Software and Web-Security
Assignment 2, Monday, February 9, 2015

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

Deadline: Monday, February 23, 24:00 (midnight)

1. Unix and Linux sytems use special files in the `/dev` directory to handle access to devices. Two such special *device files* provide a source of random numbers. These files are `/dev/random` and `/dev/urandom`.

   (a) Find out what the conceptual difference between these two files is. Write your answer to a text file named `exercise1a.txt`.

   (b) Write a program in a file called `exercise1b.c` that opens the file `/dev/urandom` for reading and then performs a loop which
   - reads one byte from `/dev/urandom`;
   - prints one line consisting of the value of this byte as signed decimal integer, as unsigned decimal integer, and in hexadecimal notation (seperated by space);
   - exits (from the loop) if the value of the byte is 42.

   The last line of output from the program should thus be

   ```
   42 42 2a
   ```

   (c) Run the program and write the output to a file called `exercise1c`.

   (d) Write another program called `exercise1d.c`, which does the same as `exercise1b.c`, except for the following:
   - Use 16-bit unsigned integers instead of bytes (datatype `uint16_t`, you need to include the file `stdint.h`).
   - In the loop, initialize the 16-bit unsigned integer with two random bytes (16 bits) from `/dev/urandom`.
   - In the loop, print one line containing the value of hte 16-bit unsigned integer as fixed-width 4-character hexadecimal value (padded at the front with leading zeros).
   - Again, terminate the loop if the value is 42, the last line of output is thus

     ```
     002a
     ```
   - Run the program 10 times and each time count the number of output lines. Write these counts to a text file called `exercise1d.txt`.

   (e) Write a brief description of how you obtained the line counts in part d) in a text file called `exercise1e.txt`.

2. 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 `exercise2.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 `exercise2.out`. Explain which variable is stored at which location in memory and write this explanation to a file called `exercise2.exp`.

3. 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 `exercise3.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?

4. Place the files

   - `exercise1a.txt`,
   - `exercise1b.c`,
   - `exercise1c`,
   - `exercise1d.c`,
   - `exercise1d.txt`,
   - `exercise1e.txt`,
   - `exercise2.c`,
   - `exercise2.out`,
   - `exercise2.exp`, 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

   - `exercise1b` (from `exercise1b.c`),
   - `exercise1d` (from `exercise1d.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.