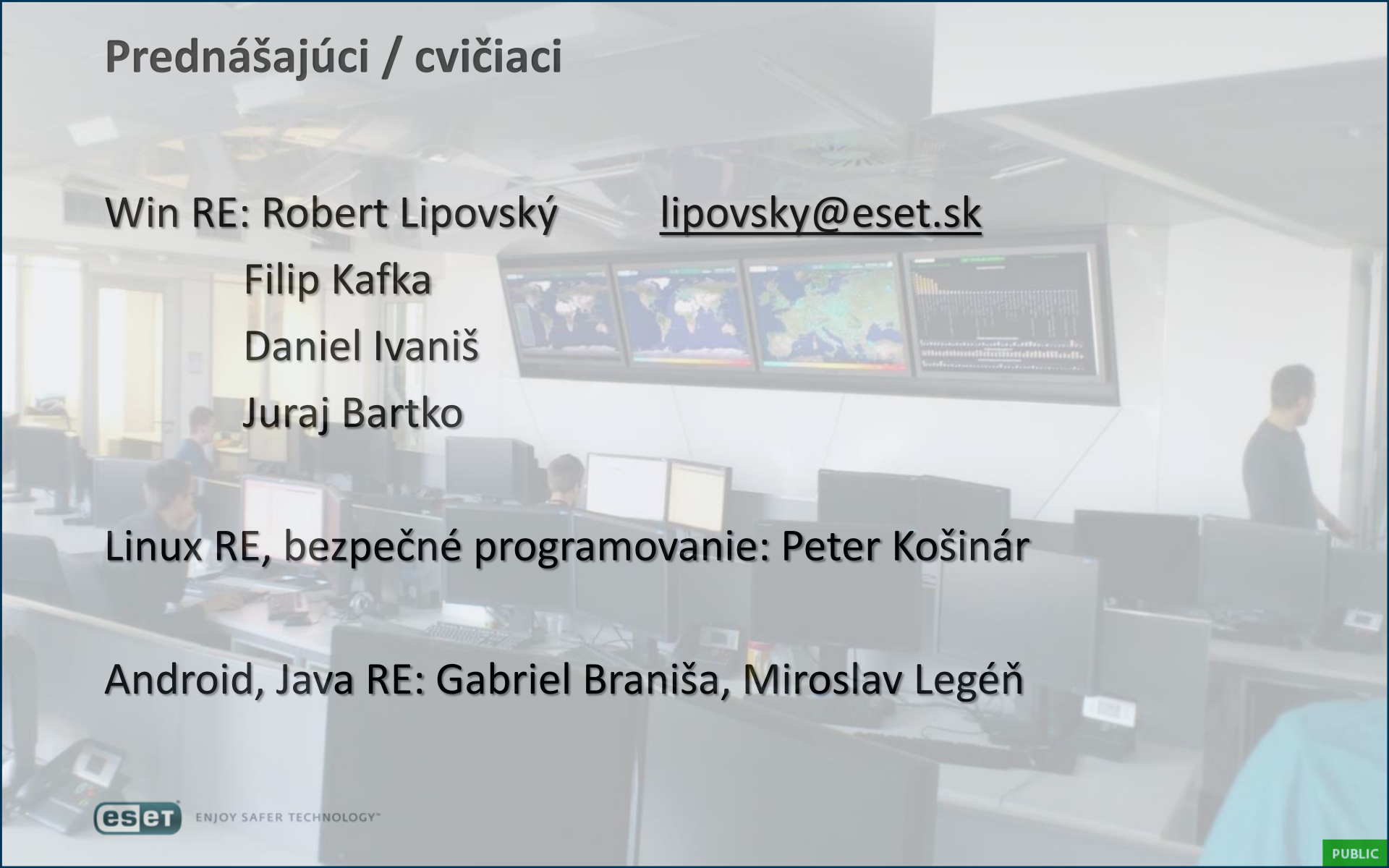
Základy reverzného inžinierstva

18.2.2016

Prednášajúci / cvičiaci



Win RE: Robert Lipovský lipovsky@eset.sk

Filip Kafka

Daniel Iványi

Juraj Bartko

Linux RE, bezpečné programovanie: Peter Košinár

Android, Java RE: Gabriel Braniša, Miroslav Legéň

Harmonogram & Hodnotenie

Týždne (cca):

- 1. 18.2 Uvod
- 2 – 7. Windows RE
  - 22.2 vstupný test, prvé cvičenie štvrtok 25.2 namiesto prednášky.
  - vseobecne vedomosti, pouzivanie toolov – disassemblery, debuggery (OllyDbg, IDA Pro)
  - anti-debug triky, unpacking, kernel drivery
  - 50b.
- 8 – 10. Linux RE, bezpecne programovanie, scriptove zranitelnosti, reverzovanie naburaneho serveru
  - 20b.
- 11 – 12 Android RE
  - 15b.

15b. Zaverecny test

Na čo to je dobré?

- Analýza malwaru
- Debugging = hľadanie a odstraňovanie bugov ☺
- Výskum zraniteľností
- Zábava ☺
- atď

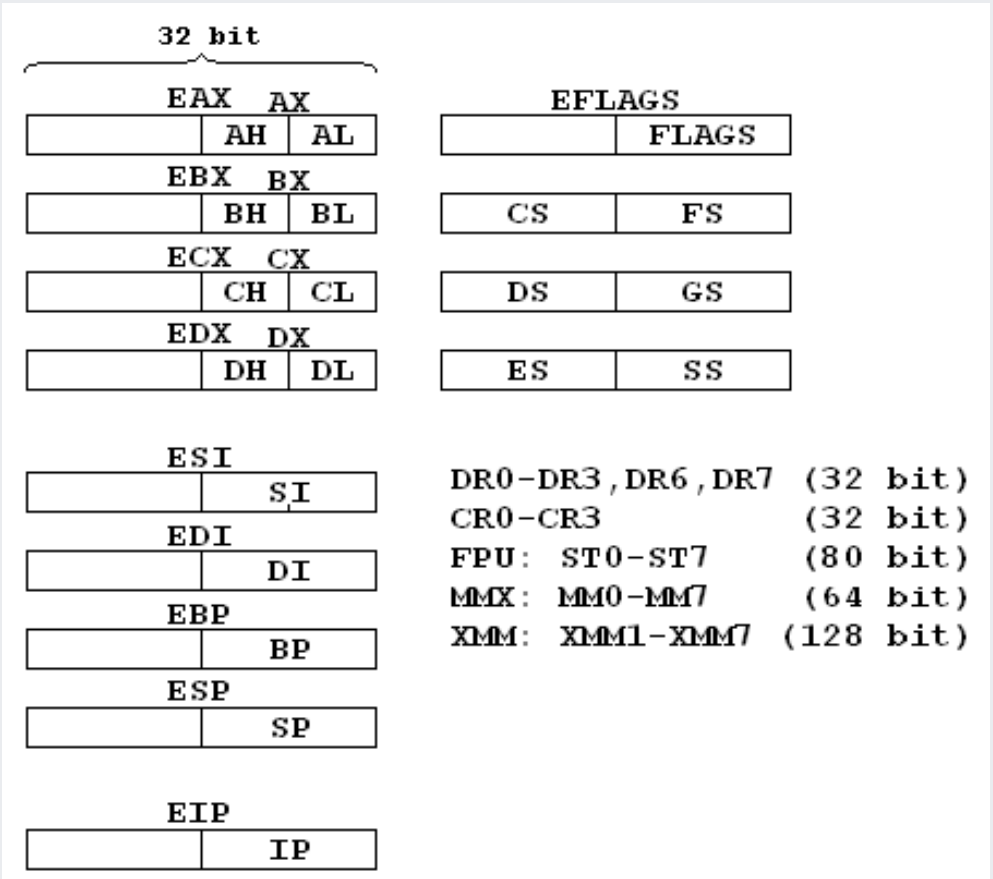
Ukážky

(na čo je to dobré a čo nás čaká)

x86 ASM

(čo by sme mali vedieť)

x86 registre, dátové typy



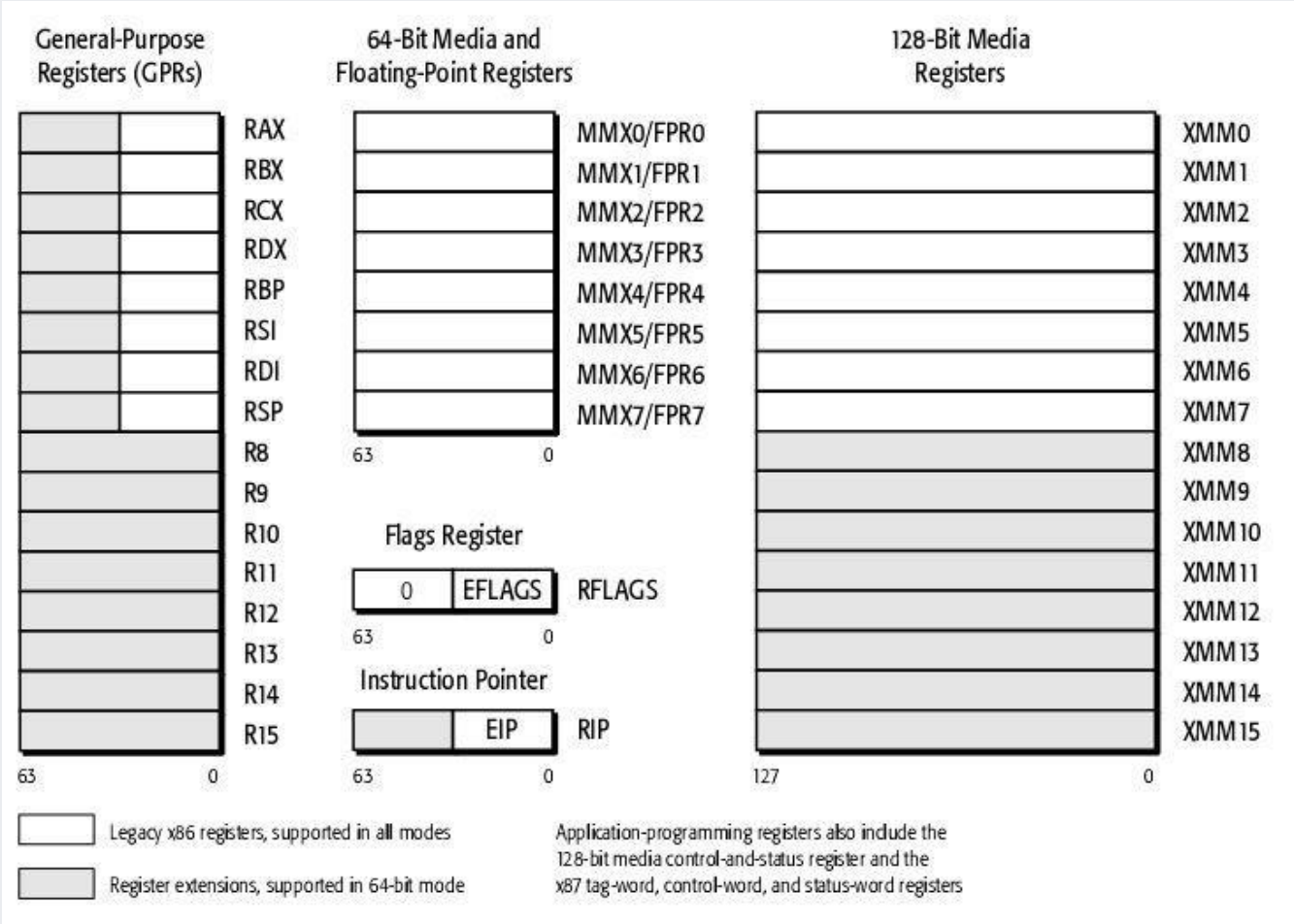
BYTE: 8 bit

WORD: 16 bit

DWORD: 32 bit

QWORD: 64 bit… EDX:EAX

x64


General-Purpose Registers (GPRs)
RAX
RBX
RCX
RDX
RBP
RSI
RDI
RSP
R8
R9
R10
R11
R12
R13
R14
R15
63
0
64-Bit Media and Floating-Point Registers
MMX0/FPR0
MMX1/FPR1
MMX2/FPR2
MMX3/FPR3
MMX4/FPR4
MMX5/FPR5
MMX6/FPR6
MMX7/FPR7
63
0
Flags Register
0
EFLAGS
RFLAGS
63
0
Instruction Pointer
EIP
RIP
63
0
128-Bit Media Registers
XMM0
XMM1
XMM2
XMM3
XMM4
XMM5
XMM6
XMM7
XMM8
XMM9
XMM10
XMM11
XMM12
XMM13
XMM14
XMM15
127
0
Legacy x86 registers, supported in all modes
Register extensions, supported in 64-bit mode
Application-programming registers also include the 128-bit media control-and-status register and the x87 tag-word, control-word, and status-word registers

Pamäť

- Virtuálna → Fyzická
- Real mode / Protected mode
- Segmentacia, flat memory model, paging

ASM Inštrukcie

Počet operandov: 0-3

- 0: **`nop`**
- 1: **`inc eax`**
- 2: **`mov eax, ebx`**
- 3: inštrukcie z inštrukčnej sady SSE, SSE2

Druhy operandov

- Register **`eax, ebx, al, ah...`**
- Pamäť **`[0x12345678], [eax], [eax+5]`**
- Konštanta (**`mov eax, 0x12345678`**)

Op codes

Syntax:

- Intel:

```
mov ecx, AABBCCDDh
mov ecx, [eax]
mov ecx, eax
```

- AT&T:

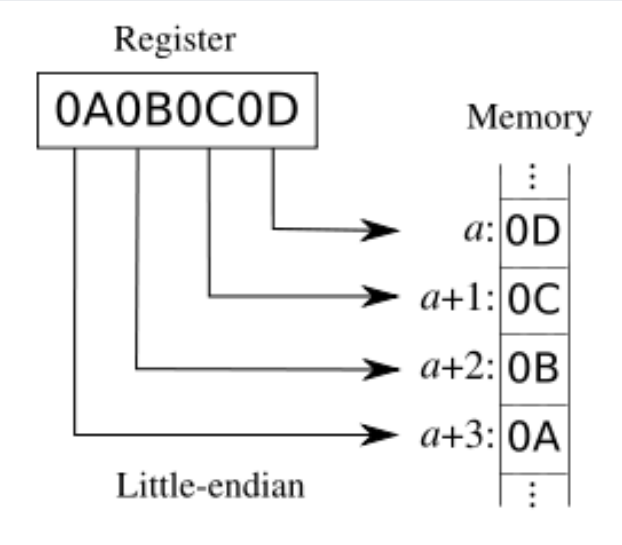
```
movl $0xAABBCCDD, %ecx
movl (%eax), %ecx
movl %eax, %ecx
```

ASM inštrukcie

- Pomocné direktívy
  - **`mov [0x00401000], 0`**
  - Koľko bytov máme zapísať na adresu `0x00401000`??? 1? 2? 4?

  - Na presné určenie slúžia pomocné direktívy:
    - **`BYTE PTR (1 byte): mov byte ptr [0x00401000], 0x00`**
    - **`WORD PTR (2 byte)`**
    - **`DWORD PTR (4 byte)`**

- x86 – Little-endian



Pamäťové inštrukcie

Slúžia na prenos údajov medzi:

- CPU –> CPU
- CPU –> RAM
- RAM –> CPU
- RAM -> RAM – nedá sa

MOV (Move)

PUSH / POP

PUSHAD / POPAD

LEA (Load effective address) - výpočet adries

```
lea eax, [ecx+eax*2+16]
lea eax, [eax+eax*2]
```

Aritmetické inštrukcie

ADD `add eax, dword ptr [0x401000]`

SUB `sub [eax], eax`

MUL / IMUL `imul ecx` (edx:eax = eax*ecx)

DIV / IDIV `div ebx`

INC – Inkrement `inc eax`

DEC – Dekrement `dec dword ptr [esi]`

NEG – Negácia `neg al`

Multiplier (explicit)	Multiplicand (implicit)	Result
8-bit reg/mem operand	AL	AX
16-bit reg/mem operand	AX	DX:AX
32-bit reg/mem operand	EAX	EDX:EAX

Divisor (explicit)	Dividend (implicit)	Quotient(result)	Remainder
8-bit reg/mem operand	AX	AL	AH
16-bit reg/mem operand	DX:AX	AX	DX
32-bit reg/mem operand	EDX:EAX	EAX	EDX

Logické, bitové inštrukcie

AND `and ax, 0x0F`

OR `or  eax, ebx`

XOR `xor eax, eax`

NOT `not eax`

SHL, **SHR** – posun o n bitov `shl eax, 5`

ROL, **ROL** – rotácia o n bitov `rol ebx, 4`

BT, **BS** – bit test, bit set `bt  eax, 0` **(C Flag)**

Často používané pri násobení a delení 2^n

`shl eax, 1` <=> vynásobenie `eax = eax*2`

Operácie so stringami

STOSB, STOSW, STOSD ...uloží AL,AX,EAX do [EDI]

LODSB, LODSW,LODSD... Načíta do AL,AX,EAX hodnotu z [ESI]

CMPSB, CMPSW, CMPSD ...Porovnávanie stringov. Porovnáva [esi] s [edi]

SCASB, SCASW, SCASD ...Hľadanie znaku - porovnávanie [EDI] s AL/AX/EAX

- možné kombinovať s **REP, REPZ, REPE, REPNE, REPNZ**
- smer kopírovania/porovnávania: DF (Direction Flag). cld (smer dopredu), std (smer dozadu)

```
push 8
pop ecx
mov esi, offset _Src
mov edi, offset _Dst
rep movsd
```

```
memcpy(&Dst, &Src, 32);
```

Operácie so stringami

SCASB, SCASW, SCASD ...Hľadanie znaku - porovnávanie [EDI] s AL/AX/EAX

```
mov al, 40h
mov edi, offset _String
repne scasb
```

Vetvenie programu, cykly

Žiadne FOR, WHILE, REPEAT!

JMP – nepodmienený skok

JE,JNE,JA,JNA... – podmienené skoky (Jcc)

LOOP

CMP Odpočíta operandy a podľa výsledku nastaví EFLAGS

TEST Vykoná AND a podľa výsledku nastaví EFLAGS

Podmienené skoky - Jcc

jb,jnae,jc	jump if below, jump if not above or equal, jump if carry	CF = 1
jae,jnb,jnc	jump if above or equal, jump if not below, jump if not carry	CF = 0
jbe,jna	jump if below or equal,jump if not above	CF = 1 or ZF = 1
ja,jnbe	jump if above,jump if not below or equal	CF = 0 and ZF = 0
je,jz	jump if equal,jump if zero	ZF = 1
jne,jnz	jump if not equal,jump if not zero	ZF = 0
jl,jnge	jump if less,jump if not greater or equal	SF ^{1} OF
jge,jnl	jump if greater or equal,jump if not less	SF = OF
jle,jng	jump if less or equal,jump if not greater	ZF = 1 or SF ^{1} OF
jg,jnle	jump if greater,jump if not less or equal	ZF = 0 and SF = OF
jp,jpe	jump if parity,jump if parity even	PF = 1
jnp,jpo	jump if not parity,jump if parity odd	PF = 0
js	jump if sign	SF = 1
jns	jump if not sign	SF = 0
jo	jump if overflow	OF = 1
jno	jump if not overflow	OF = 0

Čo robí tento kód?

```
loop_start:
   mov eax, [edi+4]
   mov eax, [eax+ebx*4]
   test eax, eax
   jz short loc_continue
   …
loc_continue:
   inc ebx
   cmp ebx, [edi]
   jl short loop_start
```

```
typedef struct _FOO
{
    DWORD size;           // +0x00
    DWORD array[…];  // +0x04
} FOO, *PFOO;

PFOO bar = …;
for (i= …; i< bar->size; i++) {
   if (bar->array[i] != 0) {
   …
   }
}
```

Funkcie – CALL, RET

```
int __cdecl addme(short a, short b)
{
     return a+b;
}
```

```
push ebp
mov ebp, esp
movsx eax, word ptr [ebp+8]
movsx ecx, word ptr [ebp+0Ch]
add eax, ecx
mov esp, ebp
pop ebp
retn
```

```
addme(x, y);
```

```
push eax
push ecx
call addme
add esp, 8
```

RET