CSE 351 Section 4 – Lab 2 Prep (x86-64, GDB)

Exercise 1: x86 to C

Write an equivalent C function for the following x86 code:

```
mystery:
   leal    (%rdx,%rdx,2), %eax
   addl    %eax, (%rdi)
   movslq  %esi, %rsi
   leaq    (%rdi,%rsi,4), %rax
   ret
```

```
int* mystery(int* a, int b, int c) {
    *a += 3*c;
    return a+b;
}
```

- `%rdx` holds c (3rd arg). While unintuitive, the "`l`" instruction suffix in **`leal`** hints that the type of c is an **`int`**: memory operand requires 8-byte register names, but the result (`%rdx+%rdx*2=3*c` into `%eax`) should match the range of the input.
- `%rdi` holds a (1st arg). We know it's a pointer because it is dereferenced in the **`addl`** instruction. The "`l`" instruction suffix tells us it points to an **`int`**. This instruction adds the value in `%eax` to the value that `%rdi` points to. We can write this in C as `*a += 3*c`.
- `%esi` holds b (2nd arg). The use of this register name as the source operand to **`movslq`** (using "`l`") tells us it is an **`int`**. This extension instruction is setup for the next instruction, which requires 8-byte register names in its memory operand.
- The **`leaq`** instruction stores the result of `%rdi+rsi*4`, which is equivalent to the value of `a+b*4`. However, since a is an **`int*`**, we can write this in C as `a+b` because of pointer arithmetic.
- By convention, `%rax` holds the return value of a function. When this function returns, `%rax` contains the result of `a+b`.

Exercise 2: Avoid `explode_bomb`

Given the following x86-64 code, determine what input argument(s) avoid the call to `explode_bomb`:

```
401152 <decision>:
  401152: cmp    $0x5,%edi
  401155: je     401167 <decision+0x15>
  401157: mov    $0x40200a,%edi            (0x40220a points to the
  40115c: mov    $0x0,%eax                 first argument of printf)
  401161: call   401030 <printf@plt>
  401166: ret
  401167: call   401136 <explode_bomb>
  40116c: jmp    401166 <decision+0x14>
```

The **`cmp`** instruction sets the flags register based on the result of `%edi-5`. Then, **`je`** will jump if this result is equal to `0`. This means that the code will take the jump to **`call`** `explode_bomb` if `%edi== 5`. Otherwise, it prints the string stored at `0x40220a` and returns.

So we want: `%edi != 5`