Evaluation of the Executional Power in Windows using Return Oriented Programming

Daniel Uroz, **Ricardo J. Rodríguez***

Universidad Zaragoza
1542

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

Outline

Universidad Zaragoza

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**

Universidad Zaragoza

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

Universidad Zaragoza

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
c3                  ret
```

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
c3                   ret
```

	Stack	Gadget
	...	
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

Universidad Zaragoza

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?

Universidad Zaragoza

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*

Universidad Zaragoza

Introduction

CONTRIBUTIONS

- **Definition of a Turing-complete virtual language**, named ROPLANG
- **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 ROPLANG
  - Returns the ROP gadgets that make up such ROP chain

Universidad Zaragoza

Outline

Universidad Zaragoza

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

Universidad Zaragoza

Definition of the Virtual Language: ROPLang

Arithmetic operations

Operation	ROP gadgets/Operations
add(dst, src)	add dst, src
	clc adc dst, src
	inc dst
sub(dst, src)	sub dst, src
	clc sbb dst, src
	dec dst
neg(dst)	xor REG1, REG1 sub REG1, dst mov(dst, REG1)
	neg dst

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

Definition of the Virtual Language: ROPLang

Arithmetic operations

Operation	ROP gadgets/Operations
add(dst, src)	add dst, src
	clc adc dst, src
	inc dst
sub(dst, src)	sub dst, src
	clc sbb dst, src
	dec dst
neg(dst)	xor REG1, REG1 sub REG1, dst mov(dst, REG1)
	neg dst

Assignment operations

Operation	ROP gadgets
mov(dst, src)	mov dst, src
	xchg dst, src
	xor dst, dst add dst, src
	xor dst, dst not dst and dst, src
	clc cmovnc dst, src
	stc cmovc dst, src
	push src pop dst
lc(dst, *value*)	pop dst; *value is set in the stack*
	popad; *value is set in the stack appropriately*

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

Universidad Zaragoza

Definition of the Virtual Language: ROPLang

Arithmetic operations

Operation	ROP gadgets/Operations
add(dst, src)	add dst, src
	clc adc dst, src
	inc dst
sub(dst, src)	sub dst, src
	clc sbb dst, src
	dec dst
neg(dst)	xor REG1, REG1 sub REG1, dst mov(dst, REG1)
	neg dst

Assignment operations

Operation	ROP gadgets
mov(dst, src)	mov dst, src
	xchg dst, src
	xor dst, dst add dst, src
	xor dst, dst not dst and dst, src
	clc cmovnc dst, src
	stc cmovc dst, src
	push src pop dst
lc(dst, *value*)	pop dst; *value is set in the stack*
	popad; *value is set in the stack appropriately*

Dereference operations

Operation	ROP gadgets
ld(dst, src)	mov dst, [src]
st(dst, src)	mov [dst], src

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

Definition of the Virtual Language: ROPLang

Arithmetic operations

Operation	ROP gadgets/Operations
add(dst, src)	add dst, src
	clc adc dst, src
	inc dst
sub(dst, src)	sub dst, src
	clc sbb dst, src
	dec dst
neg(dst)	xor REG1, REG1 sub REG1, dst mov(dst, REG1)
	neg dst

Logical operations

Operation	ROP gadgets
xor(dst, src)	xor dst, src
and(dst, src)	and dst, src
or(dst, src)	or dst, src
not(dst)	not dst
	xor dst, 0xFFFFFFFF

Assignment operations

Operation	ROP gadgets
mov(dst, src)	mov dst, src
	xchg dst, src
	xor dst, dst add dst, src
	xor dst, dst not dst and dst, src
	clc cmovnc dst, src
	stc cmovc dst, src
	push src pop dst
lc(dst, *value*)	pop dst; *value is set in the stack*
	popad; *value is set in the stack appropriately*

Dereference operations

Operation	ROP gadgets
ld(dst, src)	mov dst, [src]
st(dst, src)	mov [dst], src

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

Universidad Zaragoza

Definition of the Virtual Language: ROPLang

Comparison operations

Operation	Operation
eqc(dst, src)	sub(dst, src) neg(dst)
ltc(dst, src)	sub(dst, src)

Conditional branching

Operation	ROP gadgets/Operations
$gcf(dst_{CF}, cop(dst, src))$	lc(REG1, 0) *Comparison operation* cop(dst, src) **adc** dst_{CF}, REG1
	lc(REG1, 0) *Comparison operation* cop(dst, src) **sbb** dst_{CF}, REG1 $neg(dst_{CF})$
	$lc(dst_{CF}, 0)$ *Comparison operation* cop(dst, src) **rcl** dst_{CF}, 1
$lsd(dst_{CF}, \delta)$	$lc(REG1, \delta)$ $neg(dst_{CF})$ $and(dst_{CF}, REG1)$
spa(src)	add(REG_SP, src)
sps(src)	sub(REG_SP, src)

Unconditional branching

Operation	ROP gadgets/Operations
$jmp(dst, \delta)$	$lc(dst, \delta)$ spa(dst)

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

Universidad Zaragoza

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**

Universidad Zaragoza

Definition of the Virtual Language: ROPLang

The ROP3 tool

ROP3

- **Developed in Python, relying on Capstone** to disassemble input files
- **Supports the virtual operations that make up** ROPLang
- **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: ROPLang

The ROP3 tool – examples of YAML file

```yaml
# Add values
add:
  # add dst, src
  -
    - mnemonic: add
      op1: dst
      op2: src

  # clc
  # adc dst, src
  -
    - mnemonic: clc
    - mnemonic: adc
      op1: dst
      op2: src
```

```yaml
# NOT value
not:
  # not dst
  -
    - mnemonic: not
      op1: dst

  # xor dst, src (src = 0xFFFFFFFF)
  -
    - mnemonic: xor
      op1: dst
      op2:
        reg: src
        value: 0xFFFFFFFF
```

Universidad Zaragoza

Definition of the Virtual Language: ROPLang

The ROP3 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**

Universidad Zaragoza

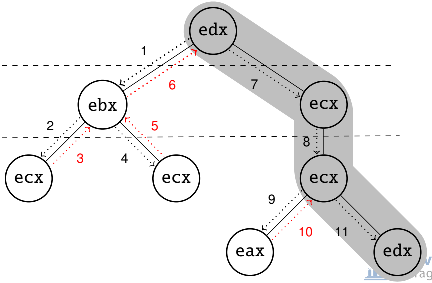
Definition of the Virtual Language: ROPLANG

The ROP3 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**

ROP chain (input)	*ROP gadgets found*
`lc(reg1)`	`pop edx ; ret` `pop edi ; ret`
`neg(reg2)`	`neg ebx ; ret` `neg ecx ; ret`
`and(reg2, reg1)`	`and ecx, eax ; ret` `and ecx, edx ; ret`

Tree structure and backtracking



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

Universidad Zaragoza

Outline

Universidad Zaragoza

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**

Universidad Zaragoza

Evaluation

Configuration of ROP3

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

Universidad Zaragoza

Evaluation

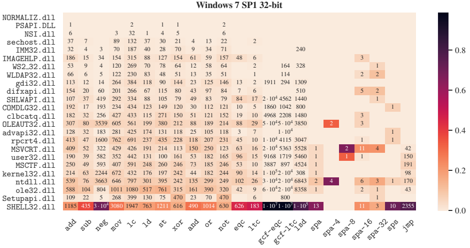
Configuration of ROP3

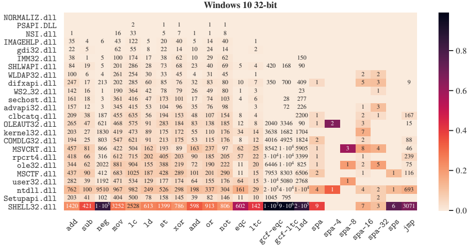
- **10-byte-length ROP gadgets**
- **Only `ret` as final instruction**
- **ROP gadgets made up of the same ins. sequence: counted only once**
- **Only the current definitions of ROPLang operations**
- ROP gadgets made up of several instructions are treated as single gadgets
- **Additional operations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

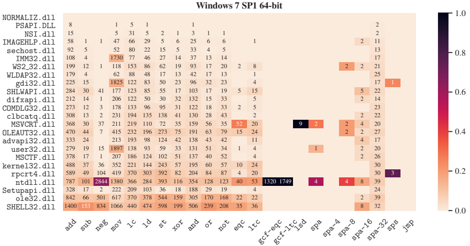
Universidad Zaragoza

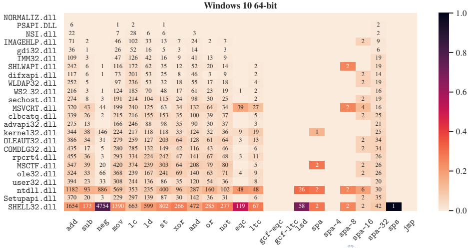
Evaluation

Prevalence of ROP Gadgets









Zaragoza

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)

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?

Universidad Zaragoza

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 `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 `mov(reg1, dst)`)
  - High probability of finding the `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*: `eqc(dst, src)`

```
sub(dst, src)   ⇒   sub(dst, src)
neg(dst)            mov(reg1, dst)
                    neg(reg1)
```

Evaluation

Simulating a Turing machine – intermediate `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 `mov(reg1, dst)`)
  - High probability of finding the `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*: `eqc(dst, src)`

```
sub(dst, src)   ⇒   sub(dst, src)
neg(dst)            mov(reg1, dst)
                    neg(reg1)
```

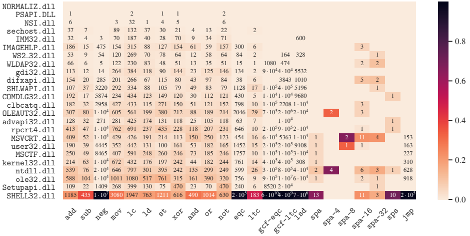
- `neg`, `eqc`, `gcf`, `lsd`, and `jmp` operations
  - **Extended with the use of intermediate `mov` between their operations**

Universidad Zaragoza

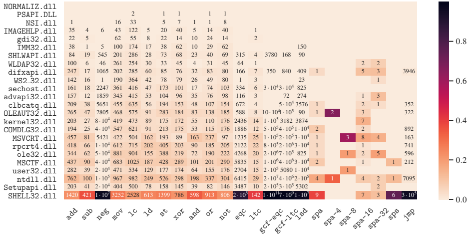
Evaluation

Simulating a Turing machine – intermediate `mov`

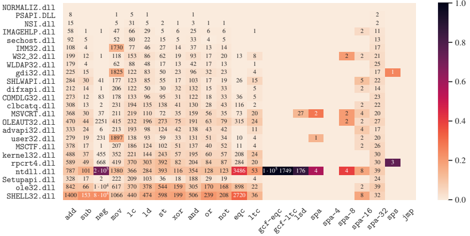
Windows 7 SP1 32-bit (with an intermediate mov)



Windows 10 32-bit (with an intermediate mov)



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



Windows 10 64-bit (with an intermediate mov)

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`

Universidad Zaragoza

Outline

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**

Universidad Zaragoza

Outline

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

Universidad Zaragoza

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

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**

Universidad Zaragoza

Evaluation of the Executional Power in Windows using Return Oriented Programming

Daniel Uroz, **Ricardo J. Rodríguez***

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