Let’s hack our first program !

Caroline LEMAN

Women in technology

Nov 26, 2019

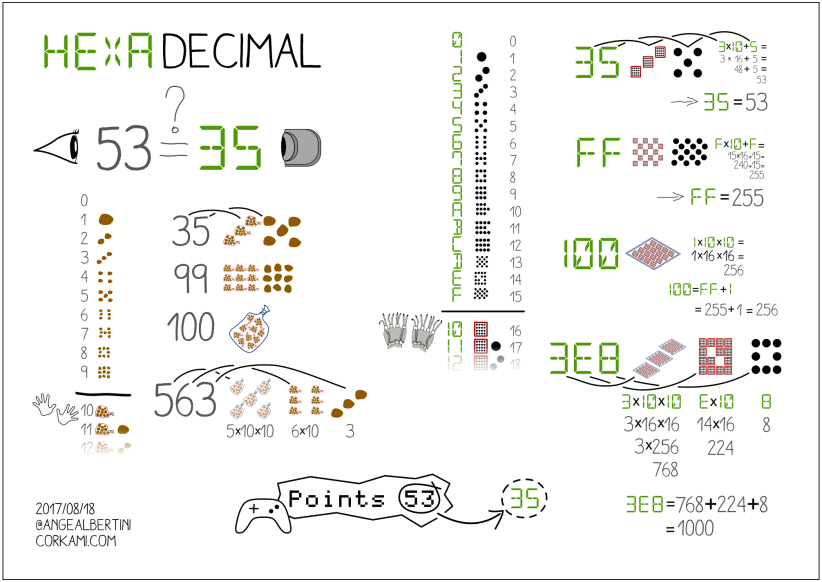
What are we going to reverse-engineer ?

- Reverse-engineering ?
  - The process of understanding how things work.
- Goal
  - Find the password to unlock the software.
  - What is a software ?
  - How does it run on a computer ?
  - Download the workshop materials here: https://bit.ly/2QWiF1E

What are we going to reverse-engineer ?

- Reverse-engineering ?
  - The process of understanding how things work.
- Goal
  - Find the password to unlock the software.
  - What is a software ?
  - How does it run on a computer ?
  - Download the workshop materials here: https://bit.ly/2QWiF1E

What is a software ?



Ascii characters

0x30	0	0x40	@	0x50	P	0x60,	`	0x70	p
0x31	1	0x41	A	0x51	Q	0x61	a	0x71	q
0x32	2	0x42	B	0x52	R	0x62	b	0x72	r
0x33	3	0x43	C	0x53	S	0x63	c	0x73	s
0x34	4	0x44	D	0x54	T	0x64	d	0x74	t
0x35	5	0x45	E	0x55	U	0x65	e	0x75	u
0x36	6	0x46	F	0x56	V	0x66	f	0x76	v
0x37	7	0x47	G	0x57	W	0x67	g	0x77	w
0x38	8	0x48	H	0x58	X	0x68	h	0x78	x
0x39	9	0x49	I	0x59	Y	0x69	i	0x79	y
0x3a	:	0x4a	J	0x5a	Z	0x6a	j	0x7a	z
0x3b	;	0x4b	K	0x5b,	[	0x6b	k	0x7b	{
0x3c	<	0x4c	L	0x5c,	\	0x6c	l	0x7c	|
0x3d	=	0x4d	M	0x5d,	]	0x6d	m	0x7d	}
0x3e	>	0x4e	N	0x5e,	^	0x6e	n	0x7e	~
0x3f	?	0x4f	O	0x5f,	_	0x6f	o	0x7f	

How does it run on a computer ?

- The RAM (Random Access Memory) holds the sequence of bytes reprensenting the program to be run.

```
00000000  7f 45 4c 46 02 01 01 00  00 00 00 00 00 00 00 00
00000010  03 00 3e 00 01 00 00 00  f0 10 00 00 00 00 00 00
00000020  40 00 00 00 00 00 00 00  00 3a 00 00 00 00 00 00
00000030  00 00 00 00 40 00 38 00  0b 00 40 00 1d 00 1c 00
00000040  06 00 00 00 04 00 00 00  40 00 00 00 00 00 00 00
00000050  40 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00
00000060  68 02 00 00 00 00 00 00  68 02 00 00 00 00 00 00
00000070  08 00 00 00 00 00 00 00  03 00 00 00 04 00 00 00
00000080  a8 02 00 00 00 00 00 00  a8 02 00 00 00 00 00 00
00000090  a8 02 00 00 00 00 00 00  1c 00 00 00 00 00 00 00
```

How does it run on a computer ?

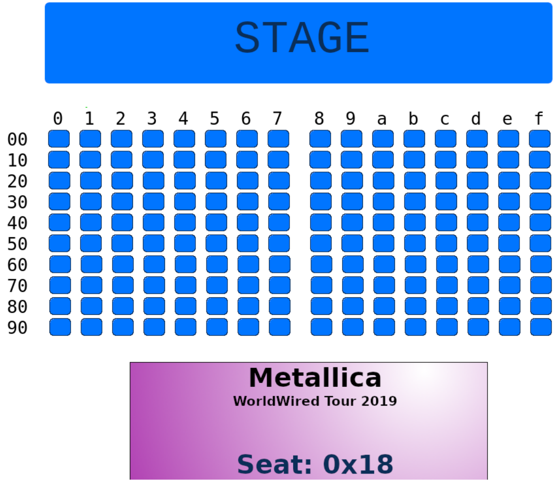
- The RAM (Random Access Memory) holds the sequence of bytes reprensenting the program to be run.

```
00000000  7f 45 4c 46 02 01 01 00  00 00 00 00 00 00 00 00
00000010  03 00 3e 00 01 00 00 00  f0 10 00 00 00 00 00 00
00000020  40 00 00 00 00 00 00 00  00 3a 00 00 00 00 00 00
00000030  00 00 00 00 40 00 38 00  0b 00 40 00 1d 00 1c 00
00000040  06 00 00 00 04 00 00 00  40 00 00 00 00 00 00 00
00000050  40 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00
00000060  68 02 00 00 00 00 00 00  68 02 00 00 00 00 00 00
00000070  08 00 00 00 00 00 00 00  03 00 00 00 04 00 00 00
00000080  a8 02 00 00 00 00 00 00  a8 02 00 00 00 00 00 00
00000090  a8 02 00 00 00 00 00 00  1c 00 00 00 00 00 00 00
```

offset 0x00000018 ?

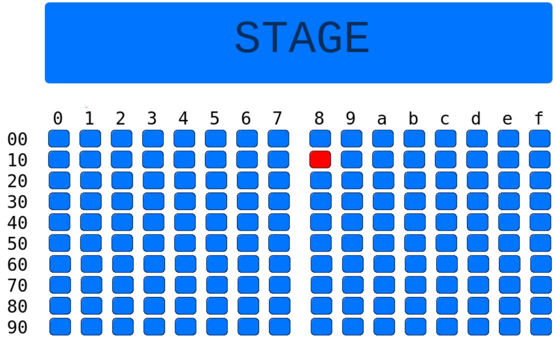
How does it run on a computer ?

- The RAM (Random Access Memory) holds the sequence of bytes reprensenting the program to be run.



How does it run on a computer ?

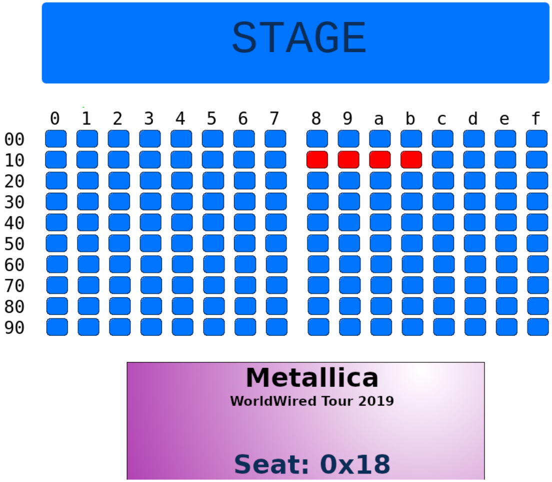
- The RAM (Random Access Memory) holds the sequence of bytes reprensenting the program to be run.



Metallica
WorldWired Tour 2019
Seat: 0x18

How does it run on a computer ?

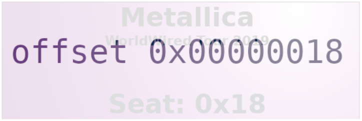
- The RAM (Random Access Memory) holds the sequence of bytes reprensenting the program to be run.



How does it run on a computer ?

- The RAM (Random Access Memory) holds the sequence of bytes reprensenting the program to be run.

```
          0  1  2  3  4  5  6  7   8  9  a  b  c  d  e  f
00000000  7f 45 4c 46 02 01 01 00  00 00 00 00 00 00 00 00
00000010  03 00 3e 00 01 00 00 00  f0 10 00 00 00 00 00 00
00000020  40 00 00 00 00 00 00 00  00 3a 00 00 00 00 00 00
00000030  00 00 00 00 40 00 38 00  0b 00 40 00 1d 00 1c 00
00000040  06 00 00 00 04 00 00 00  40 00 00 00 00 00 00 00
00000050  40 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00
00000060  68 02 00 00 00 00 00 00  68 02 00 00 00 00 00 00
00000070  08 00 00 00 00 00 00 00  03 00 00 00 04 00 00 00
00000080  a8 02 00 00 00 00 00 00  a8 02 00 00 00 00 00 00
00000090  a8 02 00 00 00 00 00 00  1c 00 00 00 00 00 00 00
```



offset 0x00000018 ?

How does it run on a computer ?

- The CPU (Central Processing Unit) :
  - reads the bytes as **instruction** ;
  - instructions are specific to a CPU;
  - uses **registers** to store intermediate results.

```
000010f0  f3 0f 1e fa 31 ed 49 89  d1 5e 48 89 e2 48 83 e4
00001100  f0 50 54 4c 8d 05 56 01  00 00 48 8d 0d df 00 00
00001110  00 48 8d 3d 58 ff ff ff  ff 15 c2 2e 00 00 f4 90
```

How does it run on a computer ?

- The CPU (Central Processing Unit) :
  - reads the bytes as **instruction** ;
  - instructions are specific to a CPU;
  - uses **registers** to store intermediate results.

```
000010f0  f3 0f 1e fa 31 ed 49 89  d1 5e 48 89 e2 48 83 e4
00001100  f0 50 54 4c 8d 05 56 01  00 00 48 8d 0d df 00 00
00001110  00 48 8d 3d 58 ff ff ff  ff 15 c2 2e 00 00 f4 90
```

```
00000000000010f0  endbr64
```

How does it run on a computer ?

- The CPU (Central Processing Unit) :
  - reads the bytes as **instruction** ;
  - instructions are specific to a CPU;
  - uses **registers** to store intermediate results.

```
000010f0  f3 0f 1e fa 31 ed 49 89  d1 5e 48 89 e2 48 83 e4
00001100  f0 50 54 4c 8d 05 56 01  00 00 48 8d 0d df 00 00
00001110  00 48 8d 3d 58 ff ff ff  ff 15 c2 2e 00 00 f4 90
```

```
00000000000010f0  endbr64
00000000000010f4  xor ebp, ebp
```

How does it run on a computer ?

- The CPU (Central Processing Unit) :
  - reads the bytes as **instruction** ;
  - instructions are specific to a CPU;
  - uses **registers** to store intermediate results.

```
000010f0  f3 0f 1e fa 31 ed 49 89  d1 5e 48 89 e2 48 83 e4
00001100  f0 50 54 4c 8d 05 56 01  00 00 48 8d 0d df 00 00
00001110  00 48 8d 3d 58 ff ff ff  ff 15 c2 2e 00 00 f4 90
```

```
00000000000010f0  endbr64
00000000000010f4  xor ebp, ebp
00000000000010f6  mov r9, rdx
```

x86 64 registers

In x86, the registers are:

- RIP: Instruction Pointer, holds the offset of the current instruction;
- RBP: Base Pointer, holds the offset of the *stack base*;
- RSP: Stack Pointer, holds the offset of the *stack*;
- RAX, RBX, RCX, RDX, RDI, RSI, R8 to R15: General purpose registers
- Flags : Special register

x86 instruction set

- Arithmetic operations

Instruction	Effects[1]
ADD RAX, RBX	RAX = RAX + RBX
MUL RAX, RBX	RAX = RAX $\star$ RBX
SUB RAX, RBX	RAX = RAX - RBX
CMP RAX, RBX	zf = 1 if RAX==RBX

- Memory access instructions

Instruction	Effects
MOV RAX, RBX	RAX = RBX
MOV RAX, [RBX]	RAX = mem at offset RBX
LEA RAX, [RBX + 3]	RAX = RBX + 3
PUSH RAX	@[RSP] = RAX; RSP=RSP-8

- Flow instructions

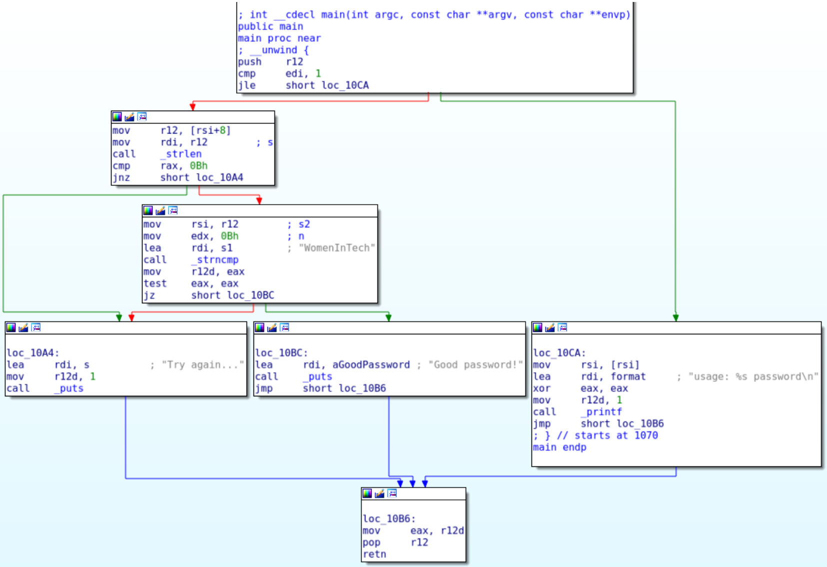
Instruction	Effects
JMP 0x12345	RIP = 0x12345
JZ 0x12345	if zf == 1 then RIP = 0x12345

[1]Effects on flags are ommited

Are you ready ?

```
.text:0000000000001070 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:0000000000001070                 push    r12
.text:0000000000001072                 cmp     edi, 1
.text:0000000000001075                 jle     short loc_10CA
.text:0000000000001077                 mov     r12, [rsi+8]
.text:000000000000107B                 mov     rdi, r12        ; s
.text:000000000000107E                 call    _strlen
.text:0000000000001083                 cmp     rax, 0Bh
.text:0000000000001087                 jnz     short loc_10A4
.text:0000000000001089                 mov     rsi, r12        ; s2
.text:000000000000108C                 mov     edx, 0Bh        ; n
.text:0000000000001091                 lea     rdi, s1         ; "WomenInTech"
.text:0000000000001098                 call    _strncmp
.text:000000000000109D                 mov     r12d, eax
.text:00000000000010A0                 test    eax, eax
.text:00000000000010A2                 jz      short loc_10BC
.text:00000000000010A4 loc_10A4:       lea     rdi, s          ; "Try again..."
.text:00000000000010AB                 mov     r12d, 1
.text:00000000000010B1                 call    _puts
.text:00000000000010B6 loc_10B6:       mov     eax, r12d
.text:00000000000010B9                 pop     r12
.text:00000000000010BB                 retn
.text:00000000000010BC loc_10BC:       lea     rdi, aGoodPassword ; "Good password!"
.text:00000000000010C3                 call    _puts
.text:00000000000010C8                 jmp     short loc_10B6
.text:00000000000010CA loc_10CA:       mov     rsi, [rsi]
.text:00000000000010CD                 lea     rdi, format     ; "usage: %s password\n"
.text:00000000000010D4                 xor     eax, eax
.text:00000000000010D6                 mov     r12d, 1
.text:00000000000010DC                 call    _printf
.text:00000000000010E1                 jmp     short loc_10B6
```

Are you ready ?



```
; int __cdecl main(int argc, const char **argv, const char **envp)
public main
main proc near
; __unwind {
push    r12
cmp     edi, 1
jle     short loc_10CA
```

```
mov     r12, [rsi+8]
mov     rdi, r12        ; s
call    _strlen
cmp     rax, 0Bh
jnz     short loc_10A4
```

```
mov     rsi, r12        ; s2
mov     edx, 0Bh        ; n
lea     rdi, s1         ; "WomenInTech"
call    _strncmp
mov     r12d, eax
test    eax, eax
jz      short loc_10BC
```

```
loc_10A4:
lea     rdi, s          ; "Try again..."
mov     r12d, 1
call    _puts
```

```
loc_10BC:
lea     rdi, aGoodPassword ; "Good password!"
call    _puts
jmp     short loc_10B6
```

```
loc_10CA:
mov     rsi, [rsi]
lea     rdi, format     ; "usage: %s password\n"
xor     eax, eax
mov     r12d, 1
call    _printf
jmp     short loc_10B6
; } // starts at 1070
main endp
```

```
loc_10B6:
mov     eax, r12d
pop     r12
retn
```

Tools needed

- IDA Freeware - Download: https://bit.ly/37sjCVz

- Softwares - Download: https://bit.ly/2QWiF1E

Now you are ready !

Analyze the second program, and find the password to unlock it !

Memo:

- [RSI + 8]: Offset of input password
- RDI: first argument of function
- RAX: return value of function
- XOR EAX, EAX: sets EAX value to 0
- NOP: do nothing

Understanding user input

```
          0  1  2  3  4  5  6  7   8  9  a  b  c  d  e  f
00012340  6d 79 69 6e 70 75 74 70  61 73 73 77 6f 72 64 00  |myinputpassword.|
```

```
rbx = 0x12340
rax = 0

[rbx + rax] ?
```

Understanding user input

```
          0  1  2  3  4  5  6  7   8  9  a  b  c  d  e  f
00012340  6d 79 69 6e 70 75 74 70  61 73 73 77 6f 72 64 00  |myinputpassword.|

rbx = 0x12340
rax = 0

[rbx + rax] ?
```

Understanding user input

```
          0  1  2  3  4  5  6  7   8  9  a  b  c  d  e  f
00012340  6d 79 69 6e 70 75 74 70  61 73 73 77 6f 72 64 00  |myinputpassword.|
```

```
rbx = 0x12340
rax = 1

[rbx + rax] ?
```

Understanding user input

```
            0  1  2  3  4  5  6  7   8  9  a  b  c  d  e  f
00012340   6d 79 69 6e 70 75 74 70  61 73 73 77 6f 72 64 00  |myinputpassword.|

rbx = 0x12340
rax = 1

[rbx + rax] ?
```

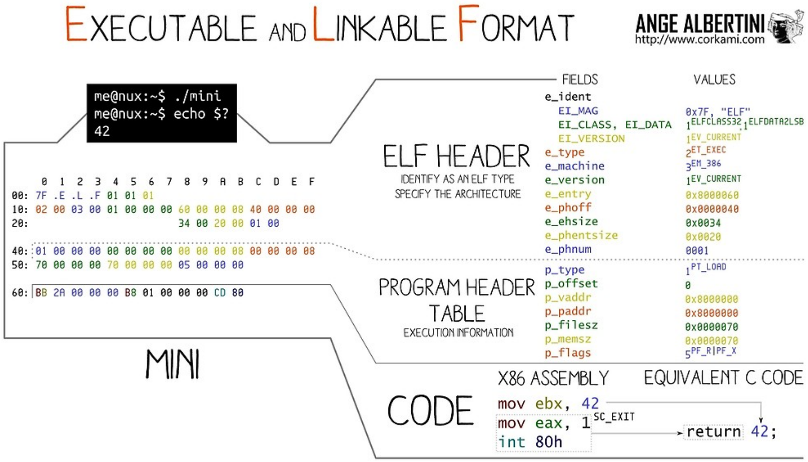
Conclusion

Now you know:

- Hexadecimal;
- How a CPU run a program ;
- How to understand assembly langage.

How to go further ?

- Challenges platform (microcorruption.com);
- If you are stuck, read the write-ups !!



EXECUTABLE AND LINKABLE FORMAT

ANGE ALBERTINI
http://www.corkami.com

```
me@nux:~$ ./mini
me@nux:~$ echo $?
42
```

MINI

```
     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
00: 7F .E .L .F 01 01 01
10: 02 00 03 00 01 00 00 00 60 00 00 08 40 00 00 00
20:                         34 00 20 00 01 00

40: 01 00 00 00 00 00 00 00 00 00 00 08 00 00 00 08
50: 70 00 00 00 70 00 00 00 05 00 00 00

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

ELF HEADER
IDENTIFY AS AN ELF TYPE
SPECIFY THE ARCHITECTURE

PROGRAM HEADER TABLE
EXECUTION INFORMATION

FIELDS	VALUES
e_ident	
EI_MAG	0x7F, "ELF"
EI_CLASS, EI_DATA	1 ELFCLASS32, 1 ELFDATA2LSB
EI_VERSION	1 EV_CURRENT
e_type	2 ET_EXEC
e_machine	3 EM_386
e_version	1 EV_CURRENT
e_entry	0x8000060
e_phoff	0x0000040
e_ehsize	0x0034
e_phentsize	0x0020
e_phnum	0001
p_type	1 PT_LOAD
p_offset	0
p_vaddr	0x8000000
p_paddr	0x8000000
p_filesz	0x0000070
p_memsz	0x0000070
p_flags	5 PF_R|PF_X

CODE

X86 ASSEMBLY

```
mov ebx, 42
mov eax, 1   SC_EXIT
int 80h
```

EQUIVALENT C CODE

```
return 42;
```