Malware Analysis

Chapter 6: Recognizing C Constructs in Assembly

王志
zwang@nankai.edu.cn

updated on Oct. 17th 2021

College of Cyber Science
Nankai University
2021/2022

Agenda

- Global vs Local Variables
- Arithmetic Operations
- If statement
- For and While Loops
- Function Call Convention
- Switch Statement
- Arrays, Structures, Linked List

Reverse Engineering

- The number of instructions of a program can be thousands or even millions
- Evaluating each instruction is tedious
- Successful reverse engineers
  - analyze instructions as groups
  - obtain a high-level picture of code functionality

Code Structure

- How to group binary instructions?
- A code abstraction level
- Define a functional property
- Not the details of its implementation
  - loops
  - if statements
  - liked lists
  - switch statement

C Code Constructs

- Goal as a malware analyst
  - to go from disassembly to high-level constructs
  - The most common constructs, such as loops and conditional statements, are compiled.
- High-level picture of code functionality

Compiler

- Compiler versions and settings can impact construct appears in disassembly.
- Most malware is written in C/C++
  - Close relationship to assembly
  - Anti-Virus Engine is written in C and Assembly
- C Compilers?
  - Windows
  - Linux

Notice

- understand the overall functionality of a program
- not analyze every single instruction

Global vs. Local Variables

- Global variables
  - can be accessed and used by any function in a program
- Local variables
  - can be accessed only by the function in which they are defined

Assembly View

- In C, the declaration of global and local variables are similar, but completely different in assembly.

Global vs. Local

```
int x = 1;
int y = 2;

void main() {
  x = x+y;
  printf("Total = %d\n", x);
}
```

```
void main() {
  int x = 1;
  int y = 2;
  x = x+y;
  printf("Total = %d\n", x);
}
```

Global vs. Local

```
00401003        mov     eax, dword_40CF60
00401008        add     eax, dword_40C000
0040100E        mov     dword_40CF60, eax ❶
00401013        mov     ecx, dword_40CF60
00401019        push    ecx
0040101A        push    offset aTotalD  ;"total = %d\n"
0040101F        call    printf
```

```
00401006        mov     dword ptr [ebp-4], 0
0040100D        mov     dword ptr [ebp-8], 1
00401014        mov     eax, [ebp-4]
00401017        add     eax, [ebp-8]
0040101A        mov     [ebp-4], eax
0040101D        mov     ecx, [ebp-4]
00401020        push    ecx
00401021        push    offset aTotalD  ; "total = %d\n"
00401026        call    printf
```

Arithmetic Operations

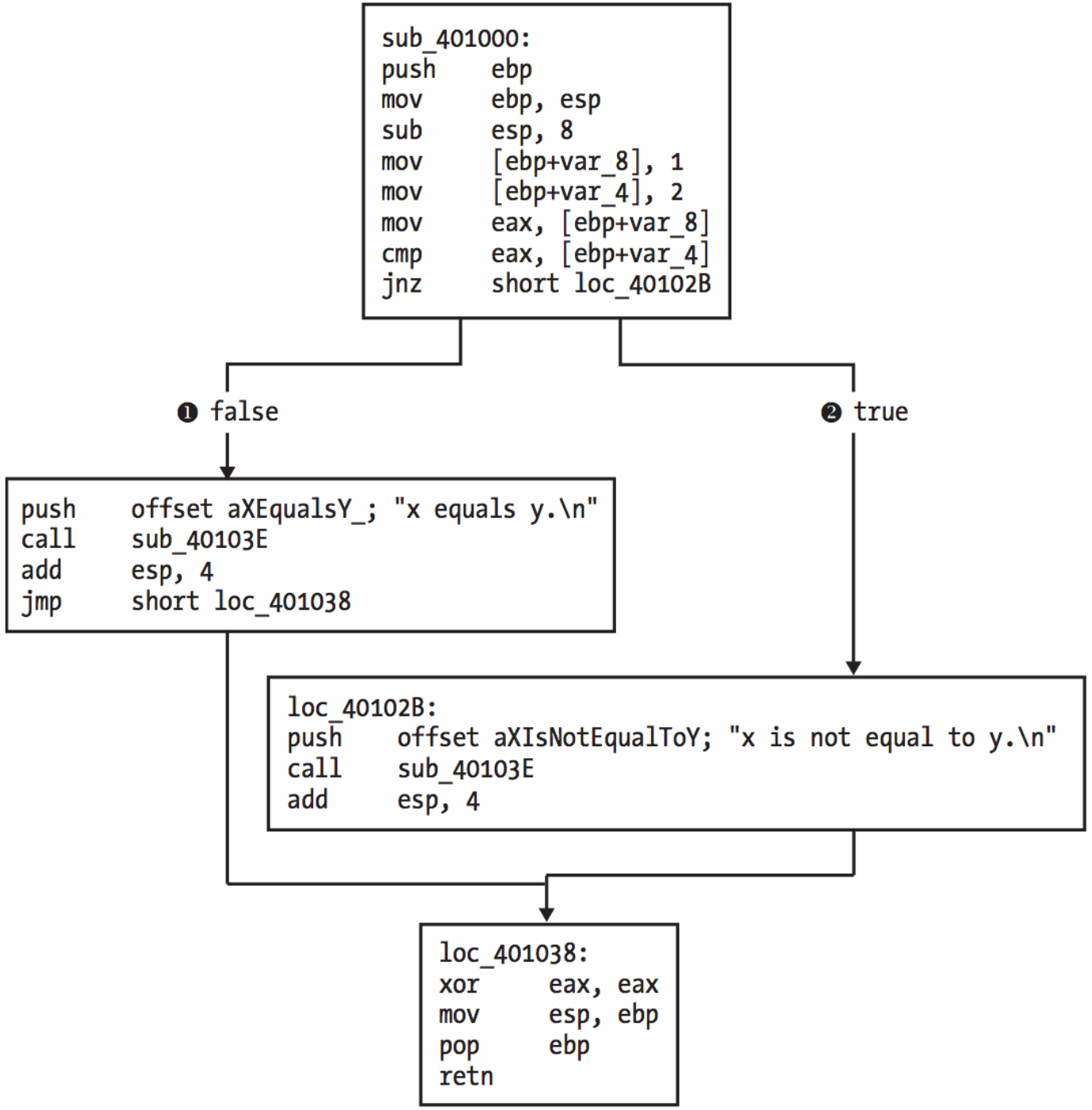
- The disassembly of math c

```
int a = 0;
int b = 1;
a = a + 11;
a = a - b;
a--;
b++;
b = a % 3;
```

```
00401006        mov     [ebp+var_4], 0
0040100D        mov     [ebp+var_8], 1
00401014        mov     eax, [ebp+var_4] ❶
00401017        add     eax, 0Bh
0040101A        mov     [ebp+var_4], eax
0040101D        mov     ecx, [ebp+var_4]
00401020        sub     ecx, [ebp+var_8] ❷
00401023        mov     [ebp+var_4], ecx
00401026        mov     edx, [ebp+var_4]
00401029        sub     edx, 1 ❸
0040102C        mov     [ebp+var_4], edx
0040102F        mov     eax, [ebp+var_8]
00401032        add     eax, 1 ❹
00401035        mov     [ebp+var_8], eax
00401038        mov     eax, [ebp+var_4]
0040103B        cdq
0040103C        mov     ecx, 3
00401041        idiv    ecx
00401043        mov     [ebp+var_8], edx ❺
```

IF Statements

- If statements change program execution based on certain conditions.
- We will learn
  - Basic if statement
  - Nested if statement



004010
004010
004010
004010
004010
004010
004010
004010
004010
004010
004010
004010

y.\n"

```
int x = (
int y = 
int z = 

if(x == 
    if(

    }els

    }
}else{
    if(z

    }els

    }
}
```

```
00401006        mov     [ebp+var_8], 0
0040100D        mov     [ebp+var_4], 1
00401014        mov     [ebp+var_C], 2
0040101B        mov     eax, [ebp+var_8]
0040101E        cmp     eax, [ebp+var_4]
00401021        jnz     short loc_401047 ❶
00401023        cmp     [ebp+var_C], 0
00401027        jnz     short loc_401038 ❷
00401029        push    offset aZIsZeroAndXY_ ; "z is zero and x = y.\n"
0040102E        call    printf
00401033        add     esp, 4
00401036        jmp     short loc_401045
00401038 loc_401038:
00401038        push    offset aZIsNonZeroAndX ; "z is non-zero and x = y.\n"
0040103D        call    printf
00401042        add     esp, 4
00401045 loc_401045:
00401045        jmp     short loc_401069
00401047 loc_401047:
00401047        cmp     [ebp+var_C], 0
0040104B        jnz     short loc_40105C ❸
0040104D        push    offset aZZeroAndXY_ ; "z zero and x != y.\n"
00401052        call    printf
00401057        add     esp, 4
0040105A        jmp     short loc_401069
0040105C loc_40105C:
0040105C        push    offset aZNonZeroAndXY_ ; "z non-zero and x != y.\n"
00401061        call    printf00401061
```

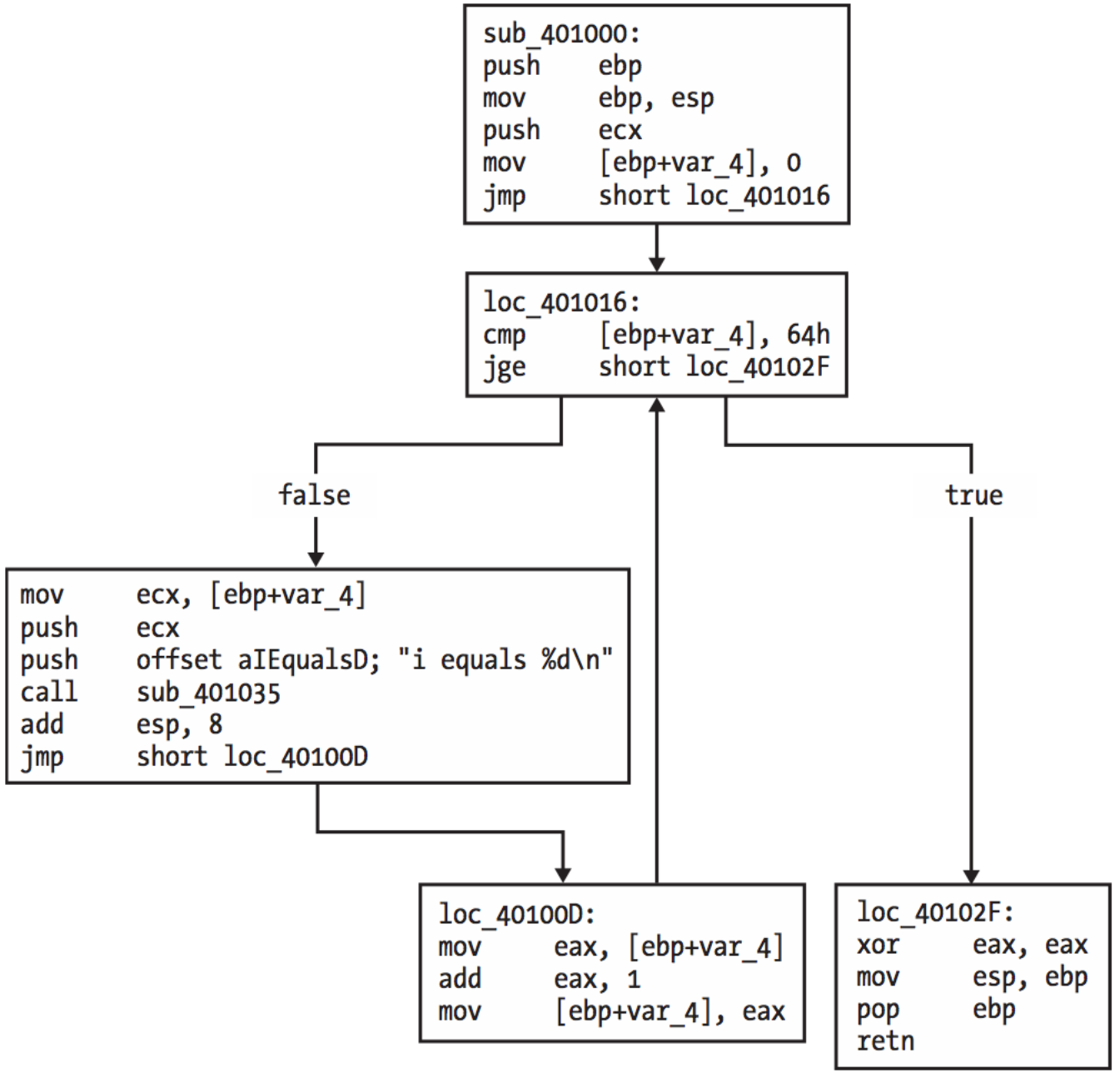
Loops

- For loops, always have four components:
  - initialization
  - comparison
  - execution blocks
  - increment or decrement

```
int i;

for(i=0; i<100; i++)
{
   printf("i equals %d\n", i);
}
```

- Locate the f



While Loops

```
int status=0;
int result = 0;

while(status == 0){
     result = performAction();
     status = checkResult(result);
}
```

```
00401036          mov     [ebp+var_4], 0
0040103D          mov     [ebp+var_8], 0
00401044 loc_401044:
00401044          cmp     [ebp+var_4], 0
00401048          jnz     short loc_401063 ❶
0040104A          call    performAction
0040104F          mov     [ebp+var_8], eax
00401052          mov     eax, [ebp+var_8]
00401055          push    eax
00401056          call    checkResult
0040105B          add     esp, 4
0040105E          mov     [ebp+var_4], eax
00401061          jmp     short loc_401044 ❷
```

Function Call Convention

- Function calls can appear differently in assembly code
- Calling conventions govern the way the function call occurs
- Caller
- Callee
- Who is responsible for cleaning up the stack when the function is complete?

Calling Convensions

- The calling convention used depends on the compiler
- Three most common calling convensions:
  - __cdecl
  - __stdcall
  - __fastcall

__cdecl

- __cdecl is short for "C Declaration"
- the default calling convention for C and C++ programs

__cdecl

Element	Implementation
Argument-passing order	Right to left.
Stack-maintenance responsibility	Caller pops the arguments from the stack.
Case-translation convention	No case translation performed.

__stdcall

- The **__stdcall** calling convention is used to call Win32 API functions.

__stdcall

Argument-passing order	Right to left.
Stack-maintenance responsibility	Callee pops its own arguments from the stack.
Case-translation convention	None

__fastcall

Element	Implementation
Argument-passing order	The first two DWORD or smaller arguments that are found in the argument list from left to right are passed in ECX and EDX registers; all other arguments are passed on the stack from right to left.
Stack-maintenance responsibility	Callee pops the arguments from the stack.

单选题 10分

设置

“push eax
push ebx
call fun
add esp 8”

Which calling convention the above code is using?

A __stdcall

B __cdecl

C __fastcall

提交

Push vs. Move

- compilers choose to use different instructions to perform the same operation

Push vs. Move

Visual Studio version			GCC version		
00401746	mov	[ebp+var_4], 1	00401085	mov	[ebp+var_4], 1
0040174D	mov	[ebp+var_8], 2	0040108C	mov	[ebp+var_8], 2
00401754	mov	eax, [ebp+var_8]	00401093	mov	eax, [ebp+var_8]
00401757	push	eax	00401096	mov	[esp+4], eax
00401758	mov	ecx, [ebp+var_4]	0040109A	mov	eax, [ebp+var_4]
0040175B	push	ecx	0040109D	mov	[esp], eax
0040175C	call	adder	004010A0	call	adder
00401761	**add**	**esp, 8**			
00401764	push	eax	004010A5	mov	[esp+4], eax
00401765	push	offset TheFunctionRet	004010A9	mov	[esp], offset TheFunctionRet
0040176A	call	ds:printf	004010B0	call	printf

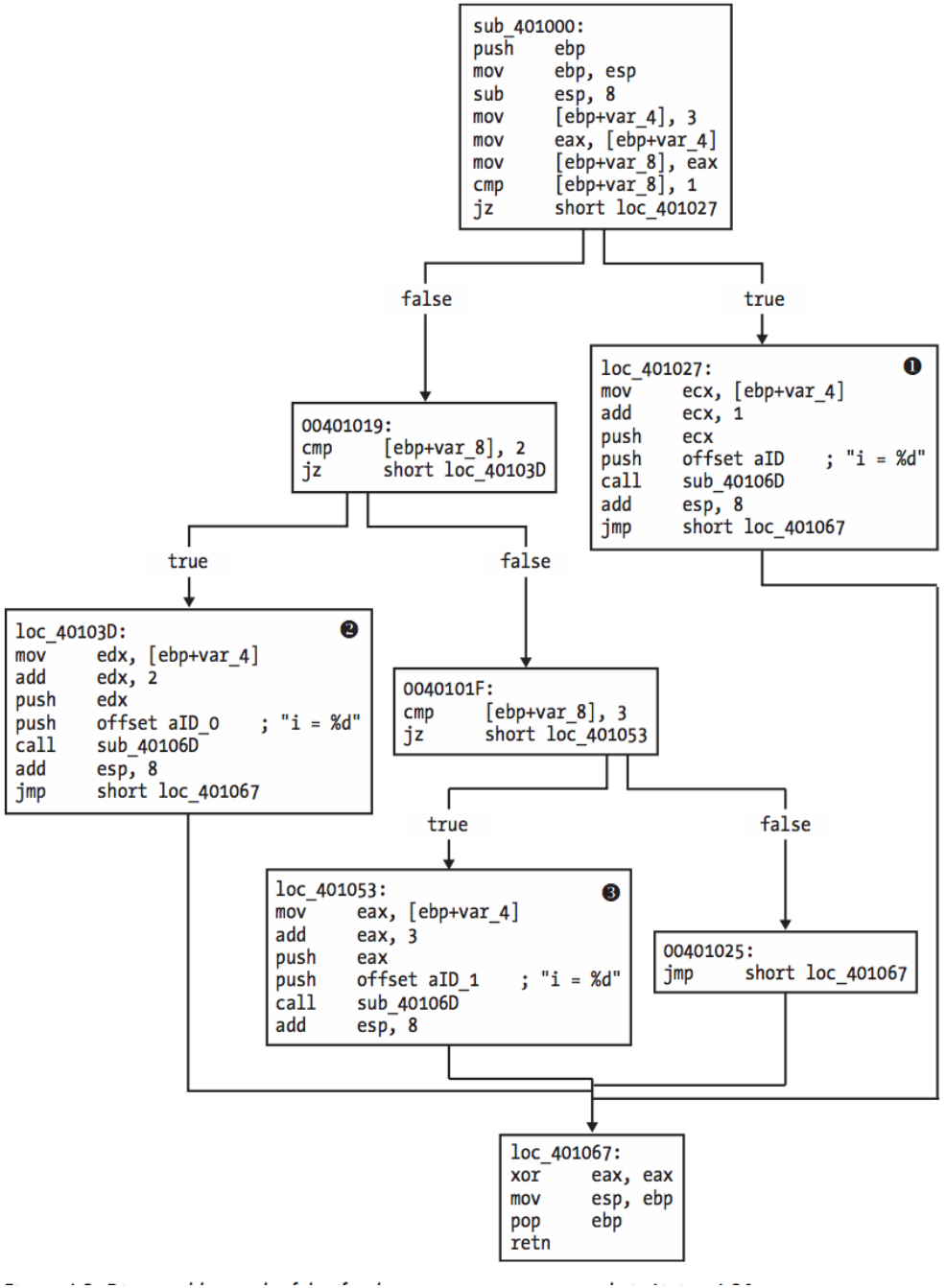
Switch Statement

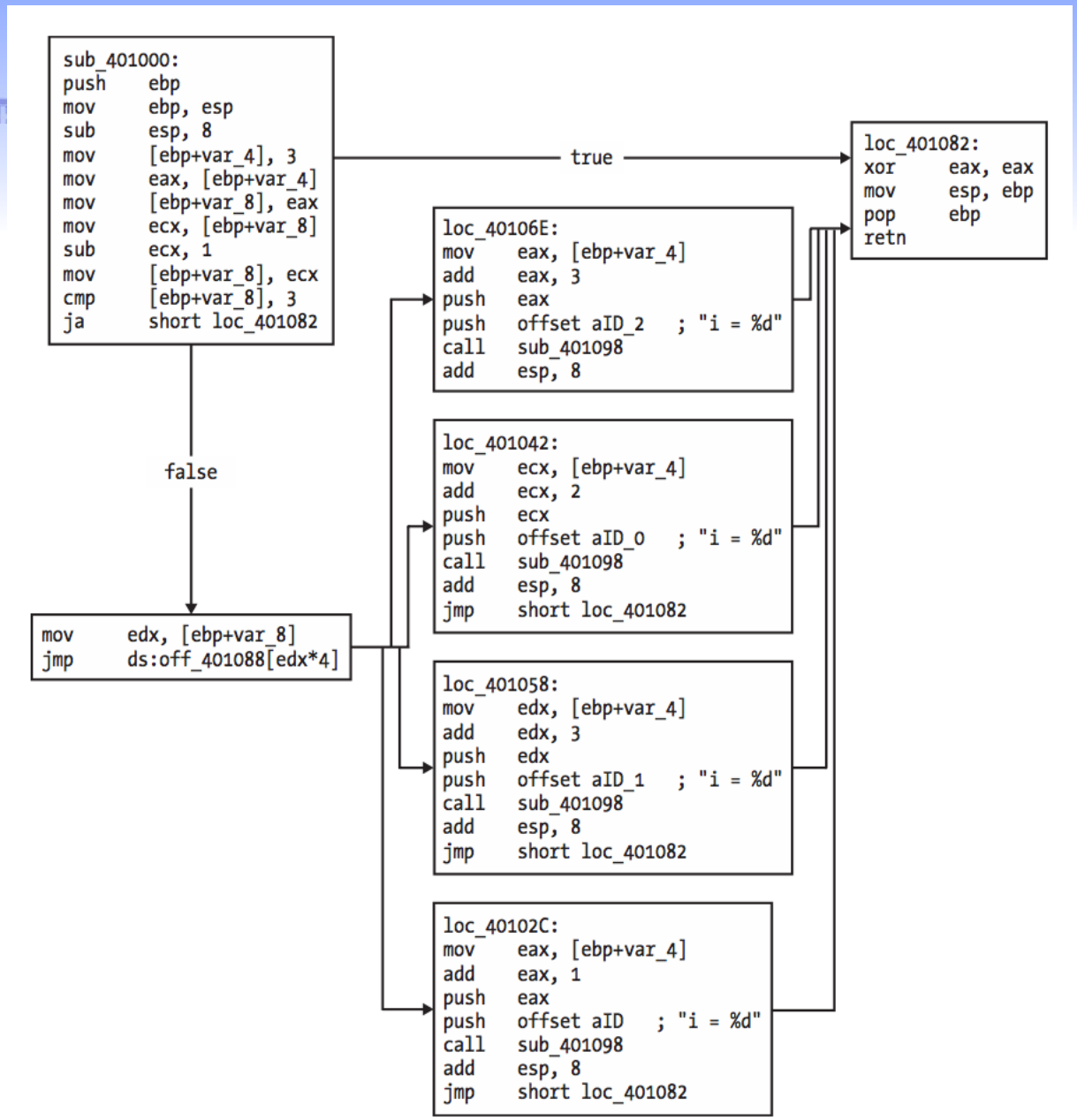
- Switch statements are used by programmers (and malware authors) to make a decision based on a character or integer

```
switch(i)
{
   case 1:
      printf("i = %d", i+1);
      break;
   case 2:
      printf("i = %d", i+2);
      break;
   case 3:
      printf("i = %d", i+3);
      break;
   default:
      break;
}
```

Switch Statements

- Switch statements are compiled in two common ways
  - using the if style
  - using jump tables


sub_401000:
push ebp
mov ebp, esp
sub esp, 8
mov [ebp+var_4], 3
mov eax, [ebp+var_4]
mov [ebp+var_8], eax
cmp [ebp+var_8], 1
jz short loc_401027
false
true
loc_401027:
❶
mov ecx, [ebp+var_4]
add ecx, 1
push ecx
push offset aID ; "i = %d"
call sub_40106D
add esp, 8
jmp short loc_401067
00401019:
cmp [ebp+var_8], 2
jz short loc_40103D
true
false
loc_40103D:
❷
mov edx, [ebp+var_4]
add edx, 2
push edx
push offset aID_0 ; "i = %d"
call sub_40106D
add esp, 8
jmp short loc_401067
0040101F:
cmp [ebp+var_8], 3
jz short loc_401053
true
false
loc_401053:
❸
mov eax, [ebp+var_4]
add eax, 3
push eax
push offset aID_1 ; "i = %d"
call sub_40106D
add esp, 8
00401025:
jmp short loc_401067
loc_401067:
xor eax, eax
mov esp, ebp
pop ebp
retn



多选题 10分

设置

Select the ways switch statements compiled

A If style

B Jump table

C Linked list

D Loop style

提交

南開大學 Nankai University

Arrays

- Arrays are used by programmers to define an ordered set of similar data items.

```
int b[5] = {123,87,487,7,978};
void main()
{
   int i;
   int a[5];

   for(i = 0; i<5; i++)
   {
      a[i] = i;
      b[i] = i;
   }
}
```

Arrays

- In assembly, arrays are accessed using a base address as a starting point.

```
00401006          mov     [ebp+var_18], 0
0040100D          jmp     short loc_401018
0040100F loc_40100F:
0040100F          mov     eax, [ebp+var_18]
00401012          add     eax, 1
00401015          mov     [ebp+var_18], eax
00401018 loc_401018:
00401018          cmp     [ebp+var_18], 5
0040101C          jge     short loc_401037
0040101E          mov     ecx, [ebp+var_18]
00401021          mov     edx, [ebp+var_18]
00401024          mov     [ebp+ecx*4+var_14], edx ❶
00401028          mov     eax, [ebp+var_18]
0040102B          mov     ecx, [ebp+var_18]
0040102E          mov     dword_40A000[ecx*4], eax ❷
00401035          jmp     short loc_40100F
```

Structures

- Structures (or structs, for short) are similar to arrays, but they comprise ele- ments of different types.
- Structures are commonly used by malware authors to group information.

Structures

```
struct my_structure { ❶
      int x[5];
      char y;
      double z;
};

struct my_structure *gms; ❷

void test(struct my_structure *q)
{
      int i;
      q->y = 'a';
      q->z = 15.6;
      for(i = 0; i<5; i++){
            q->x[i] = i;
      }
}

void main()
{
      gms = (struct my_structure *) malloc(
      sizeof(struct my_structure));
      test(gms);
}
```

Structures

- Structures (like arrays) are accessed with a base address used as a starting pointer.

```
00401050        push    ebp
00401051        mov     ebp, esp
00401053        push    20h
00401055        call    malloc
0040105A        add     esp, 4
0040105D        mov     dword_40EA30, eax
00401062        mov     eax, dword_40EA30
00401067        push    eax ❶
00401068        call    sub_401000
0040106D        add     esp, 4
00401070        xor     eax, eax
00401072        pop     ebp
00401073        retn
```

```
00401000        push    ebp
00401001        mov     ebp, esp
00401003        push    ecx
00401004        mov     eax,[ebp+arg_0]
00401007        mov     byte ptr [eax+14h], 61h
0040100B        mov     ecx, [ebp+arg_0]
0040100E        fld     ds:dbl_40B120 ❶
00401014        fstp    qword ptr [ecx+18h]
00401017        mov     [ebp+var_4], 0
0040101E        jmp     short loc_401029
00401020 loc_401020:
00401020        mov     edx,[ebp+var_4]
00401023        add     edx, 1
00401026        mov     [ebp+var_4], edx
00401029 loc_401029:
00401029        cmp     [ebp+var_4], 5
0040102D        jge     short loc_40103D
0040102F        mov     eax,[ebp+var_4]
00401032        mov     ecx,[ebp+arg_0]
00401035        mov     edx,[ebp+var_4]
00401038        mov     [ecx+eax*4],edx ❷
0040103B        jmp     short loc_401020
0040103D loc_40103D:
0040103D        mov     esp, ebp
0040103F        pop     ebp
00401040        retn
```

Linked List

- A linked list is a data structure that consists of a sequence of data records, and each record includes a field that contains a reference (link) to the next record in the sequence.

```
struct node
{
   int x;
   struct node * next;
};

typedef struct node pnode;

void main()
{
   pnode * curr, * head;
   int i;
   head = NULL;

   for(i=1;i<=10;i++) ❶
   {
      curr = (pnode *)malloc(sizeof(pnode));
      curr->x = i;
      curr->next  = head;
      head = curr;
   }

   curr = head;

   while(curr) ❷
   {
      printf("%d\n", curr->x);
      curr = curr->next ;
   }
}
```

```
0040106A        mov     [ebp+var_8], 0
00401071        mov     [ebp+var_C], 1
00401078
00401078 loc_401078:
00401078        cmp     [ebp+var_C], 0Ah
0040107C        jg      short loc_4010AB
0040107E        mov     [esp+18h+var_18], 8
00401085        call    malloc
0040108A        mov     [ebp+var_4], eax
0040108D        mov     edx, [ebp+var_4]
00401090        mov     eax, [ebp+var_C]
00401093        mov     [edx], eax ❶
00401095        mov     edx, [ebp+var_4]
00401098        mov     eax, [ebp+var_8]
0040109B        mov     [edx+4], eax ❷
0040109E        mov     eax, [ebp+var_4]
004010A1        mov     [ebp+var_8], eax
004010A4        lea     eax, [ebp+var_C]
004010A7        inc     dword ptr [eax]
004010A9        jmp     short loc_401078
004010AB loc_4010AB:
004010AB        mov     eax, [ebp+var_8]
004010AE        mov     [ebp+var_4], eax
004010B1
004010B1 loc_4010B1:
004010B1        cmp     [ebp+var_4], 0 ❸
004010B5        jz      short locret_4010D7
004010B7        mov     eax, [ebp+var_4]
004010BA        mov     eax, [eax]
004010BC        mov     [esp+18h+var_14], eax
004010C0        mov     [esp+18h+var_18], offset aD ; "%d\n"
004010C7        call    printf
004010CC        mov     eax, [ebp+var_4]
004010CF        mov     eax, [eax+4]
004010D2        mov     [ebp+var_4], eax ❹
004010D5        jmp     short loc_4010B1 ❺
```

Linked List

- To recognize a linked list, you must first recognize that some object contains a pointer that points to another object of the same type.
- The recursive nature of the objects is what makes it linked, and this is what you need to recognize from the disassembly.

Lab6

- Lab 6-1 3 questions
- Lab 6-2 6 questions
- Lab 6-3 6 questions
- Lab 6-4 6 questions

Yara规则提取-API函数名

```
strings:
    $Offset = {E8 00 00 00}

    $Close = "CloseHandle" fullword
    $Map = "MapViewOfFile" fullword

    $Mux = "CreateMutexA" fullword
    $Ser = "CreateServiceA"
    $Inet = "InternetOpenA" fullword
    $ip4 = /[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}/

    $w = "\\system32\\wupdmgr.exe"
    $EnumProc = "EnumProcessModules" fullword

    $c = "cmd.exe /c" fullword

    $lab3_3 = "NtUnmapViewOfSection" fullword
    $Month = "JanFebMarAprMayJunJulAugSepOctNovDec" fullword
    $Slp = "Sleep" nocase
```

Yara规则提取

```
private rule MalFileString
{
meta:
  feature = "mal filename strings"
strings:
  $str1 = "kerne132.dll" nocase wide ascii fullword
  $str2 = "wupdmgrd.exe" nocase wide ascii fullword
  $str3 = "practicalmalwareanalysis.log" nocase wide ascii fullword
  $str4 = "updater.exe" nocase wide ascii fullword
  $str5 = "wupdmgrd.exe" nocase wide ascii fullword
condition:
  any of ($str*)
}
```

Yara规则提取-文件哈希值

```
rule lab1_1exe     { strings: $1="kerne132.dll" fullword ascii condition: $1}
rule lab1_1dll     {strings: $1="127.26.152.13"fullword $2="sleep" fullword condition:$1 and $2}
rule lab1_2        {strings: $1="SystemTimeToFile" fullword $2="practicalmalwareanalysis.com" fullword condition: $1
and $2}//脱壳后
rule lab1_3        {strings: $1="IMSVCRTT" fullword $2="OLEAUTLA" fullword condition: $1 and $2}
rule lab1_4        {strings: $1="\\system32\\wupdmgrd.exe" fullword condition: $1 }
rule lab3_1        {strings: $1="vmx32to64.exe" fullword $2="VedioDriver" fullword condition: $1 and $2 }
rule lab3_2        {strings: $1="practicalmalwareanalysis" $2="INA+"fullword $3="IPRIP" fullword condition: $1 and $2
and $3}
import "hash" rule lab3_3          {condition: hash.md5(0,filesize)=="e2bf42217a67e46433da8b6f4507219e"}
rule lab3_4        {strings: $1="Manager Service"fullword $2="UPLOAD"fullword $3="HTTP/1.0"fullword condition:$1 and
$2 and $3}
```

Yara规则提取-文件大小过滤、Magic String特征过滤

```
*/
rule lab0101exe{
    strings:
        $s1="ker
    condition:
        uint16(0)
}

private rule IsPE
{
    condition:
                uint16(0) == 0x5A4D and // MZ 头
                uint32(uint32(0x3C)) == 0x00004550 // "PE"头
}
```

```
41  private rule entry13
42  {
43      strings:
44
45          $ep3 = { BB D0 01 40 ?? BF ?? 10 40 ?? BE }
46
47      condition:
48
49          for any of ($ep*) : ($ at pe.entry_point)
50  }
51
52  rule is__lab0103{
53
54      condition:
55              entry13 and
56              filesize == 4752
```

Yara规则提取-入口点

```
rule Lab0102
{
        strings:
                $a="Malservice" nocase
                $c={60 BE 00 50}
        condition: $a and ($c at entrypoint)
}
```

Yara特征提取-加壳文件

```
}
//Lab01-02
rule UPX_encryption{
meta:
        description = "If been UPX_encrypted."
strings:
        $UPX1 = "UPX1" wide ascii
        $UPX2 = "UPX2" wide ascii
        $UPX3 = "UPX3" wide ascii
condition:
        any of them
}
```

Yara规则提取-尽量简短

```
strings:
        $st01 = "www.practicalmalwareanalysis
        $st02 = "practicalmalwareanalysis.com
        $st03 = "practicalmalwareanalysis.com
        $st04 = "Windows XP 6.11" fullword nc
        $st05 = "http://practicalmalwareanaly
        $st06 = "practicalmalware.log" fullwc
        $st07 = "http://practicalmalwareanaly
        $st10 = "http://www.malwareanalysisbc
        $st11="kernel32.dll"fullword nocase w
condition:
        1 of them
```

Agenda

- Global vs Local Variables
- Arithmetic Operations
- If statement
- For and While Loops
- Function Call Convention
- Switch Statement
- Arrays, Structures, Linked List

Malware Analysis

Chapter 6: Recognizing C Constructs in Assembly

王志
zwang@nankai.edu.cn

updated on Oct. 17th 2021

College of Cyber Science
Nankai University
2021/2022