Advanced Topics in Malware Analysis

High Level Language Constructs In Assembly

Brendan Saltaformaggio, PhD
Assistant Professor
School of Electrical and Computer Engineering

Function calls, If/then/else, Switch

Georgia Tech

1

Learning Objectives

- Interpret high level structures in assembler
- Identify function calls
- Reconstruct switch expressions to index table of targets
- Employ loops and compiler-induced elimination of branching
- Discover arrays
- Identify structures in assembly code

Georgia Tech

2

Understanding HLL Structures in ASM

Need to understand representation of compiled high-level structures in assembler

- Functions
  - Calls and Returns discussed earlier
- Control structures
  - Loops
  - Conditional branches
  - if/then/else
  - switch statements
- Data structures
  - structs/unions
  - Arrays
  - Linked structures

3

IDA Annotated Function Calls

Full annotation
(PUSHed args)

```
.text:004010F3        push    10h                 ; namelen
.text:004010F5        lea     ecx, ❶[ebp+name]
.text:004010F8        push    ecx                 ; name
.text:004010F9        mov     edx, ❶[ebp+s]
.text:004010FF        push    edx                 ; s
.text:00401100        call    connect
```

Partial annotation
(MOVs)

```
.text:004011A5        mov     [esp+244h+var_23C], 10h
.text:004011AD        lea     eax, ❷[ebp+var_28]
.text:004011B0        mov     [esp+244h+var_240], eax
.text:004011B4        mov     eax, ❷[ebp+var_C]
.text:004011B7        mov     [esp+244h+var_244], eax
.text:004011BA        call    connect
```

4

if/then/else

Typically an assembly sequence that:

1. Computes the expression in the if portion
2. Performs a test or cmp to ensure that CPU flags are set
3. Performs a jnz/jge/etc. and a jmp

```
    cmp [z], 5          ; if z < 5 then
    jge Else
    call func_if        ;   func_if()
    jmp After
Else:                   ; else
    call func_else ;    func_else()
After:                  ; end if
```

Georgia Tech

5

Switch Case Statements

Several possible implementations:

- **Table Implementation**
  - Use a table of targets for an unconditional JMP
  - Use switch expression as index into the table of targets
  - This implementation isn't appropriate when there are numerous gaps in the case statements
  - Problem: Jump table may be embedded in the code section—problem for linear sweep disassembly
- **Tree Implementation**
  - Implements a binary search for the matching case

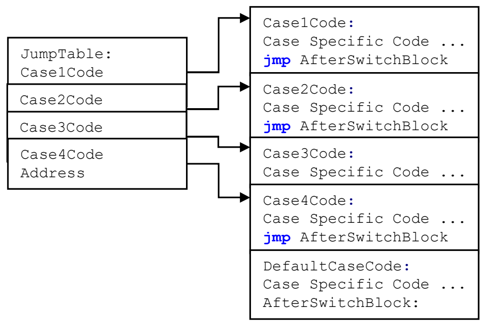
Georgia Tech

6

Switch: Table Implementation "Idea"

```
switch (a) {
 case 1:
   Case Specific Code ...
   break;
 case 2:
   Case Specific Code ...
   break;
 case 3: // no break!
   Case Specific Code ...
 case 4:
   Case Specific Code ...
   break;
 default:
   Case Specific Code ...
}
```

```
movzx eax, BYTE PTR [a]
sub eax, 1
cmp eax, 3
ja DefaultCaseCode
jmp DWORD PTR [JumpTable + eax * 4]
```



Georgia Tech

7

Switch: Table Implementation Real Code

- Compiled with gcc on 64-bit Mac with no optimizations (i.e., -O0)

```
#include <stdio.h>
void main(void) {
  int a;
  scanf("%d", &a);
  switch (a) {
  case 0:
    puts ("Zero");
    break;
  case 1:
    puts ("One");
    break;
  case 2:
    puts ("Two");
    break;
  case 3:
    puts ("Three");
    break;
  default:
    puts("No idea.");
  }
}
```

Georgia Tech

8

Switch: Table Implementation Real Code

```c
#include <stdio.h>
void main(void) {
  int a;
  scanf("%d", &a);
  switch (a) {
  case 0:
    puts ("Zero");
    break;
  case 1:
    puts ("One");
    break;
  case 2:
    puts ("Two");
    break;
  case 3:
    puts ("Three");
    break;
  default:
    puts("No idea.");
  }
}
```

```
__cstring:0000000100000F46 ; ==================================
__cstring:0000000100000F46
__cstring:0000000100000F46 ; Segment type: Pure data
__cstring:0000000100000F46 __cstring       segment byte public
__cstring:0000000100000F46                 assume cs:__cstring
__cstring:0000000100000F46                 ;org 100000F46h
__cstring:0000000100000F46 ; char aD[]
__cstring:0000000100000F46 aD              db '%d',0
__cstring:0000000100000F49 ; char aZero[]
__cstring:0000000100000F49 aZero           db 'Zero',0
__cstring:0000000100000F4E ; char aOne[]
__cstring:0000000100000F4E aOne            db 'One',0
__cstring:0000000100000F52 ; char aTwo[]
__cstring:0000000100000F52 aTwo            db 'Two',0
__cstring:0000000100000F56 ; char aThree[]
__cstring:0000000100000F56 aThree          db 'Three',0
__cstring:0000000100000F5C ; char aNoIdea_[]
__cstring:0000000100000F5C aNoIdea_        db 'No idea.',0
__cstring:0000000100000F5C __cstring       ends
__cstring:0000000100000F5C
```

Georgia Tech

9

Switch: Table Implementation Real Code

```c
#include <stdio.h>
void main(void) {
  int a;
  scanf("%d", &a);
  switch (a) {
  case 0:
    puts ("Zero");
    break;
  case 1:
    puts ("One");
    break;
  case 2:
    puts ("Two");
    break;
  case 3:
    puts ("Three");
    break;
  default:
    puts("No idea.");
  }
}
```

```
                public _main
_main           proc near               ; CODE XREF: start

var_10          = qword ptr -10h
var_4           = dword ptr -4

                push    rbp
                mov     rbp, rsp
                sub     rsp, 10h
                lea     rax, [rbp+var_4]
                xor     cl, cl
                lea     rdx, aD         ; "%d"
                mov     rdi, rdx        ; char *
                mov     rsi, rax
                mov     al, cl
                call    _scanf
                mov     eax, [rbp+var_4]
                cmp     eax, 3          ; switch 4 cases
                mov     eax, eax
                mov     [rbp+var_10], rax
                ja      short loc_100000EDC ; default case
                lea     rax, off_100000EF4
                mov     rcx, [rbp+var_10]
                mov     rdx, rcx
                movsxd  rdx, dword ptr [rax+rdx*4]
                lea     rax, [rdx+rax]
                jmp     rax             ; switch jump
; ---------------------------------------------------------------
```

Georgia Tech

10

Switch: Table Implementation Real Code

```c
#include <stdio.h>
void main(void) {
  int a;
  scanf("%d", &a);
  switch (a) {
  case 0:
    puts ("Zero");
    break;
  case 1:
    puts ("One");
    break;
  case 2:
    puts ("Two");
    break;
  case 3:
    puts ("Three");
    break;
  default:
    puts("No idea.");
  }
}
```

```
; ---------------------------------------------------------------------------
loc_100000EDC:                          ; CODE XREF: _main+2E↑j
                lea     rax, aNoIdea_   ; jumptable 0000000100000E96 default
                mov     rdi, rax        ; char *
                call    _puts

loc_100000EEB:                          ; CODE XREF: _main+57↑j
                                        ; _main+68↑j ...
                add     rsp, 10h
                pop     rbp
                retn
_main           endp
```

11

Switch: Table Implementation Real Code

```c
#include <stdio.h>
void main(void) {
  int a;
  scanf("%d", &a);
  switch (a) {
  case 0:
    puts ("Zero");
    break;
  case 1:
    puts ("One");
    break;
  case 2:
    puts ("Two");
    break;
  case 3:
    puts ("Three");
    break;
  default:
    puts("No idea.");
  }
}
```

```
                public _main
_main           proc near               ; CODE XREF: start

var_10          = qword ptr -10h
var_4           = dword ptr -4

                push    rbp
                mov     rbp, rsp
                sub     rsp, 10h
                lea     rax, [rbp+var_4]
                xor     cl, cl
                lea     rdx, aD         ; "%d"
                mov     rdi, rdx        ; char *
                mov     rsi, rax
                mov     al, cl
                call    _scanf
                mov     eax, [rbp+var_4]
                cmp     eax, 3          ; switch 4 cases
                mov     eax, eax
                mov     [rbp+var_10], rax
                ja      short loc_100000EDC ; default case
                lea     rax, off_100000EF4
                mov     rcx, [rbp+var_10]
                mov     rdx, rcx
                movsxd  rdx, dword ptr [rax+rdx*4]
                lea     rax, [rdx+rax]
                jmp     rax             ; switch jump
; ---------------------------------------------------------------------------
```

12

Switch: Table Implementation Real Code

```c
#include <stdio.h>
void main(void) {
  int a;
  scanf("%d", &a);
  switch (a) {
  case 0:
    puts ("Zero");
    break;
  case 1:
    puts ("One");
    break;
  case 2:
    puts ("Two");
    break;
  case 3:
    puts ("Three");
    break;
  default:
    puts("No idea.");
  }
}
```

```
; ---------------------------------------------------------------------------
                align 4
off_100000EF4   dd offset loc_100000E98 - 100000EF4h
                                        ; DATA XREF: _main+30↑o
                dd offset loc_100000EA9 - 100000EF4h ; jump table for switch
                dd offset loc_100000EBA - 100000EF4h
                dd offset loc_100000ECB - 100000EF4h
__text          ends
```

Georgia Tech

13

Switch: Table Implementation Real Code

```c
#include <stdio.h>
void main(void) {
  int a;
  scanf("%d", &a);
  switch (a) {
  case 0:
    puts ("Zero");
    break;
  case 1:
    puts ("One");
    break;
  case 2:
    puts ("Two");
    break;
  case 3:
    puts ("Three");
    break;
  default:
    puts("No idea.");
  }
}
```

```
                public _main
_main           proc near               ; CODE XREF: start

var_10          = qword ptr -10h
var_4           = dword ptr -4

                push    rbp
                mov     rbp, rsp
                sub     rsp, 10h
                lea     rax, [rbp+var_4]
                xor     cl, cl
                lea     rdx, aD         ; "%d"
                mov     rdi, rdx        ; char *
                mov     rsi, rax
                mov     al, cl
                call    _scanf
                mov     eax, [rbp+var_4]
                cmp     eax, 3          ; switch 4 cases
                mov     eax, eax
                mov     [rbp+var_10], rax
                ja      short loc_100000EDC ; default case
                lea     rax, off_100000EF4
                mov     rcx, [rbp+var_10]
                mov     rdx, rcx
                movsxd  rdx, dword ptr [rax+rdx*4]
                lea     rax, [rdx+rax]
                jmp     rax             ; switch jump
; ---------------------------------------------------------------------------
```

Georgia Tech

14

Switch: Table Implementation Real Code

```c
#include <stdio.h>
void main(void) {
  int a;
  scanf("%d", &a);
  switch (a) {
  case 0:
    puts ("Zero");
    break;
  case 1:
    puts ("One");
    break;
  case 2:
    puts ("Two");
    break;
  case 3:
    puts ("Three");
    break;
  default:
    puts("No idea.");
  }
}
```

```
loc_100000E98:                          ; CODE XREF: _main+46↑j
                                        ; DATA XREF: __text:off_100000EF4↓o
                lea     rax, aZero      ; jumptable 0000000100000E96 case 0
                mov     rdi, rax        ; char *
                call    _puts
                jmp     short loc_100000EEB
; ---------------------------------------------------------------------------
loc_100000EA9:                          ; CODE XREF: _main+46↑j
                                        ; DATA XREF: __text:off_100000EF4↓o
                lea     rax, aOne       ; jumptable 0000000100000E96 case 1
                mov     rdi, rax        ; char *
                call    _puts
                jmp     short loc_100000EEB
; ---------------------------------------------------------------------------
loc_100000EBA:                          ; CODE XREF: _main+46↑j
                                        ; DATA XREF: __text:off_100000EF4↓o
                lea     rax, aTwo       ; jumptable 0000000100000E96 case 2
                mov     rdi, rax        ; char *
                call    _puts
                jmp     short loc_100000EEB
; ---------------------------------------------------------------------------

loc_100000ECB:                          ; CODE XREF: _main+46↑j
                                        ; DATA XREF: __text:off_100000EF4↓o
                lea     rax, aThree     ; jumptable 0000000100000E96 case 3
                mov     rdi, rax        ; char *
                call    _puts
                jmp     short loc_100000EEB
; ---------------------------------------------------------------------------
loc_100000EDC:                          ; CODE XREF: _main+2E↑j
                lea     rax, aNoIdea_   ; jumptable 0000000100000E96 default
                mov     rdi, rax        ; char *
                call    _puts

loc_100000EEB:                          ; CODE XREF: _main+57↑j
                                        ; _main+68↑j ...
                add     rsp, 10h
                pop     rbp
                retn
_main           endp

; ---------------------------------------------------------------------------
                align 4
off_100000EF4   dd offset loc_100000E98 - 100000EF4h
                                        ; DATA XREF: _main+30↑o
                dd offset loc_100000EA9 - 100000EF4h ; jump table for switch
                dd offset loc_100000EBA - 100000EF4h
                dd offset loc_100000ECB - 100000EF4h
__text          ends
```

15

Compiled with -O2

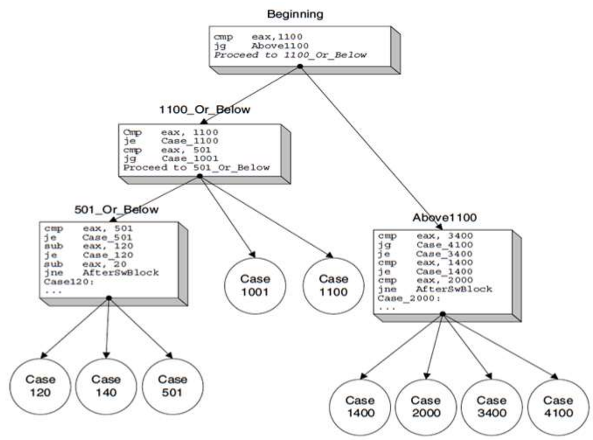
```c
#include <stdio.h>
void main(void) {
  int a;
  scanf("%d", &a);
  switch (a) {
  case 0:
    puts ("Zero");
    break;
  case 1:
    puts ("One");
    break;
  case 2:
    puts ("Two");
    break;
  case 3:
    puts ("Three");
    break;
  default:
    puts("No idea.");
  }
}
```

```
                lea     rdi, aD         ; "%d"
                lea     rsi, [rbp+var_4]
                xor     al, al
                call    _scanf
                mov     eax, [rbp+var_4]
                cmp     rax, 3          ; default case
                ja      short loc_100000EE0 ; jumptable 0000000100000EB1 def
                lea     rcx, off_100000EEC
                movsxd  rax, dword ptr [rcx+rax*4]
                add     rax, rcx
                jmp     rax             ; switch jump
; ---------------------------------------------------------------------------

loc_100000EB3:                          ; CODE XREF: _main+31↑j
                                        ; DATA XREF: __text:off_100000EEC↓o
                lea     rdi, aZero      ; jumptable 0000000100000EB1 case 0

loc_100000EBA:                          ; CODE XREF: _main+4C↓j
                                        ; _main+55↓j ...
                call    _puts
                add     rsp, 10h
                pop     rbp
                retn
; ---------------------------------------------------------------------------

loc_100000EC5:                          ; CODE XREF: _main+31↑j
                                        ; DATA XREF: __text:off_100000EEC↓o
                lea     rdi, aOne       ; jumptable 0000000100000EB1 case 1
                jmp     short loc_100000EBA
; ---------------------------------------------------------------------------

loc_100000ECE:                          ; CODE XREF: _main+31↑j
                                        ; DATA XREF: __text:off_100000EEC↓o
                lea     rdi, aTwo       ; jumptable 0000000100000EB1 case 2
                jmp     short loc_100000EBA
; ---------------------------------------------------------------------------

loc_100000ED7:                          ; CODE XREF: _main+31↑j
                                        ; DATA XREF: __text:off_100000EEC↓o
                lea     rdi, aThree     ; jumptable 0000000100000EB1 case 3
                jmp     short loc_100000EBA
; ---------------------------------------------------------------------------

loc_100000EE0:                          ; CODE XREF: _main+21↑j
                lea     rdi, aNoIdea_   ; jumptable 0000000100000EB1 default
                jmp     short loc_100000EBA
_main           endp

; ---------------------------------------------------------------------------
                align 4
off_100000EEC   dd offset loc_100000EB3 - 100000EECh
                                        ; DATA XREF: _main+23↑o
                dd offset loc_100000EC5 - 100000EECh ; jump table for switch
                dd offset loc_100000ECE - 100000EECh
                dd offset loc_100000ED7 - 100000EECh
```

16

Switch: Binary Search

```
switch (value) {
  case 120: Code...
  break;
  case 140: Code...
  break;
  case 501: Code...
  break;
  case 1001: Code...
  break;
  case 1100: Code...
  break;
  case 1400: Code...
  break;
  case 2000: Code...
  break;
  case 3400: Code...
  break;
  case 4100: Code...
  break;
};
```



Source: Eilam, E. (2011). *Reversing: secrets of reverse engineering*. Hoboken, NJ: John Wiley & Sons.

17

Brendan Saltaformaggio, PhD.
Assistant Professor
School of Electrical and Computer Engineering

18

Advanced Topics in Malware Analysis

High Level Language Constructs In Assembly

Brendan Saltaformaggio, PhD

Assistant Professor

School of Electrical and Computer Engineering

Loops

Georgia Tech

19

Loops

- Loops are much as you'd expect, except…
- Pretested loops (e.g., while/do) can end up being post-tested (do/while)

```
c = 0;
while (c < 1000)
{
    array[c] = c;
    c++;
}
```

```
    mov ebx, DWORD PTR [array]
    xor ecx, ecx                        ; c = 0
Loop:                                   ; do {
    mov DWORD PTR [ebx+ecx*4], ecx        ;    array[c]=c
    add ecx, 1                            ;    c++
    cmp ecx, 1000                       ; }
    jl Loop                             ; while (c < 1000)
```

Georgia Tech

20

Loops: Rep Instruction

- **rep** instruction prefix
  - Syntax: REP [INS] (INS operands)
- Repeats an instruction the number of times specified in the RCX/ECX/CX register or until the indicated condition of the ZF flag is no longer met

Repeat Prefix	Termination Condition 1	Termination Condition 2
REP	ECX=0	None
REPE/REPZ	ECX=0	ZF=0
REPNE/REPNZ	ECX=0	ZF=1

- Used to replace loops which consists of only one instruction (most often moving data)
  - To repeat a block of instructions, the LOOP instruction can be used (very rare) or a logical looping construct

Georgia Tech

21

Rep Example: Libc memcpy

```
__memcpy_g_internal proc near          ; CODE XREF: __strcat_c+36↓p
                                       ; __memcpy_c+26↓p

arg_0           = dword ptr  4
arg_4           = dword ptr  8
arg_8           = dword ptr  0Ch

                push    ebp             ; save clobbered regs
                push    edi
                push    esi
                mov     eax, [esp+0Ch+arg_0] ; eax = dest
                mov     ebp, [esp+0Ch+arg_4] ; ebp = source
                mov     ecx, [esp+0Ch+arg_8] ; ecx = size
                mov     edi, eax        ; edi = dest
                mov     esi, ebp        ; esi = source
                cld                     ; direction flag = 0, inc pointer after iteration
                shr     ecx, 1          ; ecx = ecx / 2
                jnb     short loc_83879 ; jump if bit shifted off ecx was 0, otherwise ...
                movsb                   ; move 1 byte from [esi] to [edi] & inc esi and edi

loc_83879:                              ; CODE XREF: __memcpy_g_internal+16↑j
                shr     ecx, 1          ; ecx = ecx / 2
                jnb     short loc_8387F ; jump if bit shifted off ecx was 0, otherwise ...
                movsw                   ; move 1 word from [esi] to [edi] & add 2 to esi and edi

loc_8387F:                              ; CODE XREF: __memcpy_g_internal+1B↑j
                rep movsd               ; while(ecx > 0)
                                        ;   move 1 DWORD from [esi] to [edi]
                                        ;   add 4 to esi and edi
                                        ;   decrement ecx

                pop     esi
                pop     edi             ; restore saved regs
                pop     ebp
                retn                    ; return to caller
```

Georgia Tech

22

Loop Unrolling

- Duplicate loop body and reduces the number of iterations
- **Goal**: Eliminate as much branching as possible at the expense of increased code size

```
int main(int argc, char* argv[])
{
    char dest[64];
    for(int i = 0; i < 64; i++)
        dest[i] = argv[1][i];
    puts(dest);
}
```

- gcc -W -Wall -Wextra -Wpedantic -O2 -S -masm=intel **-funroll-loops** fill_array.c -o fill_array.s
- Compiled on 64-bit Windows

```
main:
    push rbx ;save rbx
    sub  rsp, 96
    mov  rbx, rdx ; rbx = argv
    mov  rdx, QWORD PTR 8[rbx] ; rdx = argv[1]
    lea  rcx, 32[rsp] ; rcx = dest
    xor  eax, eax
.L2:
    movzx   r8d, BYTE PTR [rdx+rax]
    mov  BYTE PTR [rcx+rax], r8b
    movzx   r9d, BYTE PTR 1[rdx+rax]
    mov  BYTE PTR 1[rcx+rax], r9b
    movzx   r10d, BYTE PTR 2[rdx+rax]
    mov  BYTE PTR 2[rcx+rax], r10b
    movzx   r11d, BYTE PTR 3[rdx+rax]
    mov  BYTE PTR 3[rcx+rax], r11b
    movzx   ebx, BYTE PTR 4[rdx+rax]
    mov  BYTE PTR 4[rcx+rax], bl
    movzx   r8d, BYTE PTR 5[rdx+rax]
    mov  BYTE PTR 5[rcx+rax], r8b
    movzx   r9d, BYTE PTR 6[rdx+rax]
    mov  BYTE PTR 6[rcx+rax], r9b
    movzx   r10d, BYTE PTR 7[rdx+rax]
    mov  BYTE PTR 7[rcx+rax], r10b
    add  rax, 8
    cmp  rax, 64
    jne  .L2
    call puts
    xor  eax, eax
    add  rsp, 96
    pop  rbx
    ret
```

23

Loop Unrolling

- **Extreme Case**: Complete unroll
  - No loop — just replicated loop body

```
int main(int argc, char* argv[])
{
    char dest[64];
    for(int i = 0; i < 64; i++)
        dest[i] = argv[1][i];
    puts(dest);
}
```

- gcc -W -Wall -Wextra -Wpedantic **-O3** -S -masm=intel fill_array.c -o fill_array.s
- Compiled on 64-bit Windows

```
main:
    push rbx ;save rbx
    sub  rsp, 96
    mov  rbx, rdx ; rbx = argv
    mov  r8, QWORD PTR 8[rbx] ; r8=argv[1]
     ; copy 64 bytes unrolled
    mov  rax, QWORD PTR [r8]
    mov  QWORD PTR 32[rsp], rax
    mov  rax, QWORD PTR 8[r8]
    mov  QWORD PTR 40[rsp], rax
    mov  rax, QWORD PTR 16[r8]
    mov  QWORD PTR 48[rsp], rax
    mov  rax, QWORD PTR 24[r8]
    mov  QWORD PTR 56[rsp], rax
    mov  rax, QWORD PTR 32[r8]
    mov  QWORD PTR 64[rsp], rax
    mov  rax, QWORD PTR 40[r8]
    mov  QWORD PTR 72[rsp], rax
    mov  rax, QWORD PTR 48[r8]
    mov  QWORD PTR 80[rsp], rax
    mov  rax, QWORD PTR 56[r8]
    mov  QWORD PTR 88[rsp], rax

    call puts
    xor  eax, eax
    add  rsp, 96
    pop  rbx
    ret
```

24

Brendan Saltaformaggio, PhD.
Assistant Professor
School of Electrical and Computer Engineering

25

Advanced Topics in Malware Analysis

High Level Language Constructs In Assembly

Brendan Saltaformaggio, PhD
Assistant Professor
School of Electrical and Computer Engineering

Arrays

Georgia Tech

26

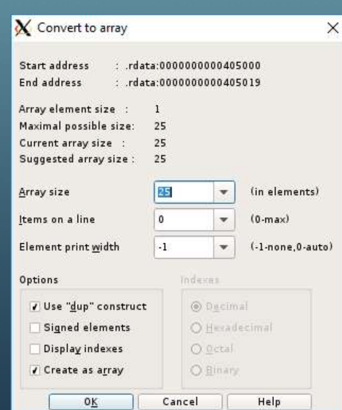
Arrays

- Arrays indexed by a variable can generally be located easily in ASM
- Sometimes possible to determine length of array by closely analyzing code
- Arrays indexed by a constant are harder
  - Compiler will optimize
  - ASM looks like structure accesses or accesses to individual variables
- Accesses to dynamically allocated arrays are easier to deduce
  - There will generally be a call to a memory allocation function
  - Accesses to individual elements are offsets from the base address of the allocation

27

Arrays In IDA

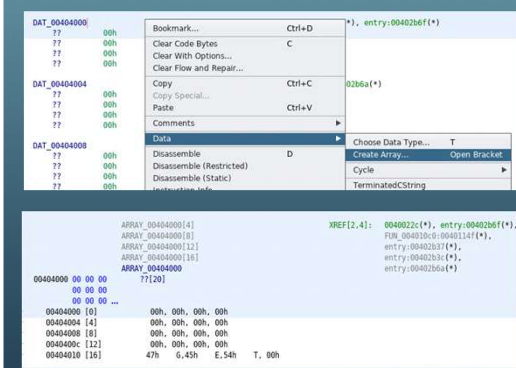
- IDA will try to be smart about array identification
- It can miss some
  - Especially arrays of union data types!
- IDA can be coaxed into displaying arrays properly via Right-Click->Array
  - see Eagle, Chapter 8 for more



28

Arrays In Ghidra

- Ghidra has similar functionality
  - A little less fleshed out
- Right Click → Data → Create Array
- Then set the size in bytes



Georgia Tech

29

Global Array Example

```c
int global_array[3];

int main() {
   int idx = 2;
   global_array[0] = 10;
   global_array[1] = 20;
   global_array[2] = 30;
   global_array[idx] = 40;
}
```

```
.text:00401000 _main          proc near
.text:00401000
.text:00401000 idx            = dword ptr -4
.text:00401000
.text:00401000                push    ebp
.text:00401001                mov     ebp, esp
.text:00401003                push    ecx
.text:00401004                mov     [ebp+idx], 2
.text:0040100B               ❶mov     dword_40B720, 10
.text:00401015               ❷mov     dword_40B724, 20
.text:0040101F               ❸mov     dword_40B728, 30
.text:00401029                mov     eax, [ebp+idx]
.text:0040102C               ❹mov     dword_40B720[eax*4], 40
.text:00401037                xor     eax, eax
.text:00401039                mov     esp, ebp
.text:0040103B                pop     ebp
.text:0040103C                retn
.text:0040103C _main          endp
```

Global/static array example

- (1), (2), (3) are constant accesses
- (4) is an indexed access
- Indexed access points out size of individual elements

Source: Eagle, C. (2011). *The Ida Pro Book: The unofficial guide to the worlds most popular disassembler*. San Francisco: No Starch Press.

Georgia Tech

30

heap Array Example

```
int main() {
   int *heap_array = (int*)malloc(3 * sizeof(int));
   int idx = 2;
   heap_array[0] = 10;
   heap_array[1] = 20;
   heap_array[2] = 30;
   heap_array[idx] = 40;
}
```

```
.text:00401000 _main      proc near
.text:00401000
.text:00401000 heap_array      = dword ptr -8
.text:00401000 idx             = dword ptr -4
.text:00401000
.text:00401000            push    ebp
.text:00401001            mov     ebp, esp
.text:00401003            sub     esp, 8
.text:00401006           ❺push    0Ch             ; size_t
.text:00401008            call    _malloc
.text:0040100D            add     esp, 4
.text:00401010            mov     [ebp+heap_array], eax
.text:00401013            mov     [ebp+idx], 2
.text:0040101A            mov     eax, [ebp+heap_array]
.text:0040101D           ❶mov     dword ptr [eax], 10
.text:00401023            mov     ecx, [ebp+heap_array]
.text:00401026           ❷mov     dword ptr [ecx+4], 20
.text:0040102D            mov     edx, [ebp+heap_array]
.text:00401030           ❸mov     dword ptr [edx+8], 30
.text:00401037            mov     eax, [ebp+idx]

.text:0040103A            mov     ecx, [ebp+heap_array]
.text:0040103D           ❹mov     dword ptr [ecx+eax*4], 40
.text:00401044            xor     eax, eax
.text:00401046            mov     esp, ebp
.text:00401048            pop     ebp
.text:00401049            retn
.text:00401049 _main      endp
```

- Heap-allocated array
- (5) reveals size
- (1) – (4) are indexed accesses

Source: Eagle, C. (2011). *The Ida Pro Book: The unofficial guide to the worlds most popular disassembler*. San Francisco: No Starch Press.

31

Brendan Saltaformaggio, PhD.
Assistant Professor
School of Electrical and Computer Engineering

32

Advanced Topics in Malware Analysis

High Level Language Constructs In Assembly

Brendan Saltaformaggio, PhD

Assistant Professor

School of Electrical and Computer Engineering

Struct/Union Analysis

Georgia Tech

33

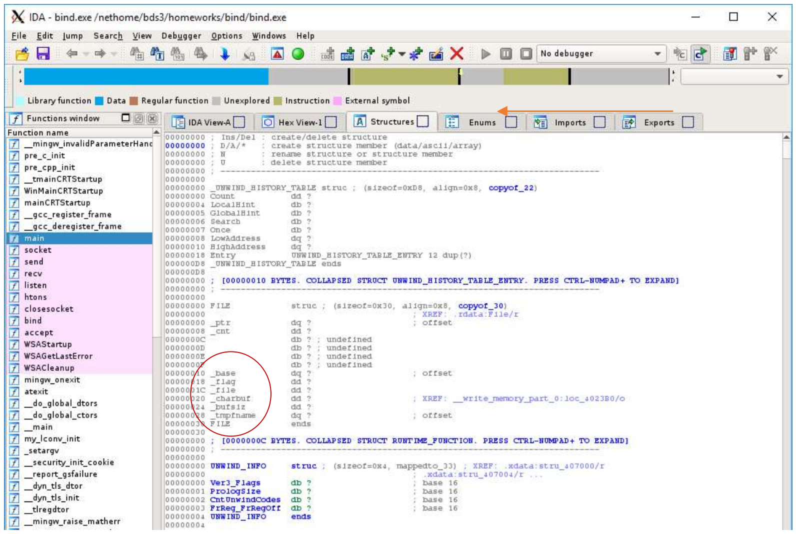
Struct/Union/etc.

- Field names are lost
  - Only remain with debugging symbols (which malware doesn't like)
  - Will be replaced with offsets by the compiler
- Global/static allocation:
  - Accesses to structure elements look like accesses to arrays via constant indices
- Dynamic allocation:
  - Similar to arrays, seeing memory allocation may help
- Struct / Union analysis is complicated by alignment issues!
- Good news! For common structures (e.g., struct sockaddr, etc.), IDA Pro may be able to automatically recognize struct accesses
- If recognition for particular struct types isn't available, you can add it manually in IDA

Georgia Tech

34

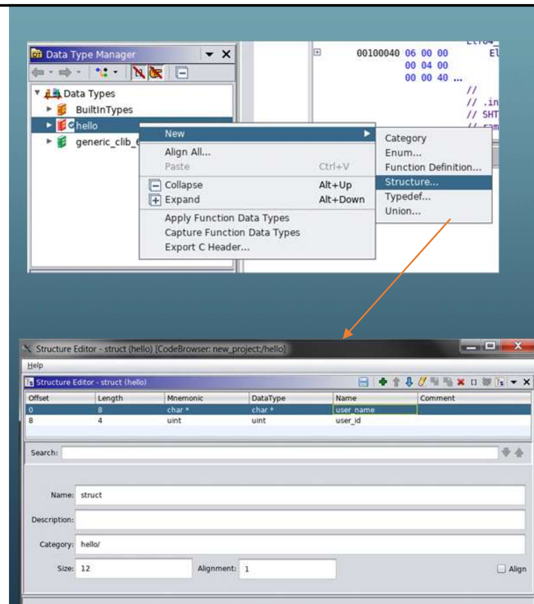
Struct/Union in ida



35

Struct/Union in Ghidra

- Ghidra does structs/unions much cleaner than IDA!
- Plus this functionality is extremely useful when paired with Ghidra's decompiler
- E.g.: If Ghidra detects a class, you can define each of its members in memory
- Makes decompiled code *significantly* more readable



36

Structure Alignment Issues

- Structures defined in source code only provide field names & types
- Compilers must lay those fields out in memory
  - Different compilers/flags will choose different field alignments
- For example: A memory-efficient compilation should choose a minimal size (packed)
- But: A processor-cache-efficient compilation may choose a power of 2 alignment
- This can confuse reverse engineers --- is the extra space actually another field?

```
struct ch8_struct {    //Size      Minimum offset      Default offset
    int field1;        //  4             0                  0
    short field2;      //  2             4                  4
    char field3;       //  1             6                  6
    int field4;        //  4             7                  8
    double field5;     //  8            11                 16
};                  //Minimum total size: 19   Default size: 24
```

Source: Eagle, Chris. The IDA pro book. No Starch Press, 2011

Georgia Tech

37

Global struct Example

```
struct ch8_struct global_struct;

int main() {
   global_struct.field1 = 10;
   global_struct.field2 = 20;
   global_struct.field3 = 30;
   global_struct.field4 = 40;
   global_struct.field5 = 50.0;
}
```

```
.text:00401000 _main        proc near
.text:00401000              push    ebp
.text:00401001              mov     ebp, esp
.text:00401003              mov     dword_40EA60, 10
.text:0040100D              mov     word_40EA64, 20
.text:00401016              mov     byte_40EA66, 30
.text:0040101D              mov     dword_40EA68, 40
.text:00401027              fld     ds:dbl_40B128
.text:0040102D              fstp    dbl_40EA70
.text:00401033              xor     eax, eax
.text:00401035              pop     ebp
.text:00401036              retn
.text:00401036 _main        endp
```

Global struct

- All offsets computed at compile time
- No runtime computation
- Hard to distinguish between struct and array access, except that elements of C arrays have equal size

Source: Eagle, C. (2011). *The Ida Pro Book: The unofficial guide to the worlds most popular disassembler*. San Francisco: No Starch Press.

Georgia Tech

38

Heap struct Example

```
.text:00401004    ❻push  24                     ; size_t
.text:00401006     call  _malloc
.text:0040100B     add   esp, 4
.text:0040100E     mov   [ebp+heap_struct], eax
.text:00401011     mov   eax, [ebp+heap_struct]
.text:00401014    ❶mov   dword ptr [eax], 10
.text:0040101A     mov   ecx, [ebp+heap_struct]
.text:0040101D    ❷mov   word ptr [ecx+4], 20
.text:00401023     mov   edx, [ebp+heap_struct]
.text:00401026    ❸mov   byte ptr [edx+6], 30
.text:0040102A     mov   eax, [ebp+heap_struct]
.text:0040102D    ❹mov   dword ptr [eax+8], 40
.text:00401034     mov   ecx, [ebp+heap_struct]
```

Heap-allocated struct

- As for dynamically allocated arrays, size is known from malloc() call
- Next slide: struct is packed (single byte alignment)

Source: Eagle, C. (2011). *The Ida Pro Book: The unofficial guide to the worlds most popular disassembler*. San Francisco: No Starch Press.

Georgia Tech

39

Same struct, but **compiler uses different alignment**

```
.text:0040100E     mov   [ebp+heap_struct], eax
.text:00401011     mov   eax, [ebp+heap_struct]
.text:00401014     mov   dword ptr [eax], 10
.text:0040101A     mov   ecx, [ebp+heap_struct]
.text:0040101D     mov   word ptr [ecx+4], 20
.text:00401023     mov   edx, [ebp+heap_struct]
.text:00401026     mov   byte ptr [edx+6], 30
.text:0040102A     mov   eax, [ebp+heap_struct]
.text:0040102D     mov   dword ptr [eax+7], 40
```

Source: Eagle, C. (2011). *The Ida Pro Book: The unofficial guide to the worlds most popular disassembler*. San Francisco: No Starch Press.

Georgia Tech

40

Brendan Saltaformaggio, PhD.
Assistant Professor
School of Electrical and Computer Engineering

41

Advanced Topics in Malware Analysis

High Level Language Constructs In Assembly

Brendan Saltaformaggio, PhD
Assistant Professor
School of Electrical and Computer Engineering

Array of Structs, Data Structures, Oh My!

Georgia Tech

42

Recall The Structure Alignment

```
struct ch8_struct {   //Size     Minimum offset     Default offset
   int field1;        //  4            0                  0
   short field2;      //  2            4                  4
   char field3;       //  1            6                  6
   int field4;        //  4            7                  8
   double field5;     //  8           11                 16
};                 //Minimum total size: 19   Default size: 24
```

Source: Eagle, C. (2011). *The Ida Pro Book: The unofficial guide to the worlds most popular disassembler*. San Francisco: No Starch Press.

Georgia Tech

43

Getting tougher: Arrays of structs

```
int main() {
   int idx = 1;
   struct ch8_struct *heap_struct;
   heap_struct = (struct ch8_struct*)malloc(sizeof(struct ch8_struct) * 5);
❶  heap_struct[idx].field1 = 10;
}
```

```
.text:00401000 _main          proc near
.text:00401000
.text:00401000 idx            = dword ptr -8
.text:00401000 heap_struct    = dword ptr -4
.text:00401000
.text:00401000                push    ebp
.text:00401001                mov     ebp, esp
.text:00401003                sub     esp, 8
.text:00401006                mov     [ebp+idx], 1
.text:0040100D               ❷push    120             ; size_t
.text:0040100F                call    _malloc
.text:00401014                add     esp, 4
.text:00401017                mov     [ebp+heap_struct], eax
.text:0040101A                mov     eax, [ebp+idx]
.text:0040101D               ❸imul    eax, 24
.text:00401020                mov     ecx, [ebp+heap_struct]
.text:00401023               ❹mov     dword ptr [ecx+eax], 10
.text:0040102A                xor     eax, eax
.text:0040102C                mov     esp, ebp
.text:0040102E                pop     ebp
.text:0040102F                retn
.text:0040102F _main          endp
```

Source: Eagle, C. (2011). *The Ida Pro Book: The unofficial guide to the worlds most popular disassembler*. San Francisco: No Starch Press.

Georgia Tech

44

Reverse Engineering A Linked List

- Compiled with gcc on 64-bit Windows with light optimizations (i.e., -O1)
- Recall: Windows ABI says 1st arg in RCX!

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

void print(struct node *ptr) {
  while (ptr != NULL) {
    printf("%d ", ptr->data);
    ptr = ptr->next;
  }
  printf("\n");
}
```

```
print:
    push    rsi
    push    rbx
    sub rsp, 40
    mov rbx, rcx
    test    rcx, rcx
    je  .L7
    lea rsi, .LC0[rip] ; "%d "
.L8:
    mov edx, DWORD PTR 8[rbx]
    mov rcx, rsi
    call    printf
    mov rbx, QWORD PTR [rbx]
    test    rbx, rbx
    jne .L8
.L7:
    mov ecx, 10
    call    putchar
    add rsp, 40
    pop rbx
    pop rsi
    ret
```

Georgia Tech

Always More Complex Structures

- Trees
- Heaps
- Trees of Heaps of Stacks
- …
- C++ / object oriented languages
- Borland Delphi (!)
- Objective C is a counterexample because of rich runtime system
- Much more difficult
- Eventually, you can't identify a structure… just a collection of data blobs
- Pray for malware to be written in ASM

Georgia Tech

Brendan Saltaformaggio, PhD.
Assistant Professor
School of Electrical and Computer Engineering

47

Advanced Topics in Malware Analysis

High Level Language Constructs In Assembly

Brendan Saltaformaggio, PhD
Assistant Professor
School of Electrical and Computer Engineering

Importing Shared Function

Georgia Tech

48

Importing Shared Functions

- External code (i.e., libraries) can be compiled statically or dynamically
- Static is easy, you have the entire library function baked into the single binary
- Dynamic is harder, the loader needs to patch the function addresses at runtime
- Different executable formats follow different conventions

```
mov     rax, [rbp+var_10]
add     rax, 8
mov     rax, [rax]
mov     rsi, rax
mov     edi, offset format ; "Hello %s\n"
mov     eax, 0
call    _printf
```

```
; Segment type: Pure code
; Segment permissions: Read/Execute
_plt            segment para public 'CODE' use64
                assume cs:_plt
                ;org 4003F0h
                assume es:nothing, ss:nothing, ds:_data, fs:no
                dq 2 dup(?)

; int printf(const char *format, ...)
_printf         proc near               ; CODE XREF: main+2D↓p
                jmp     cs:off_601018
_printf         endp
```

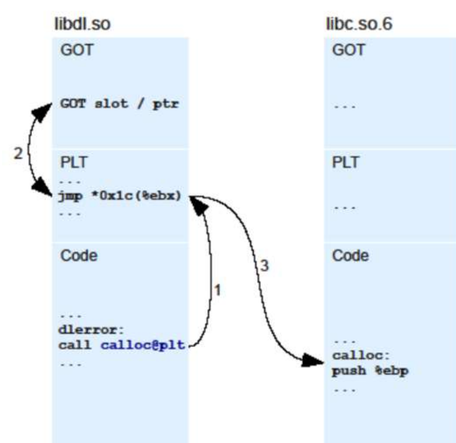
```
; ===========================================================

; Segment type: Pure data
; Segment permissions: Read/Write
; Segment alignment 'qword' can not be represented in assembly
_got_plt        segment para public 'DATA' use64
                assume cs:_got_plt
                ;org 601000h
 GLOBAL OFFSET TABLE  db   ? ;
off_601018      dq offset printf        ; DATA XREF: _printf↑r
off_601020      dq offset __libc_start_main
                                        ; DATA XREF: ___libc_start_main↑r
```

Georgia Tech

49

ELF Shared libraries (PLT + GOT)

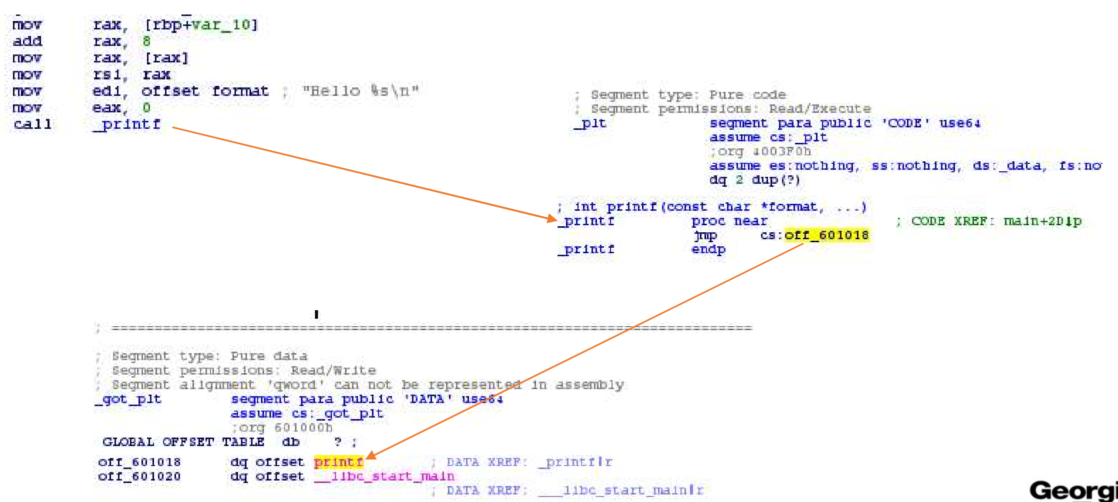
- Global Offset Table
  - GOT … not Game of Thrones ☹
  - Contains pointers (**data**) to symbols imported from other shared objects (libraries)
- Procedure Linkage Table
  - PLT
  - Contains **code** that transfers control through the GOT to a symbol in the shared object
- The entries in the GOT are initialized to point to a function in the loader to resolve symbols on-the-fly
  - Before first use, all GOT entries point to loader
  - During first call, the loader executes, loads the symbol (or loads the whole library), and patches the GOT address
  - All later calls go directly to shared object



Georgia Tech

50

Check it out in IDA



51

PE Shared libraries (IAT + .idata)

- The Windows loader is part of the OS Kernel
- The .idata section of the PE file contains
  - Import Lookup Table
    - Array of structures, each contains an ordinal or RVA of a name for each imported function
    - The ordinal represents the function’s position in the DLL’s Export Address Table
    - Structure indices are parallel to those in the Import Address Table (IAT)

- **Import Address Table** (IAT)
  - Also an array of the same structures
  - Initially both the Import Lookup Table and the IAT contain similar entries
  - The loader fills in the addresses of each imported routine in this table
- The Windows loader will proactively load some external DLLs but lazily link symbols
- Depends on how the DLL dependency was declared in the source code & how they were compiled

52

PE Shared library calls: Flavor 1 of 2

- Similar to ELF, all external symbol addresses are initialized to a loader function
  - In the case of Windows this is a helper function that hands control to the kernel
  - If the DLL has been proactively loaded, it just links
  - If not, it loads the DLL and then links
- If the original source code declared an included function as external
  - Then the compiler will optimize by just directly reading the address and calling it

```
.text:000000000040165B                 mov     r8d, 0
.text:000000000040165B                 mov     edx, 1          ; protocol
.text:0000000000401660                 mov     ecx, 2
.text:0000000000401665                 mov     rax, cs:__imp_socket
.text:000000000040166C                 call    rax ; __imp_socket

.idata:0000000000409544 ; SOCKET __stdcall socket(int af, int type, int protocol)
.idata:0000000000409544                 extrn __imp_socket:qword ; DATA XREF: main+105↑r
.idata:0000000000409544                                          ; socket↑r
```

Georgia Tech

53

PE Shared library calls: Flavor 2 of 2

- If the original source code **did not** declare an included function as external
  - Then the compiler will not know that it should optimize
  - Thus a stump function will be made in the .text section
    - No special section like ELF's PLT
  - All roads lead to the .idata section eventually

```
.text:00000000004015DA                 mov     rax, [rbp+640h+arg_8]
.text:00000000004015E1                 mov     rax, [rax]
.text:00000000004015E4                 mov     rdx, rax
.text:00000000004015E7                 lea     rcx, aYouCanChangeTh ; "[*] You can change this,
.text:00000000004015EE                 call    printf

.text:0000000000403240 ; =============== S U B R O U T I N E ===========================
.text:0000000000403240
.text:0000000000403240 ; Attributes: thunk
.text:0000000000403240
.text:0000000000403240 ; int printf(const char *Format, ...)
.text:0000000000403240                 public printf
.text:0000000000403240 printf          proc near               ; CODE XREF: main+6C↑p
.text:0000000000403240                                         ; main+8E↑p ...
.text:0000000000403240                 jmp     cs:__imp_printf
.text:0000000000403240 printf          endp
.text:0000000000403240

.idata:00000000004094A4 ; int printf(const char *Format, ...)
.idata:00000000004094A4                 extrn __imp_printf:qword ; DATA XREF: printf↑r
```

Georgia Tech

54

Excellent additional reading (Optional)

- Symantec blogger reverse engineered the loading and linking of external symbols on Linux and Windows!
- https://www.symantec.com/connect/articles/dynamic-linking-linux-and-windows-part-one
- https://www.symantec.com/connect/articles/dynamic-linking-linux-and-windows-part-two

55

Lesson Summary

- Interpret high level structures in assembler
- Identify function calls
- Reconstruct switch expressions to index table of targets
- Employ loops and compiler-induced elimination of branching
- Discover arrays
- Identify structures in assembly code

56