# CSPB 2400 - Park - Computer Systems



| | |
|---|---|
| **Started on** | Sunday, 18 February 2024, 5:03 PM |
| **State** | Finished |
| **Completed on** | Sunday, 18 February 2024, 5:22 PM |
| **Time taken** | 18 mins 34 secs |
| **Marks** | 13.00/15.00 |
| **Grade** | **8.67** out of 10.00 (**87**%) |

Question **1**

Correct

Mark 2.00 out of 2.00

Assume common data sizes (char = 1 byte, short = 2, int = 4, long = 8, float = 4, double = 8) and that alignment requirements follow the data size.

```
struct {
  char c[ 3 ];
  int i[ 4 ];
  double d;
} datum;
```

What is the offset of c[ 0 ] relative to &datum? 0

Your last answer was interpreted as follows: 0

Correct answer, well done.

What is the offset of i[ 2 ] relative to &datum? 12

Your last answer was interpreted as follows: 12

Correct answer, well done.

What is the offset of d relative to &datum? 24

Your last answer was interpreted as follows: 24

Correct answer, well done.

Question **2**

Partially correct

Mark 1.00 out of 3.00

Assume common data sizes (char = 1 byte, short = 2, int = 4, long = 8, float = 4, double = 8) and that alignment requirements follow the data size.

```
struct {
  int i[ 2 ];
  char c[ 4 ];
  double d;
} datum;
```

What is the offset of i[ 1 ] relative to &datum? 4

Your last answer was interpreted as follows: 4

Correct answer, well done.

What is the offset of c[ 3 ] relative to &datum? 15

Your last answer was interpreted as follows: 15

Incorrect answer.

What is the offset of d relative to &datum? 24

Your last answer was interpreted as follows: 24

Incorrect answer.

Question **3**

Correct

Mark 5.00 out of 5.00

Assume common data sizes (char = 1 byte, short = 2, int = 4, long = 8, float = 4, double = 8) and that alignment requirements follow the data size.

```
struct {
  char c[ 3 ];
  int i[ 6 ];
  double d[ 4 ];
} datum[ 3 ];
```

What is the offset of datum[ 0 ].c[ 0 ] relative to &datum? 0

Your last answer was interpreted as follows: 0

Correct answer, well done.

What is the offset of datum[ 0 ].i[ 3 ] relative to &datum? 16

Your last answer was interpreted as follows: 16

Correct answer, well done.

What is the offset of datum[ 0 ].d[ 2 ] relative to &datum? 48

Your last answer was interpreted as follows: 48

Correct answer, well done.

Question **4**

Correct

Mark 5.00 out of 5.00

Consider the functions below, with both C code and compiled assembly provided. Recall that `gets(buf)` simply copies input in to `buf`. The inintial values of `rsp`, `rbp`, and `rip` are provided. The initial value of `rip` tells you the first instruction which will start to execute: the `push` at the start of `test\_func`. You can assume that the `leaveq` instruction is equivalent to `movq $rbp, $rsp` followed by `popq $rbp`.

Initial State

```
%rip  =  0x40002e
%rbp  =  0xff0088
%rsp  =  0xff0068
```

```
void test_func(){
    int localArr[4] = {0xaddedfee,
                       0xfadedace,
                       0xcedeface,
                       0xabaddeed};
    get_buffer();
    printf("%x\n", localArr[0]);
}


void get_buffer(){
    char buf[8];
    gets(buf);
}


void magic(){
    printf("+1\n");
}
```

```
000000000040002e <test_func>:
  40002e:   push   %rbp
  40002f:   mov    %rsp,%rbp
  400032:   sub    $0x10,%rsp
  400036:   movl   $0xaddedfee,-0x10(%rbp)
  40003d:   movl   $0xfadedace,-0xc(%rbp)
  400044:   movl   $0xcedeface,-0x8(%rbp)
  40004b:   movl   $0xabaddeed,-0x4(%rbp)
  400052:   lea    -0x10(%rbp),%rdi
  400056:   callq  400016 <get_buffer>
  40005b:   mov    -0x10(%rbp),%edx
  40005e:   mov    $0x400794,%esi
  40006d:   callq  4004f0 <printf@plt>
  400072:   leaveq
  400073:   retq

0000000000400016 <get_buffer>:
  400016:   push   %rbp
  400017:   mov    %rsp,%rbp
  40001a:   sub    $0x08,%rsp
  40001e:   lea    -0x08(%rbp),%rdi
  400022:   mov    $0x0,%eax
  400027:   callq  4000e0 <gets@plt>
  40002c:   leaveq
  40002d:   retq

0000000000400074 <magic>:
  400074:   push   %rbp
  400075:   mov    %rsp,%rbp
  400078:   mov    $0x400798,%edi
  40007d:   callq  4004c0 <printf@plt>
  400082:   pop    %rbp
  400083:   retq
```

(a) What is the fewest number of characters you could enter to `gets`, that would cause this program to behave anomalously (ie, to experience buffer overflow)? Remember, `gets` will always add a `null` character immediately after the last character you enter, to mark the end of the string.

8 ✔

(b) Suppose the program executes until `%rip` is `0x400027`; immediately before the call to `gets`. What is the hexadecimal-formatted four-byte value at address `0xff005c`? Specifically, what would `p/wx *0xff005c` print? Remember, `%rbp` is the base of a function's stack frame.

0xabaddeed ✔

(c) What is the fewest number of characters you could enter to `gets`, that would cause this program to execute anomalous instructions (ie, set `%rip` to a value it would not otherwise have)? Remember, `gets` will always add a `null` character immediately after the last character you enter, to mark the end of the string.

16 ✔

(d) Suppose you want to construct an input to make this program print ``+1", as is done by the `magic` function. The input below includes the null character `gets` will add, and is nearly complete; fill in the single blank character.

| **ASCII** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ?? | \0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **HEX** | 0x30 | 0x30 | 0x30 | 0x30 | 0x30 | 0x30 | 0x30 | 0x30 | 0x30 | 0x30 | 0x30 | 0x30 | 0x30 | 0x30 | 0x30 | 0x30 | 0x74 ✔ | 0x00 |

(e) Suppose you want to construct an input to make this program print `0xcedeface`, instead of `0xaddedfee`. The input below includes the null character `gets` will add, and is nearly complete; fill in the single blank character. Remember, `%rbp` is the base of a function's stack frame.

| **ASCII** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ?? | \0 |
|---|---|---|---|---|---|---|---|---|---|---|
| **HEX** | 0x30 | 0x30 | 0x30 | 0x30 | 0x30 | 0x30 | 0x30 | 0x30 | 0x68 ✔ | 0x00 |