Reverse Engineering

Alexander Sotirov
alex@sotirov.net

Part I

What is reverse engineering?

Computer Underground

- Cracking software copy protection
  - PC software and games
  - modding the Xbox and Playstation
- Exploit development
  - advanced exploitation
- Reversing undocumented operating system APIs
  - virus writers
  - spyware, keyloggers, malware, rootkits

Security Industry

- Virus and malware analysis
  - AV and Anti-Spyware companies
  - forensics
- Patch analysis, vulnerability analysis
  - IDS, IPS companies
- Binary code auditing
  - discovering new vulnerabilities
- Exploit development
  - penetration testing

Roadmap for today

The goal for today: turn a x86 binary executable back into C source code.

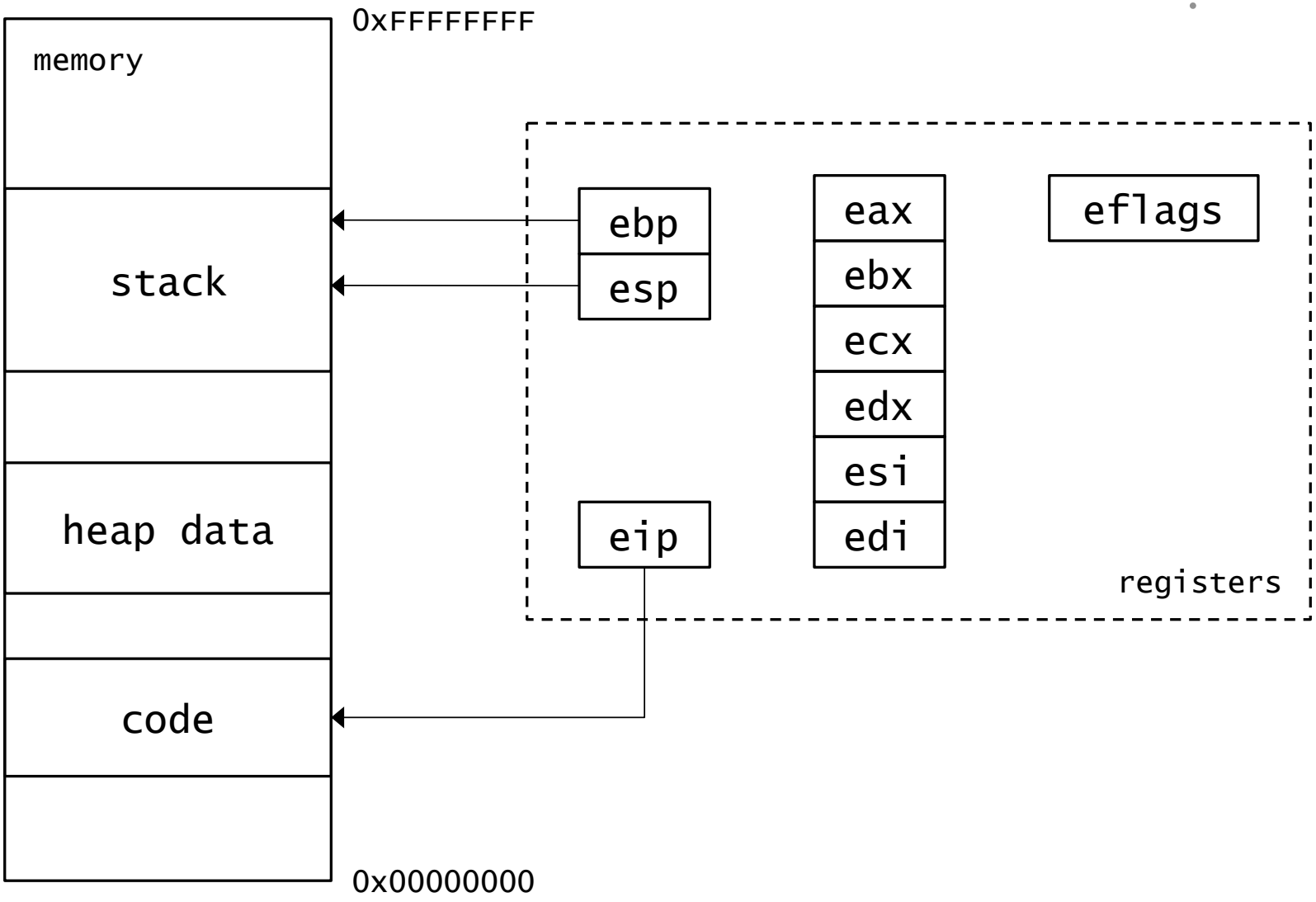
- x86 assembly in 10 minutes
- compiler design and code optimizations
- binary reverse engineering

We'll use the IDA Pro disassembler and a sample executable from a reversing contest.

Part II

x86 assembly in 10 minutes

CPU architecture



Arithmetic instructions

```
mov eax, 2          ; eax = 2
mov ebx, 3          ; ebx = 3

add eax, ebx        ; eax = eax + ebx
sub ebx, 2          ; ebx = ebx – 2
```

Accessing memory

```
mov eax, [1234]        ; eax = *(int*)1234

mov ebx, 1234          ; ebx = 1234
mov eax, [ebx]         ; eax = *ebx

mov [ebx], eax         ; *ebx = eax
```

Conditional branches

```
cmp eax, 2      ; compare eax with 2

je label1       ; if (eax == 2) goto label1
ja label2       ; if (eax > 2) goto label2
jb label3       ; if (eax < 2) goto label3
jbe label4      ; if (eax <= 2) goto label4
jne label5      ; if (eax != 2) goto label5

jmp label6      ; unconditional goto label6
```

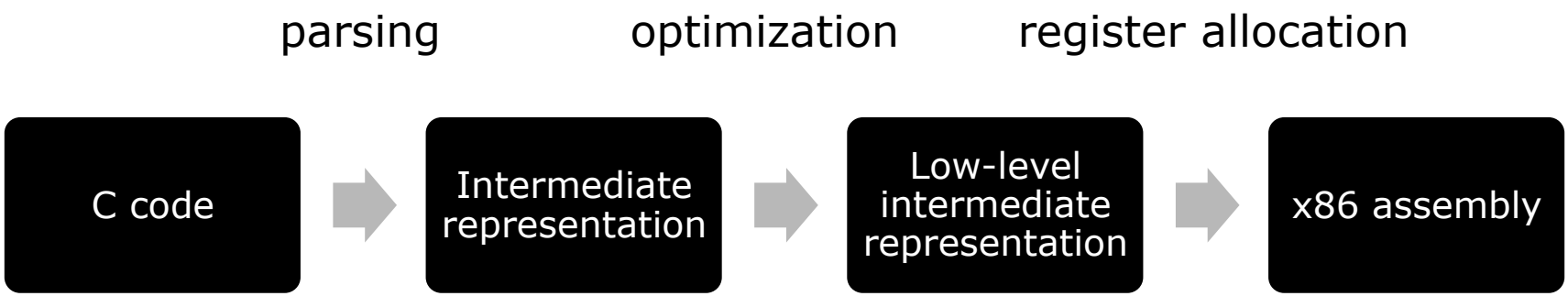
Function calls

```
call func          ; store return address on
                   ; the stack and jump to func

func:
        push esi   ; save esi
        …
        pop esi    ; restore esi
        ret        ; read return address from
                   ; the stack and jump to it
```

Part III

Compiler design and code optimizations

Modern compiler architecture

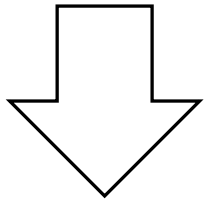


High-level Optimizations

- Inlining
- Loop unrolling
- Loop-invariant code motion
- Common subexpression elimination
- Constant folding and propagation
- Dead code elimintation

Inlining

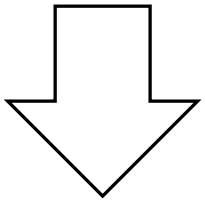
```
int foo(int a, int b) {
        return a + b;
}

c = foo(a, b+1);
```



```
c = a + b + 1;
```

Loop unrolling

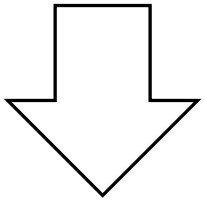
```
for (i = 0; i < 2; i++) {
        a[i] = 0;
}
```



```
a[0] = 0;
a[1] = 0;
```

Loop-invariant code motion

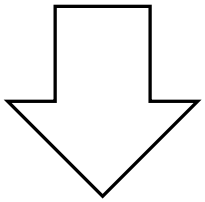
```
for (i = 0; i < 2; i++) {
        a[i] = p + q;
}
```



```
temp = p + q;
for (i = 0; i < 2; i++) {
        a[i] = temp;
}
```

Common subexpression elimination

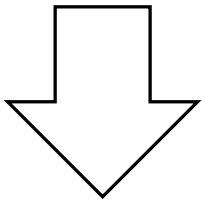
```
a = b + (z + 1)
p = q + (z + 1)
```



```
temp = z + 1
a = b + z
p = q + z
```

Constant folding and propagation

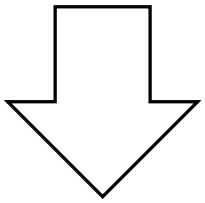
```
a = 3 + 5
b = a + 1
func(b)
```



```
func(9)
```

Dead code elimination

```
a = 1
if (a < 0) {
        printf(“ERROR!”)
}
```



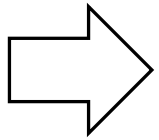
```
a = 1
```

Low-level optimizations

- Strength reduction
- Code block reordering
- Register allocation
- Instruction scheduling

Strength reduction

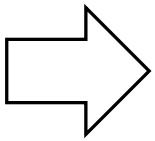
```
y = x * 2
y = x * 15
```



```
y = x + x
y = (x << 4) - x
```

Code block reordering

```
if (a < 10) goto l1
printf("ERROR")
goto label2
l1:
printf("OK")
l2:
return;
```



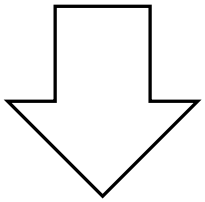
```
if (a > 10) goto l1
printf("OK")
l2:
return
l1:
printf("ERROR")
goto l2
```

Register allocation

- Memory access is slower than registers
- Try to fit as many local variables as possible in registers
- The mapping of local variables to stack locations and registers is not constant

Instruction scheduling

```
mov eax, [esi]
add eax, 1
mov ebx, [edi]
add ebx, 1
```



```
mov eax, [esi]
mov ebx, [edi]
add eax, 1
add ebx, 1
```

Part IV

Binary reverse engineering

Overview

We need to understand:

- How the compiler turns C code into assembly code
- Low-level OS structures and executable file format

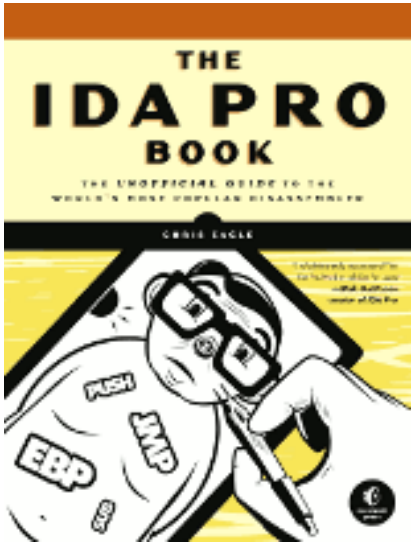
We need to think like the compiler, but in reverse!

Tools

IDA Pro disassembler from hex-rays.com

Limited demo and freeware versions are available.

Almost no documentation!



Diving into IDA Pro