Secure Programming Assignment 1 : RSA

Brian Koh Lit Yang a1782291 (UG)

Part 1

Question 1

- Your Sunday submission will be marked with no penalty.

Question 2

- According to University's policies extensions are not granted for weddings.

Question 3

- A typed up pdf

Part 2

Question 4

What does the macro O() do?

- In this code, it is a fragment of code that is given the name "O". In this case, the defined macro O is a function that is not used anywhere else in the code. Which makes it redundant.
- the pre-defined macro O takes in two values, type and field.
- the function allocates a specific amount based on the two values taken, type and field.
- the function is casted to the type and the address of the field, which is then casted to size_t. which is an unsigned int.
- the function should return the number of bytes after which the element "field" is placed in struct type.

Question 5

As both structs have the identical fields, one would expected their sizes to be identical. Yet, as you can see, the sizes are different. Please explain why. You may want to use the macro O() for investigating this.

- this is because the types of the struct are different.
- the macro function O( ) casts to the struct type and the field, which both are different.
- This directly affects the amount that is casted to size_t, which dictates the final output of the macro function O().

Question 6

Reorder the field to get the smallest and the largest possible structures with the same fields.

- re-arranging the structs in descending order would be the best option and would result in the least amount of wasted memory space.
- 64, 32, 16, 8
    - pointers are accepted as 32 bits
- re-arranging the structs in ascending order would have a higher chance of memory wastage.

```
struct bysize
{
        int8_t int8;            // 1
        uint8_t uint8;          // 1
        int16_t int16;          // 2
        uint16_t uint16;        // 2
        int32_t int32;          // 4
        uint32_t uint32;        // 4
        int64_t int64;          // 8
        uint64_t uint64;        // 8
        intptr_t intptr;        // 8
        uintptr_t uintptr;      // 8
};
```

- the least possible value that outputs from the struct is 48.
- the above code is the best possible outcome for the least memory allocation.

```
struct bysize
{
        int8_t int8;            // 1
        int64_t int64;          // 8
        uint8_t uint8;          // 1
        uint64_t uint64;        // 8
        int16_t int16;          // 2
        intptr_t intptr;        // 8
        uint16_t uint16;        // 2
        uintptr_t uintptr;      // 8
        int32_t int32;          // 4
        uint32_t uint32;        // 4
};
```

- the highest possible value is 72 bytes.
    - Due to padding in memory
    - Since memory is stored contiguously in 8 bytes
    - If 1 byte is allocated first, there will be a padding of 7 bytes after it before the next allocation is made.
    - The above code is the best possible scenario for the most amount of memory allocation.

Question 7

How much space is wasted in each instance of bysize?

- 2 bytes of space is wasted in each instance of bysize
- the first 6 bytes are allocated and then two bytes of padding is required.
- then the rest are properly allocated into 8 bytes.