ICS Quiz 3

Fall, 2019

**Suppose all the following codes are running on a little-ending x86-64 machine.*

1. Procedure Call (10')

One of the ICS TA wrote a simple C program and some assembly code is provided. Please read the code and answer the following questions.

```c
int foo (int x, int y) {
      return x>y? x: y;
}

int bar (int *arr, int start, int end) {
      if (start == end)
            return arr[start];
      return foo(arr[start], bar(arr, start + 1, end));
}
```

The assmebly code of the `foo&bar` functions is shown below. Answer the following questions.

```
foo:
    pushq  %rbp
    movq   %rsp, %rbp
    movl   %edi, -4(%rbp)
    movl   %esi, -8(%rbp)
    movl   -4(%rbp), %eax
    cmpl   %eax, -8(%rbp)
    __(1)__ -8(%rbp), %eax
    popq   %rbp
    ret
bar:
    pushq  %rbp
    movq   %rsp, %rbp
    subq   $16, %rsp
    movq   %rdi, -8(%rbp)
    movl   %esi, -12(%rbp)
    movl   %edx, -16(%rbp)
    movl   -12(%rbp), %eax
    cmpl   -16(%rbp), %eax
    jne    .L4
    movl   -12(%rbp), %eax
    cltq
    leaq   __(2)__ %rdx
    movq   -8(%rbp), %rax
    addq   %rdx, %rax
    movl   (%rax), %eax
    jmp    .L5
.L4:
    movl   -12(%rbp), %eax
    leal   __(3)__, %ecx
    movl   __(4)__, %edx
    movq   -8(%rbp), %rax
    movl   %ecx, %esi
    movq   %rax, %rdi
    call   bar
    movl   %eax, %ecx
    movl   __(5)__, %eax
    cltq
    leaq   __(6)__, %rdx
    movq   __(7)__, %rax
    addq   %rdx, %rax
    movl   (%rax), %eax
    movl   __(8)__, %esi
    movl   __(9)__, %edi
    call   foo
.L5:
    leave
    ret
```

1. Please explain what function `bar` does.

2. Fill the blanks in the assembly code.

2. Floating Point (22’)

Consider a 16-bit floating point representation based on the IEEE floating-point format, with 1 sign bit, 9 exp bits, 6 frac bits. Assume we use the IEEE round-to-even mode to do the approximation.

1. Fill in the table below. You are supposed to represent the target floating point value based on the designed format (in **hexadecimal** form) or write down the value (in the form `x` or $x*2^y$ where both `x` and `y` are integers) represented by the given hexadecimal. (2’*6=12’)

Description	Value	Representation
Largest denormalized value		
Smallest normalized value		
--	$-1*2^{-256}$	
--		0xE02C

2. Use the floating point format to calculate the addition: $(0\ 100001100\ 111111)_2$+$(0\ 100001110\ 111111)_2$. Please write your answer in **hexadecimal** form. (4’)

3. Buffer Overflow Attack (30’)

Suppose we have a simple function `func` as below, and `getbuf` uses the `gets` functions in section 3.10.3 on CSAPP. The ASCII number of ‘`0`’ and ‘\n’ is 48 and 0x0a.

```
void func(long txt){
      char *s = (void *) &txt;
      for (int i = 0; i < 8; i++) putchar(s[i]);
      exit(0);
}
```

```
0000000000401213 <getbuf>:
  401213:      55                        push   %rbp
  401214:      48 89 e5                  mov    %rsp,%rbp
  401217:      48 83 ec 10               sub    $0x10,%rsp
  40121b:      48 8d 45 f8               lea    -0x8(%rbp),%rax
  40121f:      48 89 c7                  mov    %rax,%rdi
  401222:      e8 3b ff ff ff            callq  401162 <Gets>
  401227:      b8 01 00 00 00            mov    $0x1,%eax
  40122c:      c9                        leaveq
  40122d:      c3                        retq

000000000040122e <main>:
  40122e:      55                        push   %rbp
  40122f:      48 89 e5                  mov    %rsp,%rbp
  401232:      48 83 ec 10               sub    $0x10,%rsp
  401236:      b8 00 00 00 00            mov    $0x0,%eax
  40123b:      e8 d3 ff ff ff            callq  401213 <getbuf>
  401240:      89 45 fc                  mov    %eax,-0x4(%rbp)
  401243:      bf 48 69 00 00            mov    $0x6948,%edi
  401248:      e8 74 ff ff ff            callq  4011c1 <func>
```

```
21.    40124d:     8b 45 fc                  mov    -0x4(%rbp),%eax
22.    401250:     89 c7                     mov    %eax,%edi
23.    401252:     e8 19 fe ff ff            callq  401070 <exit@plt>
```

1. What is the return address when executing the `retq` in line 10 when we type “`1234567812345678 9`” ? (6)

2. Suppose the register `%rdi` in line 7 is `0x64ffffc0`. If we want to print “`Hacked`” when executing `func`, what we should input in hex? (12)

The machine code of the operation “`mov n(%rsp), %rdi`” is “`48 8b 7c 24 xx`”, where `xx` is 8-bit 2's complement of n. And the ASCII code of “Hacked” is `48 61 63 6b 65 64` and has already been filled in the table.

0x48	0x61	0x63	0x6b	0x65	0x64	0x00	0x00

3. What if the `%rdi` in line 7 is `0x64ff0af8`. What’s the differences? How to achieve the same goal in Q2? (12)

0x48	0x61	0x63	0x6b	0x65	0x64	0x00	0x00

4. Data Structures (38’)

Please the code and answer the following questions.

```
#include <stdio.h>

union ics_u {
       short **spp;
       char ca[2][3];
       char (*cpa)[3][2];
       struct {
              short s1;
              short *ps[2];
              char ca[3];
              union {
                     char c1;
                     unsigned *pi[2];
              } u;
              short s2;
              int (*p[2]) (long arg1, int arg2, short arg3, char** arg4, float
arg5);
              char c2;
       } str[2];
};


int main () {
       union ics_u data;
       data.spp = data.ca;
       data.cpa = data.ca;
```

```
        printf (“size [1] : 0x%lx\n”, sizeof (data.str[0].u));
        printf (“size [2] : 0x%lx\n”, sizeof (data.str));
        printf (“size [3] : 0x%lx\n”, sizeof (data));
        printf (“size [4] : 0x%lx\n”, sizeof (data.spp));
        printf (“size [5] : 0x%lx\n”, sizeof (data.ca));
        printf (“size [6] : 0x%lx\n”, sizeof (data.cpa));
        printf (“&data : %p\n”, &data);
        printf (“value [1] : %p\n”, data.spp+1);
        printf (“value [2] : %p\n”, ((unsigned *)(&(data. ca[0][1])))+1);
        printf (“value [3] : %p\n”, &((*(data.cpa))[1]));
        printf (“value [4] : %p\n”, &(data.str[1]));
        printf (“value [5] : %p\n”, &(data.str[1].s1));
        printf (“value [6] : %p\n”, &(data.str[1].ps[1]));
        printf (“value [7] : %p\n”, &(data.str[1].ca));
        printf (“value [8] : %p\n”, &(data.str[1].u.c1));
        printf (“value [9] : %p\n”, &(data.str[1].u.pi));
        printf (“value [10] : %p\n”, &(data.str[1].s2));
        printf (“value [11] : %p\n”, &(data.str[1].p));
        printf (“value [12] : %p\n”, &(data.str[1].c2));
        return 0;
}
```

1. Please complete the output below. (6 * 1 + 12 * 2 = 30’)

size [1] :
size [2] :
size [3] :
size [4] :
size [5] :
size [6] :
&data: 0x7fff5e60fcc0
value [1] :
value [2] :
value [3] :
value [4] :
value [5] :
value [6] :
value [7] :
value [8] :
value [9] :
value [10] :
value [11] :
value [12] :

2. How many bytes are wasted in data? (5’)

3. If you can rearrange the declarations in the struct and union, how many bytes of memory can you save in data compared to the original declaration? Please write down your rearranged declaration. (3’)