Hacks and exploits

Julio Montes
imc.coder@gmail.com

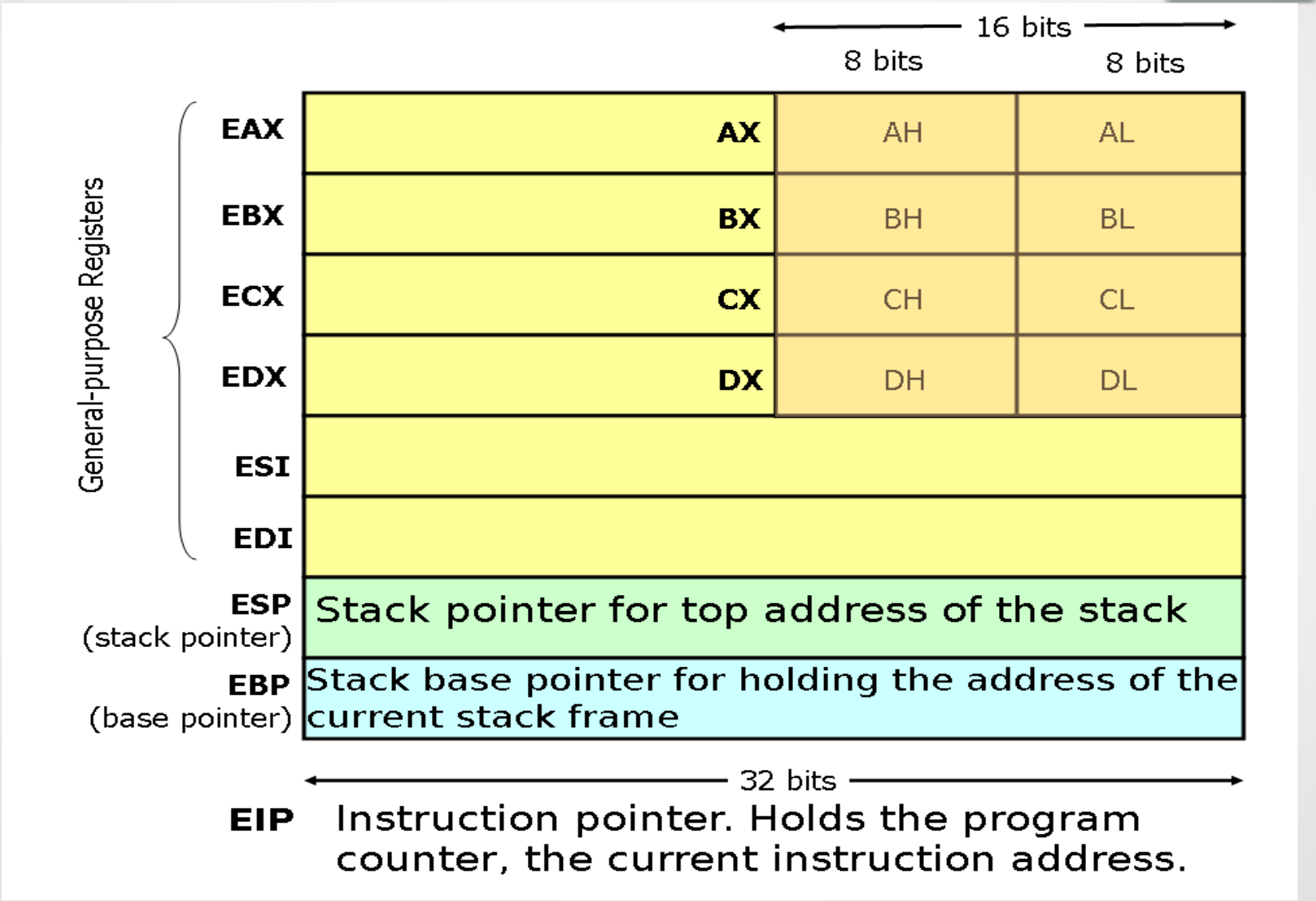
3DS Exploit (ninjhax)

- https://www.youtube.com/watch?v=iKjuy3-z054

CPU Registers

The registers are like variables built in the processor. Using registers instead of memory to store values makes the process faster and cleaner.

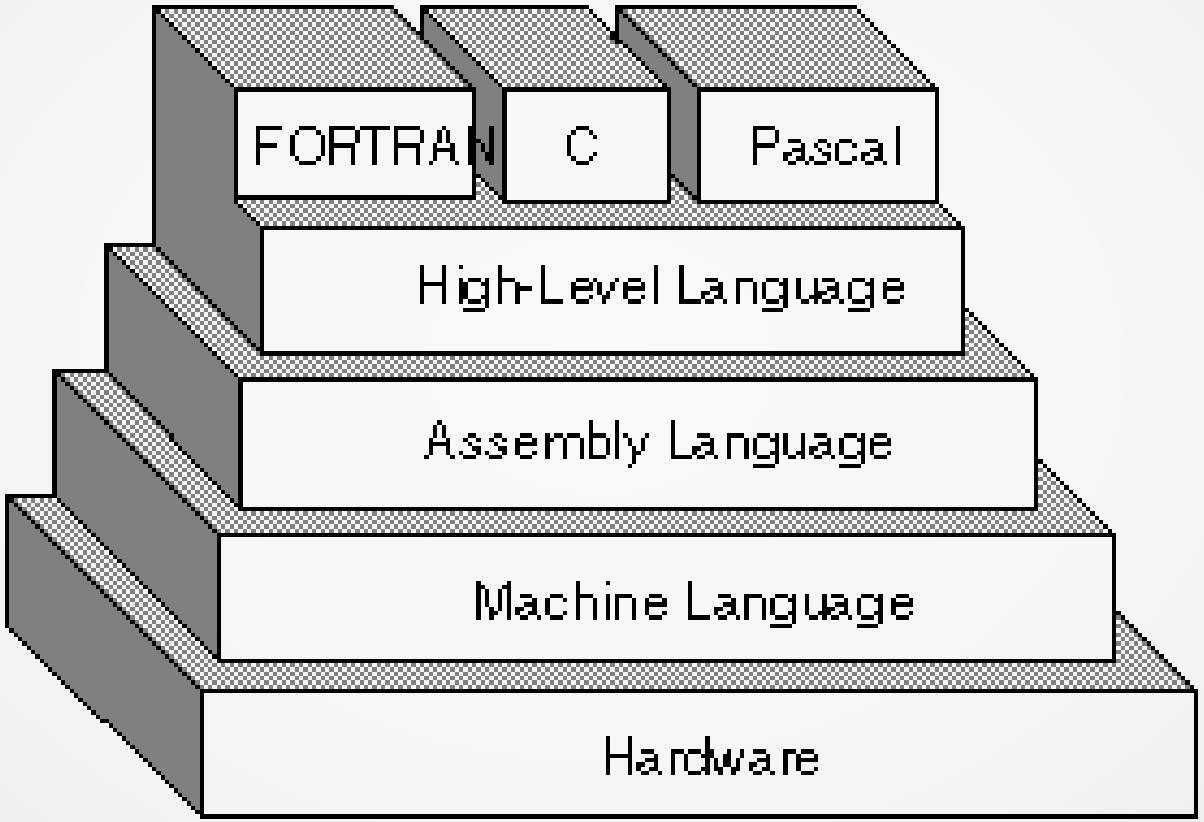
CPU Registers



Assembler code

Nowadays most personal devices (computer, smart cellphones, embedded systems and others) have a microprocessor that manages the device's arithmetical, logical and control activities. Each family of microprocessors has its own set of instructions for handling operations and perform jobs. These set of instructions are called “machine language instructions”.
Microprocessors understand only machine language instructions which are string of 1's and 0's. However machine language is too complex for software developers. So, the low-level assembly language is designed for a specific family of processors that represent various instructions in symbolic code.

Assembler code



Assembler Instructions X86

Machine instructions generally fall into three categories:

- **Data movement**
  - mov, push, pop, lea
- **Arithmetic/logic**
  - and, or, xor, add, sub, not
- **Control-flow**
  - jmp, call, ret

Assembly Sections

.BSS (Block Started by Symbol – Better Save Space) contains all global and static variables that are initialized to zero or don't have explicit initialization.

- static int i;

.DATA contains all global and static variables that are initialized with a non-zero value

- int i = 8;
- char *foo = “bar”;

.TEXT is where are placed the assembly language instructions

Linux Syscall

The system call is the fundamental interface between an application and the Linux kernel.

System calls are generally not invoked directly, but rather via wrapper functions in glibc (or perhaps some other library). Often, but not always, the name of the wrapper function is the same as the name of the system call that it invokes. For example, glibc contains a function truncate() which invokes the underlying "truncate" system calls.

- http://man7.org/linux/man-pages/man2/syscalls.2.html
- http://syscalls.kernelgrok.com/

Making syscalls

int 0x80, on both x86 and x86_x64 systems

Syscall	Param1	Param2	Param3	Param4	Param5	Param6
eax	ebx	ecx	edx	esi	edi	ebp

Return value
eax

syscall, only on x86_64, It does not access the interrupt descriptor table and is faster

Syscall	Param1	Param2	Param3	Param4	Param5	Param6
rax	rdi	rsi	rdx	rcx	r8	r9

Return value
eax

Shellcode

- The Shellcode is a small piece of code used as payload in the exploitation of a software vulnerability. It is called shellcode because it typically starts a command shell.

  Execve /bin/sh:
  "\x31\xc9\xf7\xe1\xb0\x0b\x51\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\xcd\x80"

How get our shellcode

- **exploit.s x86**

```
section .text
global _start
_start:    ;main
        xor eax, eax  ;set eax to zero
        xor ebx, ebx  ;set ebx to zero
        mov al, 1  ;1 is sys_exit
        int 0x80    ;interrupt
```

- **exploit.s x86_64**

```
section .text
global _start
_start:    ;main
        xor rax, rax    ;set rax to zero
        xor rdi, rdi    ;set rdi to zero
        mov al, 1  ;1 is sys_exit
        syscall     ;syscall
```

- **Build the object file**
  - $ nasm -f elf64 -o exploit.o exploit.s
- **Link the object file**
  - $ ld exploit.o -o exploit
- **Get shellcode**
  - $ objdump -d exploit | grep "[0-9a-f]:" | cut -d':' -f2 | sed 's/\t/\\x/' | sed 's/\t/-/g' | cut -d'-' -f1 | sed 's/ /\\x/g' | egrep -o "\\\x[[:alnum:]]+" | tr -d '\n'

Questions

- **Which are the differences between exploit.o and exploit?**
  - exploit has more assembly sections than exploit.o: .BSS
- **Is exploit Linked with glibc or any other library?**
  - No, exploit doesn't use libraries
    - **Why?**
      - exploit uses the linux syscalls directly, without wrappers

References

- http://www.cs.virginia.edu/~evans/cs216/guides/x86.html
- http://www.tutorialspoint.com/assembly_programming/assembly_quick_guide.htm
- http://syscalls.kernelgrok.com/
- http://man7.org/linux/man-pages/man2/syscalls.2.html
- http://www.eecg.toronto.edu/~amza/www.mindsec.com/files/x86regs.html