Hacking in C

Assignment 3, Thursday, March 1, 2018

Handing in your answers: Submission via Blackboard (http://blackboard.ru.nl)

Deadline: Thursday, March 8, 23:59 (midnight)

1. Recall from the lecture that there is no default initialization on the stack. There is also no cleanup, so by reading memory below the current stack frame directly before and after a function call, you can learn things about that function.

   Consider the following code snippet:

   ```
   int main (void)
   {
     ...
     magic_function();
     ...
   }
   ```

   Write this snippet to a file called `exercise1.c`. Complete the program such that it prints the amount of bytes of stack space used by `magic_function`.
   Hint 1: You do not know anything about `magic_function`, except that it does not receive any arguments and you do not use its return value.
   Hint 2: You should try with some own implementations of `magic_function`. However, compilers are smart. Due to optimizations your function might end up using no stack space at all. To prevent this:

   - make sure your `magic_function` does something meaningful with its local variables (e.g. add them, then return the result), and
   - implement your `magic_function` in a separate source file, and compile with separate compilation and linking steps. E.g:

     ```
     $ gcc -c -o magic_function.o magic_function.c
     $ gcc -o exercise1 exercise1.c magic_function.o
     ```

   When grading, we will use your program with our own implementations of `magic_function`.
   Hint 3: You may assume that `magic_function` does not use more than 4 MB (4194304 bytes) of stack space.
   Hint 4: You may need to compile with compiler option `-fno-stack-protector`.

2. This exercise is about the size of heap space available to a program.

   (a) Write a program (in a file called `exercise2.c`), which determines the maximal amount of heap space that can be allocated in one call to `malloc`. The output of the program should be of the following form (where `XXX` is replaced by the correct number):

   ```
   One malloc can allocate at most XXX bytes.
   ```

   (b) Is the output of the program always the same? Explain why. Write your answer to a file called `exercise2b`.

3. Consider the following program:

   ```
   int main() {
     int32_t x[4];
     x[0] = 23;
     x[1] = 42;
     x[2] = 5;
     x[3] = (1<<7);
   ```

```
  printf("%p\n", x);
  printf("%p\n", &x);          // (a)
  printf("%p\n", x+1);         // (b)
  printf("%p\n", &x+1);        // (c)
  printf("%d\n", *x);          // (d)
  printf("%d\n", *x+x[2]);     // (e)
  printf("%d\n", *x+*(x+3));   // (f)
  return 0;
}
```

Assume that the first call to printf prints `0x7fffb3cc3b20`. What do the other 6 calls to `printf` print? **Explain your answers.** Write your answer to a file called `exercise3`.

4. Place the files

   - `exercise1.c`,
   - `exercise2.c`,
   - `exercise2b`, and
   - `exercise3`

   in a directory called `sws1-assignment3-STUDENTNUMBER1-STUDENTNUMBER2` directory (as in the previous assignments, replace `STUDENTNUMBER1` and `STUDENTNUMBER2` by your respective student numbers). Make a `tar.gz` archive of this directory and submit the archive in Blackboard.