Jazyk symbolickych instrukcii (assembler)

Bezpecnost informacnych systemov z pohladu praxe

Peter Svec

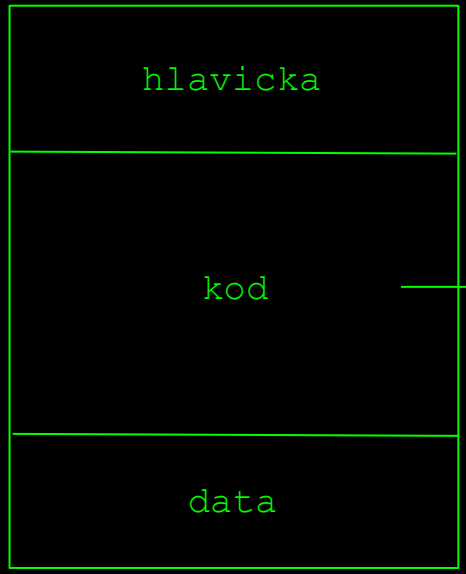
0x01

>motivacia

>zaklady su potrebne pre reverzne inzinierstvo/exploitaciu
>je vsade
>znalost zakladov pomaha aj pri programovanie v jazykoch vyssej urovne
>samozrejme programovanie zariadeni, ovladacov, optimalizacie, atd…

>spustitelny subor

>ELF format (**E**xecutable and **L**inkable **F**ormat)



0x20 0x75 0x6c → AND BYTE PTR[RBP+0x6c], DH

0010 0000 0111 0101 0110 1100

>jazyk symbolickych instrukcii

>tri zakladne koncepty

>instrukcie:
    >manipulacia s datami (matematicke operacie,...)
    >kontrola toku programu
    >systemove volanie
>registre:
    >ukladanie docasnych dat
>pamat:
    >samotne instrukcie
    >zasobnik

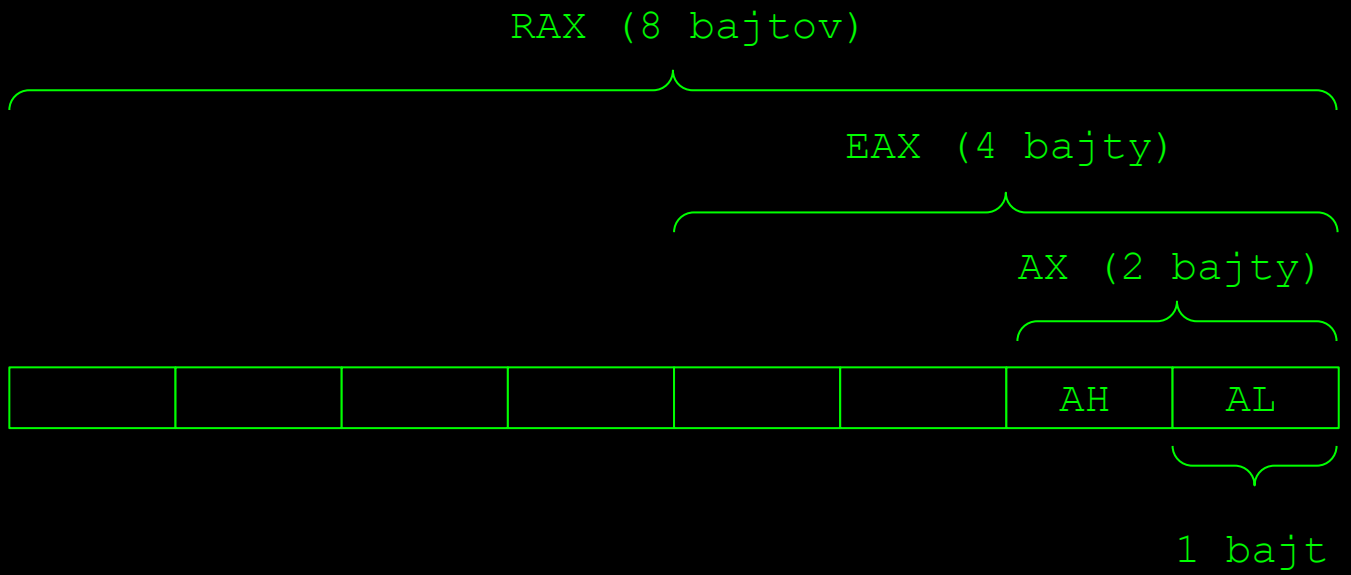
>registre

>maly a rychly ulozny priestor (8 bajtov na x86_64)

>vseobecne registre:
- >8086: AX, BX, CX, DX, **SP**, **BP**, SI, DI
- >x86: EAX, EBX, ECX, EDX, **ESP**, **EBP**, ESI, EDI
- >x86_64: RAX, RBX, RCX, RDX, **RSP**, **RBP**, RSI, RDI, R8, R9, R10, R11, R12, R13, R14, R15

>adresa nasledujucej instrukcie:
- > **IP**(8086), **EIP**(x86), **RIP**(x86_64)

>registre




```
>RAX = 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff
>AH  = 0xcc
>RAX = 0xff 0xff 0xff 0xff 0xff 0xff 0xcc 0xff
>EAX = 0xaa 0xaa 0xaa 0xaa
>RAX = 0x00 0x00 0x00 0x00 0xaa 0xaa 0xaa 0xaa
```

>instrukcie

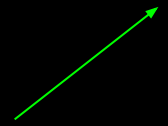
>vseobecna forma (typicky 0-2 operandy):

```
INSTRUKCIA OPERAND, OPERAND,...
```

>instrukcia -> co sa ma vykonat (scitanie? nasobenie? skok?)
>operand -> nad akymi datami sa ma vykonat instrukcia

```
mov rcx, rax
add rcx, 5
cmp rcx, rbx
je navestie
navestie:
inc rcx
```

```
mov rax, rbx
mov rax, [rbx+4]
add rax, rbx
mul rsi
inc rax
inc [rax]
```



intel syntax

>sposoby adresovania

>priame

```
MOV RAX, 0x45
```

>register

```
MOV RAX, RBX
```

>pamat

```
MOV RAX, [RBX]
```

>pomocne direktivy

```
mov [0x1337000], 1
```

```
>0x01?
>0x00 0x01?
>0x00 0x00 0x00 0x01??
```

```
mov BYTE PTR [0x1337000], 1     BYTE: 8 bitov (1 bajt)
mov WORD PTR [0x1337000], 1     WORD: 16 bitov (2 bajty)
mov DWORD PTR [0x1337000], 1    DWORD: 32 bitov (4 bajty)
mov QWORD PTR [0x1337000], 1    QWORD: 64 bitov (8 bajtov)
```

>tok programu

>skoky prebiehaju na zaklade registra RFLAGS
>register sa nastavuje pri:
    >aritmeticke operacie (add, mul, div, sub,...)
    >instrukcia cmp rax, rbx (rax - rbx)
    >instrukcia test rax, rbx (rax & rbx)

>RFLAGS priklady bitov:
    >ZF - Zero Flag
    >OF - Overflow Flag
    >SF - Signed Flag
    >CF - Carry Flag

```
cmp rax, rbx
je navestie
// ZF = 1
```

```
cmp rax, rbx
jb navestie
// CF = 1
```

>dalsie mozne skoky: jmp, jne, jg, jl, jle, jge,...

>systemove volania

>interakcia s OS
>open, read, write, fork, exec,...
>instrukcia **syscall** (iba na x86_64)
>priklad:
>chceme zavolat systemove volanie exit s hodnotou 42
>cislo systemoveho volania je 60[1]

```
mov rax, 60
mov rdi, 42
syscall      // navratova hodnota v RAX
```

[1]https://blog.rchapman.org/posts/Linux_System_Call_Table_for_x86_64/

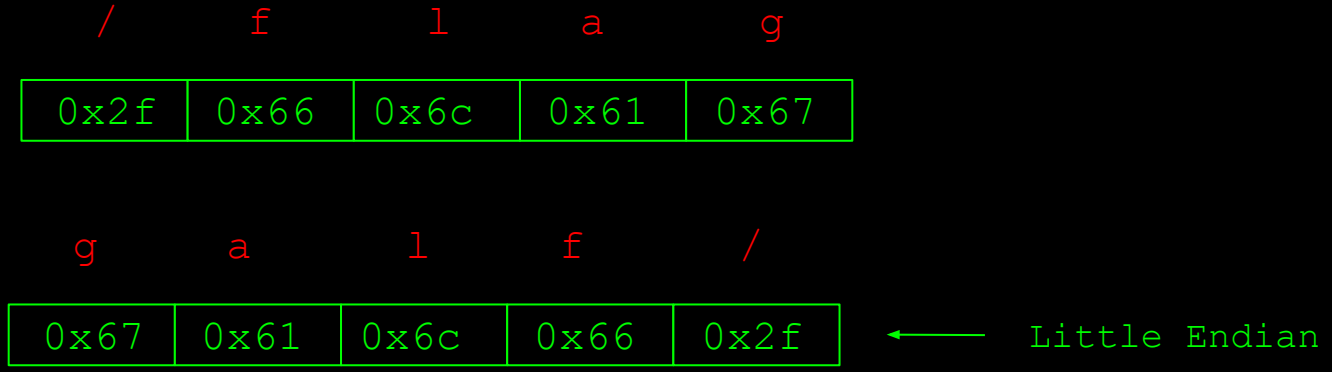
>zasobnik

>zasobnikovy ramec pre volanie funkcii
>obsahuje:
    >kde zacina zasobnikovy ramec predchadzajuceho volania
    >lokalne premenne pre funkciu
    >navratova adresa (navrat z funkcie - instrukcia RET)
>registre ovladajuce zasobnik:
    >RSP -> vrch zasobnika (**S**tack **P**ointer)
    >RBP -> spodok zasobnika (**B**ase **P**ointer)
>praca so zasobnikom: PUSH, POP

```
push rax        pop rax
push 0xff       pop [rcx]
```

>endianita

>data na x86 architekturach su ulozene v opacnom poradi
>Little Endian



>dalsie instrukcie

>logicke operacie:
    >XOR, AND, OR, NEG,...

>bitove posuny:
    >SHR, SHL, ROL, ROR,...

>volanie funkcie:
    >CALL, RET

>nacitanie adresy:
    >LEA (Load Effective Address) -> LEA, [EAX + EBX]

>ziadna operacia:
    >NOP (No Operation)

>hello world

> hello.s

```
.global _start
_start:
.intel_syntax noprefix
    mov rax, 1
    mov rdi, 1
    lea rsi, [rip+hello]
    mov rdx, 1000
    syscall
hello:
    .string “Hello World!”
```

> gcc -nostdlib main.s -o hello

>dalsie zdroje

>rappel

> https://github.com/yrp604/rappel

>dokumentacia instrukcii

> https://www.felixcloutier.com/x86/