



UNIVERSITÀ DEGLI STUDI DI MILANO
FACOLTÀ DI SCIENZE E TECNOLOGIE
DIPARTIMENTO DI INFORMATICA

Reverse engineering: *disassembly*

Srdjan Matic <srdjan@security.di.unimi.it>
Aristide Fattori <joystick@security.di.unimi.it>

A.A. 2013–2014

Cos'è?

stream of bytes

disassembly

sequence of
assembly
instructions

Esempio

```
$ echo -ne "\x89\x04\x24\xc3" | ndisasm -u -  
00000000 890424      mov [esp],eax  
00000003 C3           ret
```

Indecidibilità della separazione tra *codice* e *dati*

Intel x86: 248/256 byte rappresentano l'inizio di un'istruzione valida

Esempio

... 0x81 0xc3 0x90 0x90 0x90 0x90 ...

- se consideriamo il byte 0x81 come codice:

```
$ echo -ne \x81\xc3\x90\x90\x90\x90 | ndisasm -b32 -  
00000000 81C390909090          add ebx,0x90909090
```

- se consideriamo il byte 0x81 come data e il byte 0xc3 come codice:

```
$ echo -ne \xc3\x90\x90\x90\x90 | ndisasm -b32 -  
00000000  C3                      ret  
00000001  90                      nop  
00000002  90                      nop  
00000003  90                      nop  
00000004  90                      nop
```

Disassembly

Problema: esplosione di complessità

```
1  int main(int argc, char **argv)
2  {
3      int num;
4
5      scanf("%d", &num);
6
7      if(num == 1234)
8          printf("ok\n");
9      else
10         printf("no\n");
11
12     return 0;
13 }
```

Disassembly

Problema: esplosione di complessità

```
push %ebp                push %ebx                leave ret                lea   -1(%eax),%esi    add   $0xc,%esp
mov %esp,%ebp           sub $0x4,%esp          ret nop                cmp   $-1,%esi        pop   %ebx
sub $0x8,%esp          call 8048310            push %ebp             je    8048419          pop   %esi
call 8048304           pop %ebx               mov %esp,%ebp        lea  0x0(%esi),%esi   pop   %edi
call 8048360           add $0x12c8,%ebx      sub $0x18,%esp      call *(%edi,%esi,4)  pop   %ebp
call 8048490           mov -4(%ebx),%edx     and $-16,%esp       dec %esi              ret
leave ret              je 8048326             mov $0x0,%eax       cmp  $-1,%esi        mov   (%esp),%ebx
ret                    push 80482d0           add $0xf,%eax       jne  8048410          ret
pushl 0x80495dc        call %eax              add $0xf,%eax       lea  0x0(%esi),%esi   nop
jmp *0x80495e0         pop %eax               add $0xf,%eax       call 80484c4          ...
add *%al,(%eax)       pop %ebx               shr $0x4,%eax       add $0xc,%esp        push  %ebp
jmp *0x80495e4        leave %ebx             shl $0x4,%eax       pop %ebx              mov   %esp,%ebp
push $0x0              ret                    sub %eax,%esp       pop %esi              push  %ebx
jmp 8048290            nop                    lea -4(%ebp),%eax   pop %esi              sub   $0x4,%esp
jmp *0x80495e8        ..                    mov %eax,0x4(%esp) pop %edi              mov   0x80494f8,%eax
push $0x8              push %ebp              movl $0x80484e8,(%esp)ret  pop %ebp              cmp   $-1,%eax
jmp 8048290           mov %esp,%ebp         call 80482b0         je    80484bd
jmp *0x80495ec        sub $0x8,%esp         mov -4(%ebp),%eax   push %ebp             mov   $0x80494f8,%ebx
push $0x10            cmpb $0x0,0x8049600   mov %eax,%edx       push %edi             lea  0x0(%edi),%edi
jmp 8048290           je 804834b            jmp 804835d         mov %esp,%ebp        lea  0x0(%edi),%edi
jmp *0x80495f0        jmp 804835d           cmp $0x4,%eax       push %edi             call  *%eax
push $0x18            mov %eax,0x80495fc   call %edx            push %esi             mov   -4(%ebx),%eax
jmp 8048290           call %edx             mov 0x80495fc,%eax  push %ebx             sub   $0x4,%ebx
xor %ebp,%ebp         mov (%eax),%edx      mov %edx,%edx       cmp  $-1,%eax        jne  80484b0
pop %esi              jne 8048341           mov %eax,%edx       call 80482a0         sub   %eax,%esp
mov %esp,%ecx         and $-16,%esp        mov 80483d7         jmp 8048483          add  $0x119d,%ebx
and $-16,%esp        push %eax             movl $0x80484ee,(%esp)push %eax              pop   %eax
push %eax             leave %eax            ret nop              lea  -224(%ebx),%eax
push %esp            ret                    ..                    lea  -224(%ebx),%edx
push %edx            ..                    push %ebp           sar  $0x2,%eax
push $0x80483e0       nop                    mov %esp,%ebp       mov  %eax,-16(%ebp)  push  %ebp
push $0x8048430      push %ebp            push %edi           je    804847b         mov   %esp,%ebp
push %ecx            mov %esp,%ebp        push %esi           xor  %edi,%edi       push  %ebx
push %esi            sub $0x8,%esp        push %ebx           mov  %edx,%esi       sub   $0x4,%esp
push $0x8048384      mov 0x8049508,%eax   call 8048483        lea  0x0(%esi),%esi  call 80484d0
call 80482c0         test %eax,%eax       add $0x11ed,%ebx   lea  0x0(%edi),%edi  pop   %ebx
hit nop              je 8048381           sub $0xc,%esp      lea  %edi             add  $0x1108,%ebx
nop                  mov $0x0,%eax        lea -224(%ebx),%eax inc %edi              call 8048330
push %ebp            test %eax,%eax       lea -224(%ebx),%edi call *(%esi)          pop   %ecx
mov %esp,%ebp        je 8048381           sub %edi,%eax      cmp  %edi,-16(%ebp)  pop   %ebx
mov %esp,%ebp        movl $0x8049508,(%esp)sar $0x2,%eax          jne  8048470         leave ret
```

Instruction set

- istruzioni a lunghezza fissa
 - ogni istruzione inizia ad un indirizzo multiplo della sua lunghezza
- istruzioni a lunghezza variabile
 - disassembly difficoltoso
 - il processo di disassembly può “desincronizzarsi”

Instruction set

- istruzioni a lunghezza fissa
 - ogni istruzione inizia ad un indirizzo multiplo della sua lunghezza
- istruzioni a lunghezza variabile
 - disassembly difficoltoso
 - il processo di disassembly può “desincronizzarsi”

Problemi

- interposizione di dati e istruzioni nello stesso address space (come distinguerli?)
- istruzioni di lunghezza variabile
- trasferimenti di controllo indiretti (function pointer, dynamic linking, jump table, ...)

Problemi²

Nella fase di compilazione si perdono:

- nomi delle variabili
- informazioni sui tipi
- concetto di “blocco”
- macro
- commenti

Problemi³

- riconoscere i limiti delle funzioni
- distinguere i parametri delle funzioni

Disassembly

Sintassi AT&T vs Intel

```
$ objdump -M att -d /bin/ls
...
push    %ebp
xor     %ecx,%ecx
mov     %esp,%ebp
sub     $0x8,%esp
mov     %ebx,(%esp)
mov     0x8(%ebp),%ebx
mov     %esi,0x4(%esp)
mov     0xc(%ebp),%esi
mov     (%ebx),%edx
mov     0x4(%ebx),%eax
xor     0x4(%esi),%eax
xor     (%esi),%edx
or      %edx,%eax
je      8049c60 <exit@plt+0x13c>
mov     %ecx,%eax
mov     (%esp),%ebx
mov     0x4(%esp),%esi
mov     %ebp,%esp
pop     %ebp
ret
```

```
$ objdump -M intel -d /bin/ls
...
push    ebp
xor     ecx,ecx
mov     ebp,esp
sub     esp,0x8
mov     DWORD PTR [esp],ebx
mov     ebx,DWORD PTR [ebp+0x8]
mov     DWORD PTR [esp+0x4],esi
mov     esi,DWORD PTR [ebp+0xc]
mov     edx,DWORD PTR [ebx]
mov     eax,DWORD PTR [ebx+0x4]
xor     eax,DWORD PTR [esi+0x4]
xor     edx,DWORD PTR [esi]
or      eax,edx
je      8049c60 <exit@plt+0x13c>
mov     eax,ecx
mov     ebx,DWORD PTR [esp]
mov     esi,DWORD PTR [esp+0x4]
mov     esp,ebp
pop     ebp
ret
```

08048350	B8 66 83 04 08	mov \$0x8048366,%eax
08048355	FF D0	call *%eax
08048357	C3	ret
08048358	48 65 6C 6C 6F 20 57	(data)
0804835F	6F 72 6C 64 21 0A 0D	(data)
08048366	BA 0E 00 00 00	mov \$0xe,%edx
0804836B	B9 58 83 04 08	mov \$0x8048358,%ecx
08048370	BB 00 00 00 00	mov \$0x0,%ebx
08048375	B8 04 00 00 00	mov \$0x4,%eax
0804837A	CD 80	int \$0x80
0804837C	B8 00 00 00 00	mov \$0x0,%eax
08048381	C3	ret

Tecniche utilizzate

- linear sweep disassembly
- recursive traversal disassembly

Linear sweep disassembly

(e.g., objdump)

- inizia al primo byte del segmento testo
- procedi decodificando un'istruzione dopo l'altra
`$ objdump -D file`
- *assunzione*: istruzioni memorizzate in locazioni adiacenti

08048350	B8 66 83 04 08	mov \$0x8048366,%eax
08048355	FF D0	call *%eax
08048357	C3	ret
08048358	48	dec %eax
08048359	65	gs
0804835A	6C	insb (%dx),%es:(%edi)
0804835B	6C	insb (%dx),%es:(%edi)
0804835C	6F	outs1 %ds:(%esi),(%dx)
0804835D	20 57 6F	and %dl,0x6f(%edi)
08048360	72 6C	jb 0x80483ce
08048362	64 21 0A	and %ecx,%fs:(%edx)
08048365	0D BA 0E 00 00	or \$0xeba,%eax
0804836A	00 B9 58 83 04 08	add %bh,0x8(%ecx)
08048370	BB 00 00 00 00	mov \$0x0,%ebx
08048375	B8 04 00 00 00	mov \$0x4,%eax
0804837A	CD 80	int \$0x80
0804837C	B8 00 00 00 00	mov \$0x0,%eax
08048381	C3	ret

Linear sweep disassembly

(e.g., objdump)

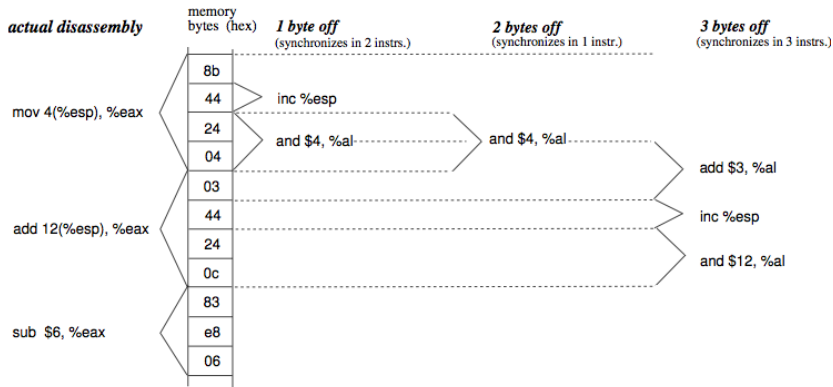
- inizia al primo byte del segmento testo
- procedi decodificando un'istruzione dopo l'altra
`$ objdump -D file`
- *assunzione*: istruzioni memorizzate in locazioni adiacenti

08048350	B8 66 83 04 08	mov \$0x8048366,%eax
08048355	FF D0	call *%eax
08048357	C3	ret
08048358	48	dec %eax
08048359	65	gs
0804835A	6C	insb (%dx),%es
0804835B	6C	insb (%dx),%es:(%edi)
0804835C	6F	outs1 %ds:(%esi),(%dx)
0804835D	20 57 6F	and %dl,0x6f(%edi)
08048360	72 6C	jb 0x80483ce
08048362	64 21 0A	and %ecx,%fs:(%edx)
08048365	0D BA 0E 00 00	or \$0xeba,%eax
0804836A	00 B9 58 83 04 08	add %bh,0x8(%ecx)
08048370	BB 00 00 00 00	mov \$0x0,%ebx
08048375	B8 04 00 00 00	mov \$0x4,%eax
0804837A	CD 80	int \$0x80
0804837C	B8 00 00 00 00	mov \$0x0,%eax
08048381	C3	ret

IA-32 disassembly
è self-repairing

Linear sweep disassembly

IA-32: self-repairing disassembly



Recursive traversal disassembly

(e.g., IDA Pro)

- 1 inizia dall'entry point
- 2 quando incontri un'istruzione di branch:
 - determina possibili successori
 - riprendi il disassembly a questi indirizzi

08048350	B8 66 83 04 08	mov \$0x8048366,%eax
08048355	FF D0	call <i>*%eax</i>
08048357	C3	ret

Cos'è?

- struttura fondamentale in program analysis
- grafo orientato in cui:
 - **nodi**: *computazioni* (basic block)
 - **archi**: possibili *flussi di esecuzione*

Basic block

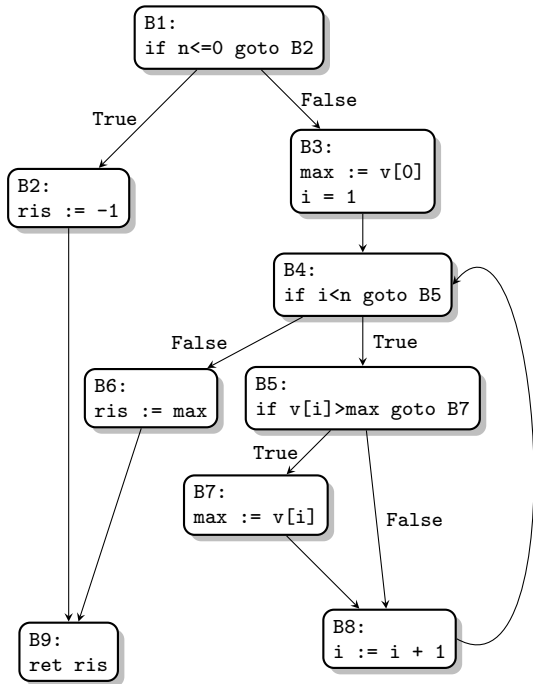
sequenza *non interrompibile* di istruzioni

- singolo entry point
- singolo exit point

```

1  int find_max(int *v, int n)
2  {
3      unsigned int max, i;
4      int ris;
5
6      if(n <= 0)
7          ris = -1;
8      else {
9          max = v[0];
10         i = 1;
11         while(i < n) {
12             if(v[i] > max)
13                 max = v[i];
14             i = i + 1;
15         }
16         ris = max;
17     }
18     return ris;
19 }

```



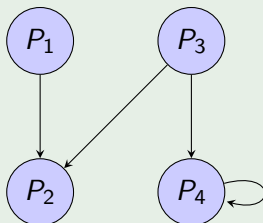
Cos'è?

- grafo orientato $C = (R, F)$
 - i nodi rappresentano **procedure**
 - gli archi sono relazioni *chiamante-chiamato*
- se $(r_i, r_j) \in R$, nel corso di r_i si ha almeno una chiamata a r_j

Cos'è?

- grafo orientato $C = (R, F)$
 - i nodi rappresentano **procedure**
 - gli archi sono relazioni *chiamante-chiamato*
- se $(r_i, r_j) \in R$, nel corso di r_i si ha almeno una chiamata a r_j

Esempio





Hands on!



```
$ wget http://security.di.unimi.it/sicurezza1314/demo{,.c}
```

Regola #1 di IDA

Everything you see and you do, you are working on the database. Changes are NOT automatically reflected to the binary.

Regola #2 di IDA

There are plenty of things you can do. But there is no undo.

- `sub_xxxx`: a function at address `xxxx`
- `loc_xxxx`: an instruction at address `xxxx`
- `(byte|word|dword)_xxxx`: data at address `xxxx`
- `var_xx`: *local* variable at `EBP-xx`
- `arg_n`: *argument* at `EBP+8+n`

https://www.hex-rays.com/products/ida/support/frefiles/IDA_Pro_Shortcuts.pdf

<http://security.di.unimi.it/sicurezza1314/homeworks/homework3.tar.gz>

Due Oct 23 @ 23:59:59

Utilizzando IDA (!hexrays), *reversare* gli ELF scaricati. Ogni programma contiene una chiave che può essere individuata staticamente e confermata eseguendo il programma e fornendo in input la chiave.

Scrivere una *breve* relazione (pdf/txt), contenente, \forall programma:

- la chiave individuata
- come l'avete individuata/estratta
- descrizione alto livello del programma

- G. Vigna
Malware Detection, chapter Static Disassembly and Code Analysis
- C. Eagle
The IDA Pro Book: The Unofficial Guide to the World's Most Popular Disassembler
- Intel 64 and IA-32 Architectures Software Developers Manual Volume 2 (2A, 2B & 2C): Instruction Set Reference, A-Z
- <http://ref.x86asm.net/coder32.html>