Software and Web-Security

Assignment 2, version 2, Monday, February 17, 2014

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

Deadline: Tuesday, February 25, 24:00 (midnight)

Marks: You can score a total of 100 points.

1. You are given the following code fragment:

```
int main (void)
{
  short i = 0x1234;
  char x = -127;
  long sn1 = <STUDENT NUMBER OF TEAM MEMBER 1, WITHOUT LEADING S>;
  long sn2 = <STUDENT NUMBER OF TEAM MEMBER 2, WITHOUT LEADING S>;
  int[2] y = {0x11223344,0x44332211};

  ...

}
```

   (a) Write this code snippet to a file called `exercise1.c`.

   (b) Set the values of `sn1` and `sn2` to your student numbers.

   (c) Replace the `...` by code that prints the size in bytes of each of the local variables.

   (d) Extend the functionality of the program to print the memory layout of the local variables, in a byte-by-byte fashion, so a four-byte integer becomes four lines. More specifically, your program should print a table of the following form (addresses and data are fictional):

```
address    content (hex)     content (dec)
------------------------------------------
0x...00    0xFF              255
0x...01    0x12               18
0x...02    ...               ...
```

   You do not have to sort the output.

   (e) Compile your program with `gcc -O3 -Wall` and run the program. Write the output of the program to a file called `exercise1.out`. Explain which variable is stored at which location in memory and write this explanation to a file called `exercise1.exp`.

2. Since the C99 standard, the C programming language has a `bool` data type. Programs that use this data type have to include the file `stdbool.h`. They have to be compiled with the compiler flag `-std=c99`. Write a program (in a file called `exercise2.c`), which finds out about the internal representation of bool. Specifically, your program shall print the following:

   - How many bytes does a `bool` use?
   - What hexadecimal representation does a `bool` have, if you set it to `true`?
   - What hexadecimal representation does a `bool` have, if you set it to `false`?
   - Can you assign other hexadecimal values than these two to a `bool` variable? Are those interpreted as `true` or as `false` or do they cause an error?

3. 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 `exercise3.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 exercise3 exercise3.c magic_function.o
     ```

   If it still looks as though your test function uses no stack at all, drop by (HG02.066) or send an e-mail with a problem description and your code as an attachment to pol.vanaubel@cs.ru.nl
   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.

4. Place the files `exercise1.c`, `exercise1.out`, `exercise1.exp`, `exercise2.c`, and `exercise3.c` in a directory called `sws1-assignment2-STUDENTNUMBER1-STUDENTNUMBER2` (again, replace `STUDENTNUMBER1` and `STUDENTNUMBER2` by your respective student numbers). Write a `Makefile` that (with a single invocation of `make` in the `sws1-assignment2-STUDENTNUMBER1-STUDENTNUMBER2` directory) builds programs `exercise1` (from `exercise1.c`), `exercise2` (from `exercise2.c`), and `exercise3` (from `exercise3.c`). Make sure that this `Makefile` is also in the `sws1-assignment2-STUDENTNUMBER1-STUDENTNUMBER2` directory.
   Make a `tar.gz` archive of the whole `sws1-assignment2-STUDENTNUMBER1-STUDENTNUMBER2` directory and submit this archive in Blackboard.