Outline

1 Introduction

2 Definition of the Virtual Language: ROPL\textsubscript{ANG}

3 Evaluation

4 Related Work

5 Conclusions and Future Work
Introduction

Return-Oriented-Programming (ROP) attacks

- A type of code-reuse techniques, introduced in 2007 by Shacham
- Hijacking of the control flow of a victim program without injected code

**ROP gadgets**: (relatively short) code snippets already present in the victim’s memory address space and ending in an assembly instruction that changes the control flow

**ROP chain**: a chain of ROP gadgets

```
b8 89 41 08 c3
mov eax, 0xc3084189
89 41 08
mov [ecx+8], eax
c3 ret
```

```
... esp → 0x7c37638d
pop ecx; ret 0xF13C1A02
0x7c341591 → pop edx; ret
0xBAADF00D 0x7c367042 → xor eax, eax; ret
0x7c34779f → add eax, ecx; ret
0x7c347f97 → mov ebx, eax; ret
...
```

Result: ecx = 0xF13C1A02, edx = 0xBAADF00D, eax = ebx = 0xF13C1A02
Introduction

Return-Oriented-Programming (ROP) attacks

- A type of code-reuse techniques, introduced in 2007 by Shacham
- Hijacking of the control flow of a victim program without injected code
- Known to be Turing-complete (i.e., performing any arbitrary computation)

Terminology

- ROP gadgets: (relatively short) code snippets already present in the victim’s memory address space and ending in an assembly instruction that changes the control flow
- ROP chain: a chain of ROP gadgets
Introduction
Return-Oriented-Programming (ROP) attacks

- A type of code-reuse techniques, introduced in 2007 by Shacham
- Hijacking of the control flow of a victim program without injected code
- Known to be Turing-complete (i.e., performing any arbitrary computation)

Terminology

- **ROP gadgets**: (relatively short) code snippets already present in the victim’s memory address space and ending in an assembly instruction that changes the control flow
- **ROP chain**: a chain of ROP gadgets

```
b8 89 41 08 c3        mov eax, 0xc3084189
                      
89 41 08              mov [ecx+8], eax
89 c3                 ret
```

Result:

- ecx = 0xF13C1A02
- edx = 0xBAADF00D
- eax = ebx = 0xF13C1A02

Executional Power in Windows using ROP [CC BY-NC-SA 4.0 © D. Uroz, R. J. Rodríguez]
Return-Oriented-Programming (ROP) attacks

- A **type of code-reuse techniques**, introduced in 2007 by Shacham
- Hijacking of the control flow of a victim program without injected code
- Known to be **Turing-complete** (i.e., performing any arbitrary computation)

**Terminology**

- **ROP gadgets**: (relatively short) code snippets already present in the victim’s memory address space and ending in an assembly instruction that changes the control flow
- **ROP chain**: a chain of ROP gadgets

```
b8 89 41 08 c3    mov eax, 0xc3084189
89 41 08          mov [ecx+8], eax
  c3              ret
```

Result:

```
 ecx=0xF13C1A02,
 edx=0xBAADF00D,
 eax=ebx=0xF13C1A02
```
Introduction

*How much is the executional power of an adversary?*
Introduction

How much is the executional power of an adversary?

Research Questions

**Q1** How often do ROP gadgets emerge for any arbitrary operation in real world programs?

**Q2** Is it possible to chain gadgets for any desired computation? Can adversaries build any kind of algorithm using a ROP chain?
Introduction

How much is the executional power of an adversary?

Research Questions

Q1 How often do ROP gadgets emerge for any arbitrary operation in real world programs?

Q2 Is it possible to chain gadgets for any desired computation? Can adversaries build any kind of algorithm using a ROP chain?

Adversary model

- ASLR is not deployed on the target system, or a break is available for ASLR
- CFI protection mechanisms are disabled in the victim program, or a break is available for CFI protection mechanisms deployed
- The content of the memory address space of the victim program is known
Contributions

- **Definition of a Turing-complete virtual language**, named ROPL\textsubscript{ANG}

- **Quantification of the executional power of an adversary in Windows 7 and Windows 10** (in their x86 and x86-64 versions)

- **The software tool ROP3**:  
  - Takes as input a set of program files and a ROP chain described with ROPL\textsubscript{ANG}  
  - Returns the ROP gadgets that make up such ROP chain
Outline

1. Introduction

2. Definition of the Virtual Language: ROPLANG

3. Evaluation

4. Related Work

5. Conclusions and Future Work
Definition of the Virtual Language: ROPLANG

Virtual operations

- Simulated using sequences of instructions of the vulnerable program conformed by ROP gadgets
- Similar notation to Intel’s assembly notation
  - Our language adheres to the Intel x86 syntax
Definition of the Virtual Language: ROPLANG

Virtual operations

- Simulated using sequences of instructions of the vulnerable program conformed by ROP gadgets
- Similar notation to Intel’s assembly notation
  - Our language adheres to the Intel x86 syntax

Categories of operations

- Arithmetic: addition (add), subtraction (sub), and negation (neg)
- Assignment: assign values to variables (logical registers of the CPU)
- Dereference: visit a memory location for reading or writing (ld, st)
- Logical: xor, and, or, and not operations
  - By De Morgan’s Laws, they can be simplified to an operation {and, or} plus an operation of the set {xor, not, neg}
- Branching: conditional and unconditional
  - Conditional branching operations require some tricky steps up front
**Definition of the Virtual Language: ROPL**

### Arithmetic operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>ROP gadgets/Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(dst, src)</td>
<td>add dst, src</td>
</tr>
<tr>
<td></td>
<td>clc</td>
</tr>
<tr>
<td></td>
<td>adc dst, src</td>
</tr>
<tr>
<td></td>
<td>inc dst</td>
</tr>
<tr>
<td>sub(dst, src)</td>
<td>sub dst, src</td>
</tr>
<tr>
<td></td>
<td>clc</td>
</tr>
<tr>
<td></td>
<td>sbb dst, src</td>
</tr>
<tr>
<td></td>
<td>dec dst</td>
</tr>
<tr>
<td>neg(dst)</td>
<td>xor REG1, REG1</td>
</tr>
<tr>
<td></td>
<td>sub REG1, dst</td>
</tr>
<tr>
<td></td>
<td>mov(dst, REG1)</td>
</tr>
<tr>
<td></td>
<td>neg dst</td>
</tr>
</tbody>
</table>

The *ret* instruction (at the end of each ROP gadget) was deliberately omitted.
Definition of the Virtual Language: ROPLANG

### Arithmetic operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>ROP gadgets/Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(dst, src)</td>
<td>add dst, src</td>
</tr>
<tr>
<td></td>
<td>clc</td>
</tr>
<tr>
<td></td>
<td>adc dst, src</td>
</tr>
<tr>
<td></td>
<td>inc dst</td>
</tr>
<tr>
<td>sub(dst, src)</td>
<td>sub dst, src</td>
</tr>
<tr>
<td></td>
<td>clc</td>
</tr>
<tr>
<td></td>
<td>sbb dst, src</td>
</tr>
<tr>
<td></td>
<td>dec dst</td>
</tr>
<tr>
<td>neg(dst)</td>
<td>xor REG1, REG1</td>
</tr>
<tr>
<td></td>
<td>sub REG1, dst</td>
</tr>
<tr>
<td></td>
<td>mov(dst, REG1)</td>
</tr>
<tr>
<td></td>
<td>neg dst</td>
</tr>
</tbody>
</table>

### Assignment operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>ROP gadgets</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov(dst, src)</td>
<td>mov dst, src</td>
</tr>
<tr>
<td></td>
<td>xchg dst, src</td>
</tr>
<tr>
<td></td>
<td>xor dst, dst</td>
</tr>
<tr>
<td></td>
<td>add dst, src</td>
</tr>
<tr>
<td></td>
<td>xor dst, dst</td>
</tr>
<tr>
<td></td>
<td>not dst</td>
</tr>
<tr>
<td></td>
<td>and dst, src</td>
</tr>
<tr>
<td></td>
<td>clc</td>
</tr>
<tr>
<td></td>
<td>cmovnc dst, src</td>
</tr>
<tr>
<td></td>
<td>stc</td>
</tr>
<tr>
<td></td>
<td>cmovc dst, src</td>
</tr>
<tr>
<td></td>
<td>push src</td>
</tr>
<tr>
<td></td>
<td>pop dst</td>
</tr>
<tr>
<td></td>
<td>pop dst; value is set in the stack</td>
</tr>
<tr>
<td></td>
<td>popad; value is set in the stack appropriately</td>
</tr>
</tbody>
</table>

*The ret instruction (at the end of each ROP gadget) was deliberately omitted.*

Executional Power in Windows using ROP [CC BY-NC-SA 4.0 © D. Uroz, R. J. Rodríguez]
Definition of the Virtual Language: ROPLAN

Arithmetic operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>ROP gadgets/Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(dst, src)</td>
<td>add dst, src</td>
</tr>
<tr>
<td></td>
<td>clc</td>
</tr>
<tr>
<td></td>
<td>adc dst, src</td>
</tr>
<tr>
<td></td>
<td>inc dst</td>
</tr>
<tr>
<td>sub(dst, src)</td>
<td>sub dst, src</td>
</tr>
<tr>
<td></td>
<td>clc</td>
</tr>
<tr>
<td></td>
<td>sbb dst, src</td>
</tr>
<tr>
<td></td>
<td>dec dst</td>
</tr>
<tr>
<td>neg(dst)</td>
<td>xor REG1, REG1</td>
</tr>
<tr>
<td></td>
<td>sub REG1, dst</td>
</tr>
<tr>
<td></td>
<td>mov(dst, REG1)</td>
</tr>
<tr>
<td></td>
<td>neg dst</td>
</tr>
</tbody>
</table>

Assignment operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>ROP gadgets</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov(dst, src)</td>
<td>mov dst, src</td>
</tr>
<tr>
<td></td>
<td>xchg dst, src</td>
</tr>
<tr>
<td></td>
<td>xor dst, dst</td>
</tr>
<tr>
<td></td>
<td>add dst, src</td>
</tr>
<tr>
<td></td>
<td>xor dst, dst</td>
</tr>
<tr>
<td></td>
<td>not dst</td>
</tr>
<tr>
<td></td>
<td>and dst, src</td>
</tr>
<tr>
<td></td>
<td>clc</td>
</tr>
<tr>
<td></td>
<td>cmovnc dst, src</td>
</tr>
<tr>
<td></td>
<td>stc</td>
</tr>
<tr>
<td></td>
<td>cmovc dst, src</td>
</tr>
<tr>
<td></td>
<td>push src</td>
</tr>
<tr>
<td></td>
<td>pop dst</td>
</tr>
<tr>
<td></td>
<td>lc(dst, value) pop dst; value is set in the stack</td>
</tr>
<tr>
<td></td>
<td>popad; value is set in the stack appropriately</td>
</tr>
</tbody>
</table>

Dereference operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>ROP gadgets</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld(dst, src)</td>
<td>mov dst, [src]</td>
</tr>
<tr>
<td>st(dst, src)</td>
<td>mov [dst], src</td>
</tr>
</tbody>
</table>

The ret instruction (at the end of each ROP gadget) was deliberately omitted
### Arithmetic operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>ROP gadgets/Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(dst, src)</td>
<td>add dst, src</td>
</tr>
<tr>
<td></td>
<td>clc</td>
</tr>
<tr>
<td></td>
<td>adc dst, src</td>
</tr>
<tr>
<td></td>
<td>inc dst</td>
</tr>
<tr>
<td>sub(dst, src)</td>
<td>sub dst, src</td>
</tr>
<tr>
<td></td>
<td>clc</td>
</tr>
<tr>
<td></td>
<td>sbb dst, src</td>
</tr>
<tr>
<td></td>
<td>dec dst</td>
</tr>
<tr>
<td>neg(dst)</td>
<td>xor REG1, REG1</td>
</tr>
<tr>
<td></td>
<td>sub REG1, dst</td>
</tr>
<tr>
<td></td>
<td>mov(dst, REG1)</td>
</tr>
<tr>
<td></td>
<td>neg dst</td>
</tr>
</tbody>
</table>

### Logical operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>ROP gadgets</th>
</tr>
</thead>
<tbody>
<tr>
<td>xor(dst, src)</td>
<td>xor dst, src</td>
</tr>
<tr>
<td>and(dst, src)</td>
<td>and dst, src</td>
</tr>
<tr>
<td>or(dst, src)</td>
<td>or dst, src</td>
</tr>
<tr>
<td>not(dst)</td>
<td>not dst</td>
</tr>
</tbody>
</table>

### Assignment operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>ROP gadgets</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov(dst, src)</td>
<td>mov dst, src</td>
</tr>
<tr>
<td></td>
<td>xchg dst, src</td>
</tr>
<tr>
<td></td>
<td>xor dst, dst</td>
</tr>
<tr>
<td></td>
<td>add dst, src</td>
</tr>
<tr>
<td></td>
<td>xor dst, dst</td>
</tr>
<tr>
<td></td>
<td>not dst</td>
</tr>
<tr>
<td></td>
<td>and dst, src</td>
</tr>
<tr>
<td></td>
<td>clc</td>
</tr>
<tr>
<td></td>
<td>cmovnc dst, src</td>
</tr>
<tr>
<td></td>
<td>stc</td>
</tr>
<tr>
<td></td>
<td>cmovc dst, src</td>
</tr>
<tr>
<td></td>
<td>push src</td>
</tr>
<tr>
<td></td>
<td>pop dst</td>
</tr>
</tbody>
</table>

### Dereference operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>ROP gadgets</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld(dst, src)</td>
<td>mov dst, [src]</td>
</tr>
<tr>
<td>st(dst, src)</td>
<td>mov [dst], src</td>
</tr>
</tbody>
</table>

*The ret instruction (at the end of each ROP gadget) was deliberately omitted*
Definition of the Virtual Language: ROPL\textsubscript{ANG}

Comparison operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>eqc(dst, src)</td>
<td>sub(dst, src)</td>
</tr>
<tr>
<td>neg(dst)</td>
<td>sub(dst, src)</td>
</tr>
</tbody>
</table>

Conditional branching

<table>
<thead>
<tr>
<th>Operation</th>
<th>ROP gadgets/Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>lc(REG1, 0)</td>
<td>gcf(dst\textsubscript{CF}, cop(dst, src))</td>
</tr>
<tr>
<td>adc dst\textsubscript{CF}, REG1</td>
<td></td>
</tr>
<tr>
<td>lc(REG1, 0)</td>
<td></td>
</tr>
<tr>
<td>cop(dst, src)</td>
<td></td>
</tr>
<tr>
<td>sbb dst\textsubscript{CF}, REG1</td>
<td></td>
</tr>
<tr>
<td>neg(dst\textsubscript{CF})</td>
<td></td>
</tr>
<tr>
<td>lc(dst\textsubscript{CF}, 0)</td>
<td></td>
</tr>
<tr>
<td>cop(dst, src)</td>
<td></td>
</tr>
<tr>
<td>rcl dst\textsubscript{CF}, 1</td>
<td></td>
</tr>
<tr>
<td>lsd(dst\textsubscript{CF}, δ)</td>
<td>lc(REG1, δ)</td>
</tr>
<tr>
<td>neg(dst\textsubscript{CF})</td>
<td></td>
</tr>
<tr>
<td>and(dst\textsubscript{CF}, REG1)</td>
<td></td>
</tr>
<tr>
<td>spa(src)</td>
<td>add(REG_SP, src)</td>
</tr>
<tr>
<td>sps(src)</td>
<td>sub(REG_SP, src)</td>
</tr>
</tbody>
</table>

Unconditional branching

<table>
<thead>
<tr>
<th>Operation</th>
<th>ROP gadgets/Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>jmp(dst, δ)</td>
<td>lc(dst, δ)</td>
</tr>
<tr>
<td>spa(dst)</td>
<td></td>
</tr>
</tbody>
</table>

The \textit{ret} instruction (at the end of each ROP gadget) was deliberately omitted
Definition of the Virtual Language: ROPLang

Some remarks

- Non-exhaustive list of ROP gadgets
- Some operations are virtual operations, while others are ROP gadgets
- **Assumption:** no harmful side effects occur between sequences of virtual operations

**ROPLang is Turing-complete**

- Simulation of a classic Turing machine with ROPLang in the paper
Definition of the Virtual Language: ROPL\textsubscript{A}NG

The ROP3 tool

ROP3

- Developed in Python, relying on Capstone to disassemble input files
- Supports the virtual operations that make up ROPL\textsubscript{A}NG
- Defining operations using YAML syntax
  - Custom operations are possible (as a single or as multiple YAML files)
  - Logical CPU registers and register masks can be specified
  - Arbitrary values can also be set
- Similar approach to the Galileo algorithm to search for ROP gadgets
Definition of the Virtual Language: ROPL\textsc{ang}

The ROP3 tool – examples of YAML file

```yaml
1 # Add values
2 add:
3   # add dst, src
4   - mnemonic: add
5     op1: dst
6     op2: src
7
8 # clc
9 # adc dst, src
10 -
11   - mnemonic: clc
12   - mnemonic: adc
13     op1: dst
14     op2: src
```

```yaml
1 # NOT value
2 not:
3   # not dst
4   -
5     - mnemonic: not
6       op1: dst
7
8 # xor dst, src (src = 0xFFFFFFFF)
9 -
10   - mnemonic: xor
11     op1: dst
12     op2:
13       reg: src
14       value: 0xFFFFFFFF
```
Definition of the Virtual Language: ROPLANG

The R0P3 tool – construction of ROP chains

- **Specified by virtual operations of** ROPLANG
- **Search algorithm:**
  1. Finds all gadgets that comply with each ROPLANG operation in the chain
  2. Builds a tree structure, considering the order of operations defined in the chain
  3. Resolves data dependencies between operations by traversing the tree recursively in depth-first order with backtracking

- **Handling of side effects in the chain:** TODO
Definition of the Virtual Language: ROPL\textsubscript{ANG}

The ROP3 tool – construction of ROP chains

- **Specified by virtual operations of ROPL\textsubscript{ANG}**
- **Search algorithm:**
  1. Finds all gadgets that comply with each ROPL\textsubscript{ANG} operation in the chain
  2. Builds a tree structure, considering the order of operations defined in the chain
  3. Resolves data dependencies between operations by traversing the tree recursively in depth-first order with backtracking

- **Handling of side effects in the chain:** TODO

<table>
<thead>
<tr>
<th>ROP chain (input)</th>
<th>ROP gadgets found</th>
<th>Tree structure and backtracking</th>
</tr>
</thead>
<tbody>
<tr>
<td>lc(reg1)</td>
<td>pop edx ; ret</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pop edi ; ret</td>
<td></td>
</tr>
<tr>
<td>neg(reg2)</td>
<td>neg ebx ; ret</td>
<td></td>
</tr>
<tr>
<td></td>
<td>neg ecx ; ret</td>
<td></td>
</tr>
<tr>
<td>and(reg2, reg1)</td>
<td>and ecx, eax ; ret</td>
<td></td>
</tr>
<tr>
<td></td>
<td>and ecx, edx ; ret</td>
<td></td>
</tr>
</tbody>
</table>

Executional Power in Windows using ROP [CC BY-NC-SA 4.0 © D. Uroz, R. J. Rodríguez]
Definition of the Virtual Language: ROPLang

The ROP3 tool

- **Released under the GNU/GPLv3 license**
- Accepts **many parameters**:
  - Maximum byte size of ROP gadgets
  - Gadget final instructions (*ret, jmp, retf*)
  - ...

- Is also a Python3 library

https://github.com/reverseame/rop3
Outline

1. Introduction

2. Definition of the Virtual Language: ROPLANG

3. Evaluation

4. Related Work

5. Conclusions and Future Work
Evaluation

Test-bed

- Subset of DLLs contained in the KnownDlls system object
  - Common DLLs across all the versions of Windows considered for the experimentation
- Windows on top of Oracle VirtualBox hypervisor, 32-bit and 64-bit versions
  - Windows 10 Education 10.0.14393 Build 14393 (32-bit) and
    Windows 10 Pro 1703 Build 15063.726 (64-bit)
  - Windows 7 Professional 6.1.7601 Service Pack 1 Build 7601

Regarding the plots...

- Heatmap of the occurrence (in %) for each operation within each DLL
- Annotations show the number of results
  - Most significant digit and order of magnitude when the number of results is $\geq 10^4$
- DLLs sorted by byte size
Evaluation

**Configuration of ROP3**

- 10-byte-length ROP gadgets
- Only `ret` as final instruction
Evaluation

**Configuration of ROP3**

- 10-byte-length ROP gadgets
- Only `ret` as final instruction
- ROP gadgets made up of the same ins. sequence: counted only once
- **Only the current definitions of ROPLang operations**
- ROP gadgets made up of several instructions are treated as single gadgets
- **Additional operations considered**
  - `spa-4`, `spa-8`, `spa-16`, and `spa-32`
  - `gcf` divided into `gcf-eqc` and `gcf-ltc`
Evaluation

Prevalence of ROP Gadgets

Windows 7 SP1 32-bit

Windows 10 32-bit

Windows 7 SP1 64-bit

Windows 10 64-bit

Executional Power in Windows using ROP [CC BY-NC-SA 4.0 © D. Uroz, R. J. Rodríguez] WOOT’21 19/28
Evaluation
Prevalence of ROP Gadgets – Discussion

- **Branching virtual operations are the least frequent**, in both architectures
  - No results for unconditional branching in 64-bit systems

- **Different results in the other virtual operations**
  - The larger the DLLs, the greater the number of results (as expected)

**NOTE**: in 32-bit assembly, the instructions can have references to memory addresses that are randomized by ASLR. We have considered each DLL with its base address. Hence, these results: Are highly dependent on the base addresses of the DLLs Can change when the base addresses are different

How does ASLR affect the prevalence of ROP gadgets on 32-bit Windows systems?
Evaluation
Prevalence of ROP Gadgets – Discussion

- **Branching virtual operations are the least frequent**, in both architectures
  - No results for unconditional branching in 64-bit systems

- **Different results in the other virtual operations**
  - The larger the DLLs, the greater the number of results (as expected)

- **The number of virtual operations in Windows 10 is always greater than in Windows 7, and in 64-bit than in 32-bit**
  - May be motivated due to differences in DLL sizes

- **NOTE**: in 32-bit assembly, the instructions can have references to memory addresses that are randomized by ASLR. We have considered each DLL with its base address. Hence, these results:
  - Are highly dependent on the base addresses of the DLLs
  - Can change when the base addresses are different
Evaluation
Prevalence of ROP Gadgets – Discussion

- **Branching virtual operations are the least frequent**, in both architectures
  - No results for unconditional branching in 64-bit systems
- **Different results in the other virtual operations**
  - The larger the DLLs, the greater the number of results (as expected)
- **The number of virtual operations in Windows 10 is always greater than in Windows 7, and in 64-bit than in 32-bit**
  - May be motivated due to differences in DLL sizes
- **NOTE**: in 32-bit assembly, the instructions can have references to memory addresses that are randomized by ASLR. We have considered each DLL with its base address. Hence, these results:
  - Are highly dependent on the base addresses of the DLLs
  - Can change when the base addresses are different

How does ASLR affect the prevalence of ROP gadgets on 32-bit Windows systems?
Evaluation
Simulating a Turing machine – intermediate mov

- Very limited results for conditional and unconditional operations
- Mandatory operations to simulate a classic Turing machine
Evaluation
Simulating a Turing machine – intermediate \texttt{mov}

- Very limited results for conditional and unconditional operations
  - **Mandatory operations** to simulate a classic Turing machine

- **IDEA**: Relax data dependency constraints on certain operations by adding intermediate assignment operations (like \texttt{mov(reg1, dst)})
  - High probability of finding the \texttt{mov(reg1, dst)} operation
  - By contrast, the length of the ROP chain will increase and more side effects are likely to occur

- **Example of extension**: \texttt{eqc(dst, src)}
  
  $\text{sub(dst, src)} \Rightarrow \text{sub(dst, src)}$
  $\text{neg(dst)} \Rightarrow \text{mov(reg1, dst)}$
  $\text{neg(reg1)}$
Evaluation

Simulating a Turing machine – intermediate \texttt{mov}

- **Very limited results for conditional and unconditional operations**

  - **Mandatory operations** to simulate a classic Turing machine

- **IDEA**: Relax data dependency constraints on certain operations by adding intermediate assignment operations (like \texttt{mov(reg1, dst)})

  - High probability of finding the \texttt{mov(reg1, dst)} operation
  - By contrast, the length of the ROP chain will increase and more side effects are likely to occur

- **Example of extension**: \texttt{eqc(dst, src)}

  \[
  \begin{align*}
  \text{sub}(\texttt{dst}, \texttt{src}) & \Rightarrow \text{sub}(\texttt{dst}, \texttt{src}) \\
  \text{neg}(\texttt{dst}) & \Rightarrow \text{mov}(\texttt{reg1}, \texttt{dst}) \\
  \text{neg}(\texttt{reg1}) &
  \end{align*}
  \]

- \texttt{neg, eqc, gcf, lsd, and jmp} operations

  - **Extended with the use of intermediate \texttt{mov} between their operations**
Evaluation

Simulating a Turing machine – intermediate mov

Windows 7 SP1 32-bit (with an intermediate mov)

Windows 7 SP1 64-bit (with an intermediate mov)

Windows 10 32-bit (with an intermediate mov)

Windows 10 64-bit (with an intermediate mov)

Zaragoza

Executional Power in Windows using ROP [CC BY-NC-SA 4.0 © D. Uroz, R. J. Rodríguez]  WOOT’21 22 / 28
Evaluation
Simulating a Turing machine – Discussion

- More results on 32-bit systems, still discrete results on 64-bit systems
  - No results yet for unconditional operation on Windows 7 SP1 64 bits
- A sophisticated link of other operations increases the probability of simulating any operation when it is not found directly
  - Simple extension of virtual operations supported by ROP3
Outline

1. Introduction

2. Definition of the Virtual Language: ROPLANG

3. Evaluation

4. Related Work

5. Conclusions and Future Work
Related Work

- **Tools focused on detection and mitigation ROP attacks**
  - DROP, ROPDefender, ROPGuard, kBouncer (to name a few)

- **Tools more focused on offensive technology**
  - ROPInjector, Frankenstein, ROPOB, RopSteg, SpecROP

- **Generation and analysis of ROP chains**
  - deROP, SROP, ROPEMU, AMOCO
  - ropper, ROPgadget, ropium

Our approach

- Simpler solution
- Easy extension to search for semantically equivalent operations
- Automatic generation of ROP chains backing in ROPLANG operations
Outline

1 Introduction

2 Definition of the Virtual Language: ROPLANG

3 Evaluation

4 Related Work

5 Conclusions and Future Work
Conclusions and Future Work

- Defined a virtual language, dubbed ROPLang, whose operations are mapped to ROP gadgets
- Developed ROP3, a tool that allows a user to find ROP gadgets and build a ROP chain specified by ROPLang
Conclusions and Future Work

- Defined a virtual language, dubbed ROPLang, whose operations are mapped to ROP gadgets
- Developed ROP3, a tool that allows a user to find ROP gadgets and build a ROP chain specified by ROPLang

- Any virtual operation is found, the branching operation ones being the least frequent
  - Careful linking of virtual operations can be performed to find operations that are not found directly
- The size of the program file clearly impacts the prevalence of ROP gadgets

Future work

- Eliminate side-effects that can occur with some ROP gadgets
- Evaluate the executional powers in other operating systems
Conclusions and Future Work

- Defined a virtual language, dubbed ROPLang, whose operations are mapped to ROP gadgets.

- Developed ROP3, a tool that allows a user to find ROP gadgets and build a ROP chain specified by ROPLang.

- Any virtual operation is found, the branching operation ones being the least frequent.
  - Careful linking of virtual operations can be performed to find operations that are not found directly.

- The size of the program file clearly impacts the prevalence of ROP gadgets.

Future work

- Eliminate side-effects that can occur with some ROP gadgets.

- Evaluate the executional powers in other operating systems.
Evaluation of the Executional Power in Windows using Return Oriented Programming

Daniel Uroz, Ricardo J. Rodríguez*

© All wrongs reversed – under CC-BY-NC-SA 4.0 license

Universidad Zaragoza
Dept. of Computer Science and Systems Engineering
University of Zaragoza, Spain

May 27, 2021

15th IEEE Workshop on Offensive Technologies (online)

*Corresponding author: rjrodriguez@unizar.es