Introduction to Intel x86-64 Assembly, Architecture, Applications, & Alliteration

Xeno Kovah – 2014
xkovah at gmail
All materials is licensed under a Creative Commons “Share Alike” license.

- http://creativecommons.org/licenses/by-sa/3.0/

Attribution condition: You must indicate that derivative work

"Is derived from Xeno Kovah's 'Intro x86-64' class, available at http://OpenSecurityTraining.info/IntroX86-64.html"
//Journey to the center of memcpy
#include <stdio.h>

typedef struct mystruct{
    int var1;
    char var2[4];
} mystruct_t;

int main()
{
    mystruct_t a, b;
    a.var1 = 0xFF;
    memcpy(&b, &a, sizeof(mystruct_t));
    return 0xAce0Ba5e;
}

main:
sub    rsp,38h
mov    dword ptr [a],0FFh
mov    r8d,8
lea    rdx,[a]
lea    rcx,[b]
call   memcpy (0140001046h)
mov    eax,0ACE0BA5Eh
add    rsp,38h
ret

memcpy:
mov         r11,rcx ; rcx == &b
mov         r10,rdx ; rdx == &a
cmp         r8,10h ; r8 == sizeof(mystruct_t) == 8
jbe         mcpy00aa+95h (07FEEB9DA349h)
;It will take the jump because 0x8 is below or equal (JBE) 0x10
MoveBytes16:
mov         r10,r11 ; doesn't need to keep rdx copy anymore
MoveBytes16a:
lea         r9,[__mbctype_initialized (07FEEBC10000h)]
mov         rax,r8
mov         eax,dword ptr [r9+r8*4+4A363h]
add         rax,r9 ; the 4 preceding instructions are just
calculating based on the size (r8) and some lookup table,
where to jump next to continue
jmp         rax

It begins…
MoveSmall8: ; oh, well that's a convenient name…
            mov rax,qword ptr [rdx]
            mov qword ptr [r10],rax ; bam, 8 byte copy and done!
            mov rax,r11
            ret ; done already? But I just got here!

; So that was all fairly un-interesting...And we didn't find any new
instructions. So let's go back and change the size of our struct
so that we don't take that initial JBE and see what happens on
the other path…

typedef struct mystruct{
    int var1;
    char var2[4];
} mystruct_t;

typedef struct mystruct{
    int var1;
    char var2[16];
} mystruct_t;
memcpy:
  mov    r11,rcx ; rcx == &b == destination
  mov    r10,rdx ; rdx == &a == source
  cmp    r8,10h ; r8 == sizeof(mystruct_t) == 0x14
  jbe    mcpy00aa+95h (07FEEB9DA349h)
;This time it will NOT take the jump because 0x16 is not below or
  equal (JBE) 0x10. So it falls through to…
  sub    rdx,rcx
  jae    mcpy00aa (07FEEDE0A2B4h) ; if the copy destination
  is above (unsigned) compare or equal to the source, then we can
  skip the next check. In our case it happens to not be
  mov    rax,r10 ; copy the start address of the src
  add    rax,r8  ; calculate the last byte of the src to be copied
  cmp    rcx,rax ; check if the dst's start address is less than
  the last byte of the src (meaning they overlap)
  jl     MoveSmall+297h (07FEEDE0A5FAh)
mcpy00aa:
mcpy00aa:
bt       dword ptr [__favor (07FEEDF93408h)],1 ; check some bit
that we have no idea what it is (but probably a configuration bit)
jae      mcpy00aa+1Dh (07FEDE0A2D1h) ; if it's set, jmp
; in our case it seems not to be set, so we fall through
push     rdi ; save rdi (because it's going to be used)
push     rsi
mov      rdi,rcx ; move dst into rdi
mov      rsi,r10 ; move src into rsi
mov      rcx,r8  ; move size into dcx
rep movs   byte ptr [rdi],byte ptr [rsi] ; that which we seek!
pop      rsi ; restore
pop      rdi ; restore
mov      rax,r11 ; set return value to the copy of dst
ret

So, what's the deal with "rep movs"?

REP MOVVS
Repeat Move Data String to String

- MOVVS is one of number of instructions that can have the “rep” prefix added to it, which repeat a single instruction multiple times.
- All rep operations use *cx register as a “counter” to determine how many times to loop through the instruction. Each time it executes, it decrements *cx. Once *cx == 0, it continues to the next instruction.
- Either stores 1, 2, 4, or 8 bytes at a time
- Either fill 1 byte at [di] with [si] or fill 2/4/8 bytes at [*di] with [*si].
- Moves the *di register forward 1/2/4/8 bytes at a time, so that the repeated store operation is storing into consecutive locations.
- So there are 3 pieces which must happen before the actual rep stos occurs: set *di to the starting destination, *si to the starting source, and *cx to the number of times to store
- Note: Unlike MOV, MOVVS can move memory to memory…but only between [*si] and [*di]
- A lot of people don’t pay attention to the fact that it’s REP MOVVS, not REP MOV (even though you may say it like “rep move”)

Book p. 274 & 278
High level pseudo-code approximation
(how interesting… it's like I went in reverse of the normal software engineering process…)

TODO: fixme

memcpy(void * dst, void * src, unsigned int len){
  if(len <= 0x10){
    //sequence of individual mov instructions
    //as appropriate for the size to be copied
  }
  else{
    if(dst & 3 != 0){
      //Other path we didn’t take, @ 1026ED74
    }
    if((len / 4) >= 8){
      ecx = len / 4;
      rep movs dword dst, src;
    }
    else{
      //sequence of individual mov instructions
      //as appropriate for the size to be copied
    }
  }
Instructions we now know (30)

- NOP
- PUSH/POP
- CALL/RET
- MOV
- ADD/SUB
- IMUL
- MOVZ/X/MOV/SX
- LEA
- JMP/Jcc (family)
- CMP/TEST
- AND/OR/XOR/NOT
- INC/DEC
- SHR/SHL/SAR/SAL
- DIV/IDIV
- REP STOS
- REP MOV/S