0x0101 Shellcoding-Lab 32Bit AntiCast – adduser to /etc/passwd

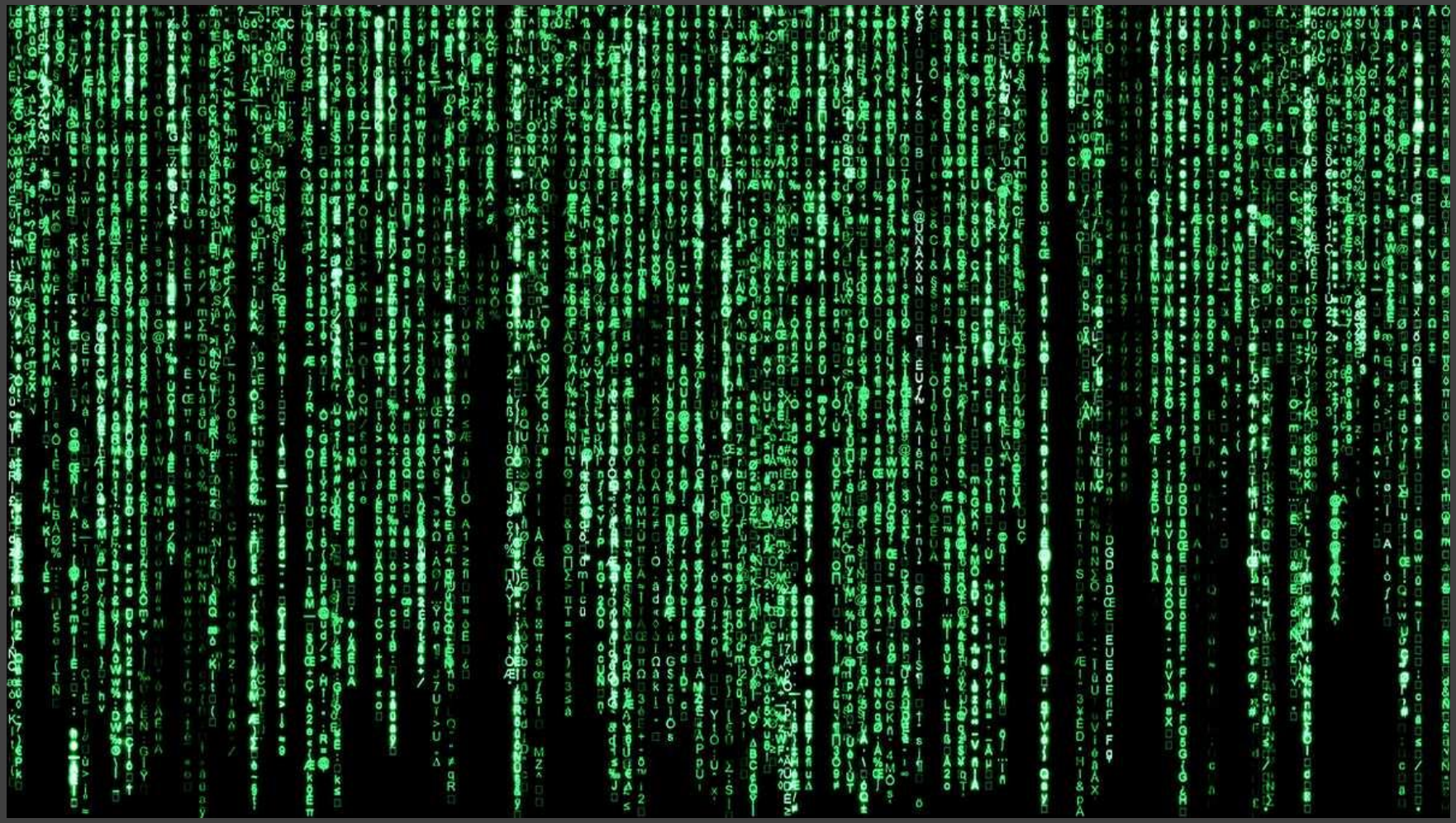
By Marco Lux

Chapter 0x0 (Intro)

Welcome Everyone ☺

- Who am I?
- Why Assembly?
- Why Shellcoding?
- Why x86_32?

Assembly



Assembly

- Assembly is the last language for mortal human beings to learn
  - Next step would be binary
- Assembly is close to the operating system and the hardware
- Understanding Assembly gives you better understanding of high-level-languages
- Assembly is easy btw
- But becomes quickly complex and hard to manage
- Debugging pure assembly is fun and can be surprising

Shellcode

Shellcoding

- Shellcoding is not shellscripting – Really its not ☺
- Taking the compiled opcodes which are instructions to the CPU
- That's not only pretty low-level but also quite cool
  - "Hey there, I'am a shellcoder!1"

Shellcoding

- Why would you want to write your own shellcode
  - Understand how things are working
  - Msfvenom is not enough
  - Custom shellcode supports your specific tasks
  - You need custom shellcode or encoders to circumvent AV/EDR
  - Built up a foundation on low-level analysis and binary exploitation

X86_32

- Isn't it a bit outdated? Ever heard of 8086 or 286, based on 16Bit?
  - Even today CPU starts in legacy – 16 Bit - Mode
- 32Bit was one of the most common platforms for over a decade
- Instructionset will aid you also on 64Bit
  - Actually, you even use 32Bit instructions on 64Bit
- A lot of software still comes as 32Bit
- It's still assembly and shellcode
  - Concepts, Ideas and Techniques can be absorbed and transferred
  - The idea of a NOP, a Debug Instruction or JMP is still valid on 64Bit or ARM
- It's building up your foundational skills

Chapter 0x1 (Shellcode Quick Overview)

- Registers
- Basics in Assembly
- Strings
- Syscalls

Registers

CPU Registers

- EAX → Accumulator
- EBX → Baseregister
- ECX → Counter
- EDX → Data
- ESI → Source Index
- EDI → Destination Index

- ESP → StackPointer
- EBP → BasePointer
- EIP → Instruction Pointer

32 BIT Registers

CPU General Purpose Registers

Reg	Accu	Base	Count	Data	Source	Dest.
32Bit	EAX	EBX	ECX	EDX	ESI	EDI
16Bit	AX	BX	CX	DX	SI	DI
8Bit High	AH	BH	CH	DH		
8Bit Low	AL	BL	CL	DL		

Basic Assembly Instructions

Basic Assembly Instructions

xor	eax	eax
xor	0x12345678	0x12345678
Result	0x00000000	0x00000000

- XOR, Exclusive OR
- Boolean logic
- Major use in shellcoding:
  - Null registers (example)
  - Encoder/Decoder
  - Encryption

Basic Assembly Instructions

mov	eax	ebx
mov	0x12345678	0xC0FEBABE
Result	0xC0FEBABE	0xC0FEBABE

- MOV
- Its copy
- Major use in Shellcode:
  - It is a basic operation, you need for almost everything
  - Place values in registers

Basic Assembly Instructions

Register	Eax	0x41424344
push	eax	/
Result on stack	0x41424344	/

- PUSH, Opposite POP
- Place something on the stack
- You can also push a value to the stack
- Major use in shellcoding:
  - Strings, Values, places data/code on the stack

Basic Assembly Instructions

Register	/
push	0xAABBCCDD
Result on stack	0xAABBCCDD

- PUSH, Opposite POP
- Place something on the stack
- You can also push a value to the stack
- Major use in shellcoding:
  - Strings, Values, places data/code on the stack

Basic Assembly Instructions

Stack	0x41424344
pop	eax
Result in EAX	0x41424344

- POP, Opposite PUSH
- Take from the stack
- You can pop into a register
- Major use in shellcoding:
  - Whatever has been saved on the stack, place it in a register

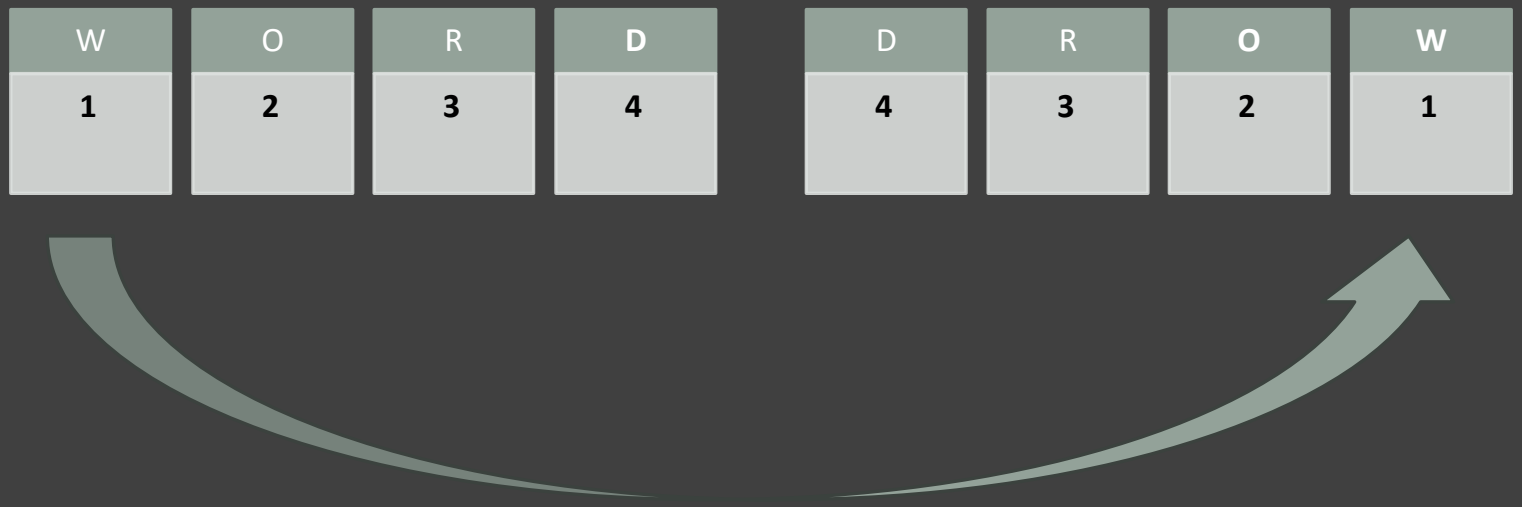
Basic Assembly Instructions (Demo Instructions)

Strings

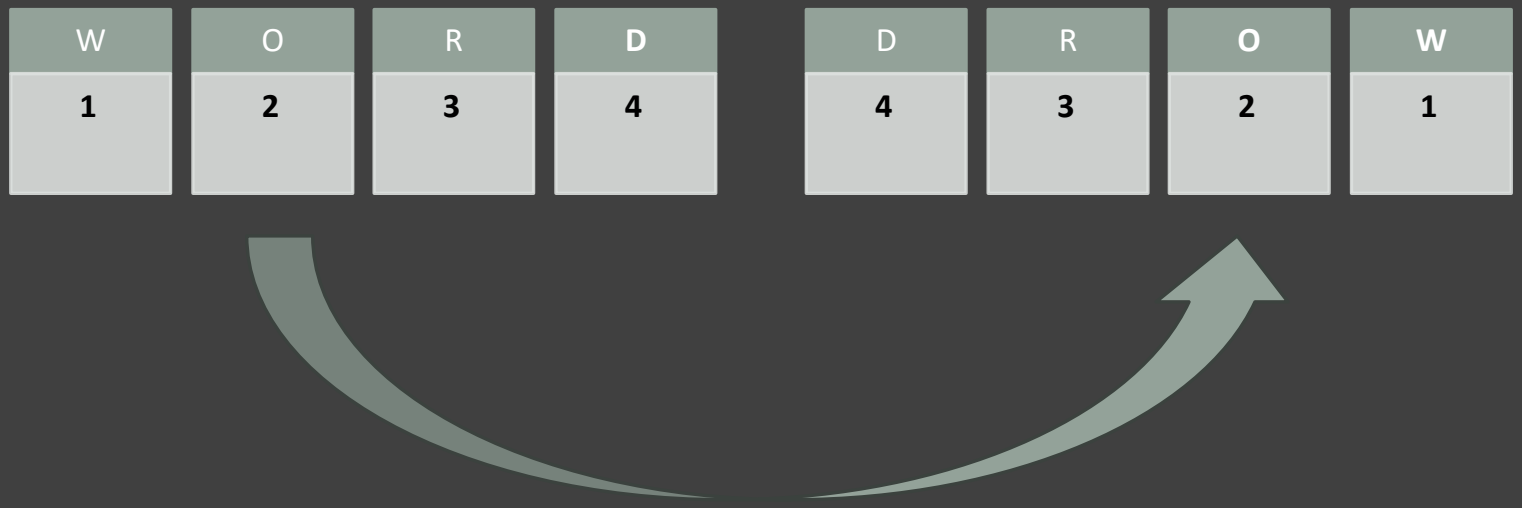
Strings

- We can place also Strings on the stack
- As x86_32 is LE (little endian):
  - Strings need to be reversed before pushing
- Also we want to convert them to hexadecimal

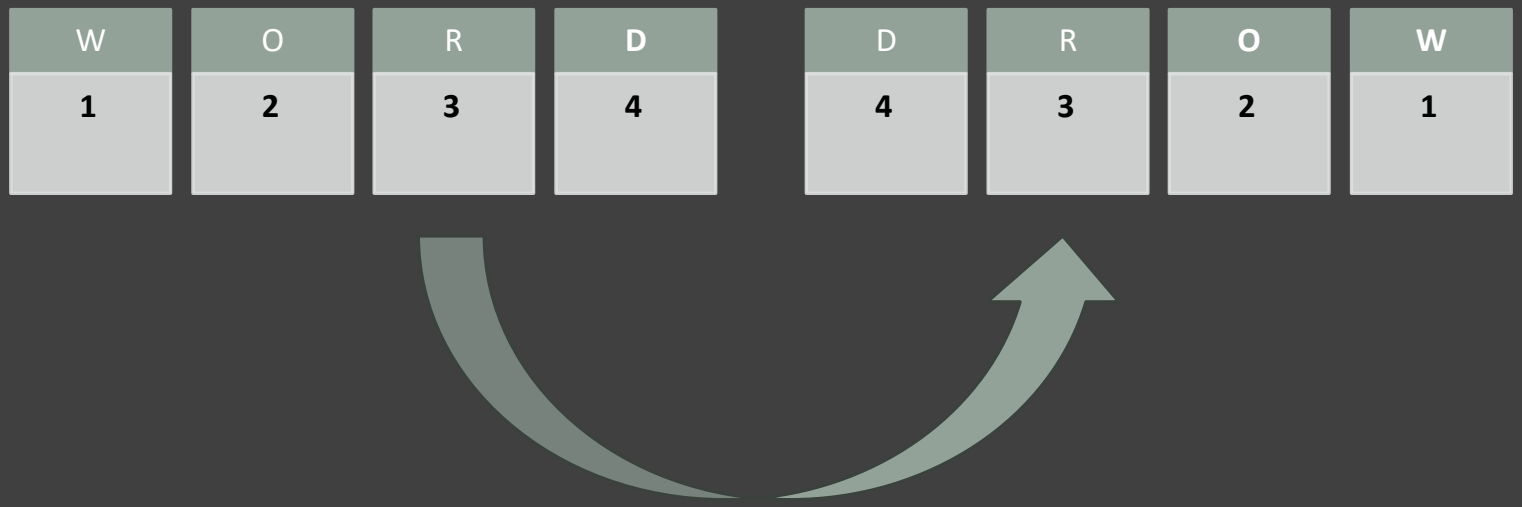
Strings - Reverse



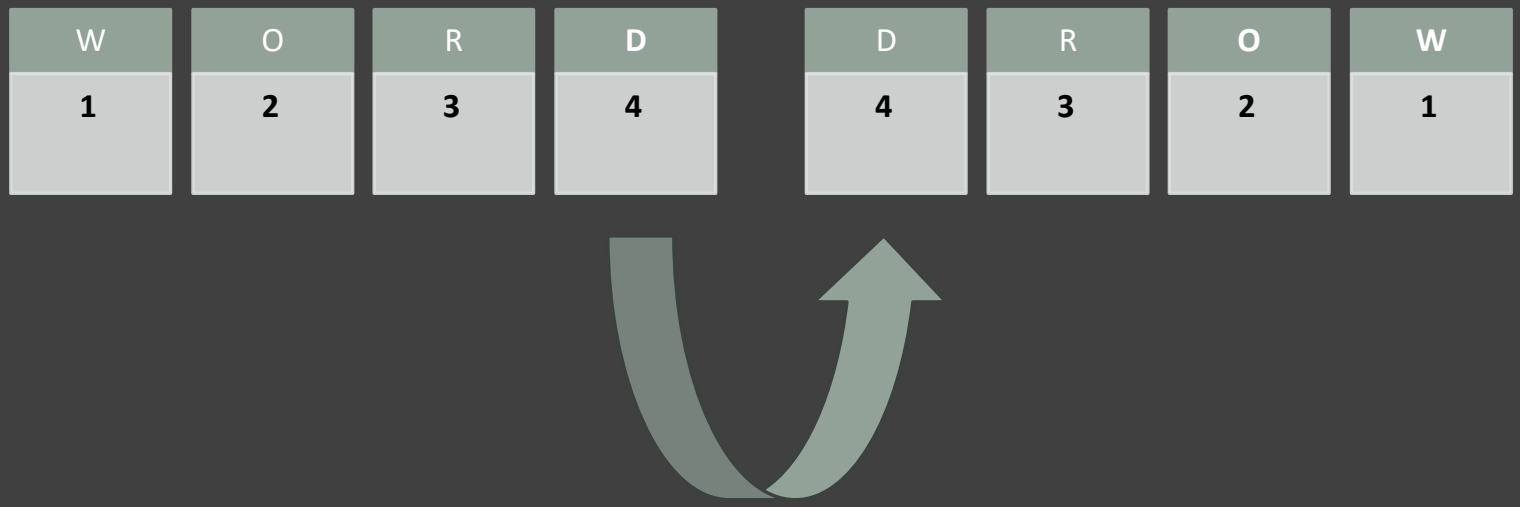
Strings - Reverse



Strings - Reverse

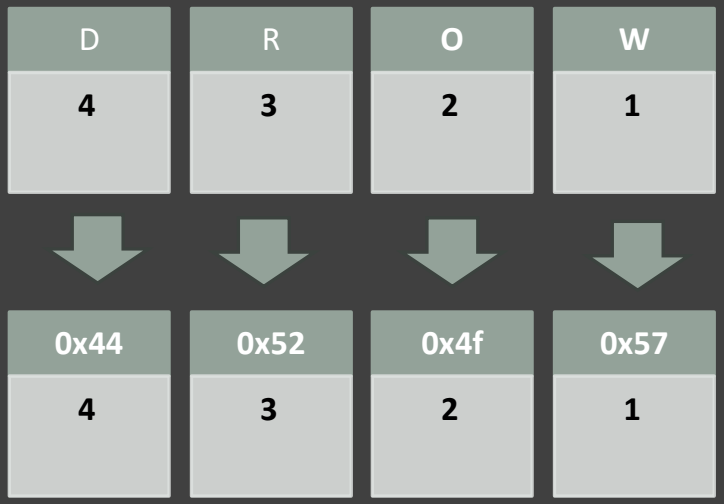


Strings - Reverse



Strings - Convert

- Convert to hexadecimal
- Convert DROW -> 0x44524f57



Strings - Demo

Syscalls

Syscalls

- Last but not the least in our overview: Syscalls
  - System calls, for all system operations
  - Like: open/close/write/read/exit/execute
- Interface is logic and simple
- Registers are used for syscall number and arguments
- Return value is also in a register

Syscall Examples

	32BIT
exit	1
read	3
write	4
open	5
close	6
execve	11
chdir	12
chmod	15
setuid	23
kill	37
reboot	88
socket	102
connect	102
accept	102
bind	102
listen	102

	64Bit
exit	60
read	0
write	1
open	2
close	3
execve	59
chdir	80
chmod	90
setuid	105
kill	62
reboot	169
socket	41
connect	42
accept	43
bind	49
listen	50

Syscall

Register	EAX	EBX	ECX	EDX	ESI	EDI	EBP
Value	Syscall	Arg1	Arg2	Arg3	Arg4	Arg5	Arg6

Register	EAX
Result	Result of the syscall

Syscall: exit

EAX	EBX
exit	int status

- man 2 exit
- Simple but important call

Syscall: exit

EAX	EBX
1	4

- Find ‘unistd_32.h’
  - find /usr/include –name unistd_32.h
- Syscall Exit is 1
- Errorcode: 4
  - You can choose any return code ☺

Chapter 0x2 (Shellcode Time)

- Lets have a look at an actual shellcode task

adduser to /etc/passwd

adduser to /etc/passwd

- man 2 open / write / close
- Lets have a look at the syscall: open
- open(const char *pathname, int flags);

EAX	EBX	ECX
open	char *pathname	int flags

adduser to /etc/passwd

- Next is the syscall write:
- ssize_t write(int fd, const void *buf, size_t count);

EAX	EBX	ECX	EDX
write	Fd /Filedescriptor	*buf	count/length

adduser to /etc/passwd

- Next is the syscall close:
- int close(int fd);

EAX	EBX
close	Fd /Filedescriptor

adduser to /etc/passwd

- Check following include files:

/usr/include/bits/fcntl.h

/usr/include/bits/fcntl-linux.h

- And find the passage:

define O_CREAT 0100

define O_EXCL 0200

define O_NOCTTY 0400

define O_TRUNC 01000

 # define O_APPEND 02000 <--- we want to append

define O_NONBLOCK 04000

- How to convert this?

$ gdb --quiet --batch -ex 'print /x 02000 | 01'

$1 = 0x401

adduser to /etc/passwd

- You can use the crypt_des_tool.py
- ./crypt_des_tool.py hack3r

- Convert the string to something fitting your assembly code
- push_string Tool supports your efforts

adduser to /etc/passwd

- Watch out for:
  - End of the strings
  - Lonely bytes (push byte)
  - Missing Newline

adduser to /etc/passwd

- File: shellcode/bad_adduser_passwd/bad_adduser_passwd.asm
- Objective:
  - Fix the bad_adduser_passwd
  - Open the passwd file
  - Write a new passwd entry for a privileged user
  - Close the file cleanly.
  - Check with strace if everything worked.
  - Check with su / login if you can login as the new user
- Extramile:
  - Add password not to /etc/passwd but /etc/shadow
  - Add not crypt recent crypto to the file

Outlook

- This was a quick intro to shellcoding at 32bit level
- We will cover a lot more topics in our course
- More than 30 hands-on codes
- And a steep learning curve

Fin

- Thank you for listening!
- Questions?
- Contact: ping@curesec.com