ECE-458 (Spring 2023) – Assignment 2

Due on 2023-07-11, 23:00 (Tuesday, 11pm)

Instructions

You are allowed to work with your classmates and discuss ideas and solutions; however, the submitted work MUST BE STRICTLY YOUR OWN WRITING. This also applies to looking up ideas on the Internet or textbooks. Moreover, any collaboration MUST be acknowledged in the submission (explicitly name the persons that helped you or that you helped, persons that shared ideas with you or that you shared ideas with them; explicitly name online sources or textbooks where you found ideas that helped you write your solution, etc.).

The standard advice on what to do about collaborations to avoid getting in trouble is: you may discuss and share ideas with classmates, but then, put aside any annotations and DO NOT LOOK at them when writing your own solution (this ensures that your writing is original and distinct). Also, NEVER share any of your written solutions or your written code with anyone; preferably, DO NOT ALLOW ANYONE to look at your written solutions (keep in mind that if they have good memory and write the exact text in your solutions and submit it, you will be equally liable in the academic dishonesty incident — see Course Outline).

It is assumed that you have carefully read the additional conditions included in the Course Outline (available through LEARN) regarding academic integrity, assignments submissions, etc.

Timing Attack on Passwords

You will implement a timing attack on a password validation facility to determine the password of a target user, as described in class (slides set on side-channel analysis – timing). The password will contain exclusively lowercase letters from the English alphabet.

To keep things simple and under control,[1] the password validation function will be embedded in your program, and you will hardcode a particular password. You don't need to specify a username. When your T.A. grades your submission, they will change the password and try — your program should not be optimized or in any way work based on assumptions about the particular password that you hardcoded.

The password validation function could simply be a one-liner `return strcmp(pwd,"pwd123") == 0;` (or `pwd == "pwd123"` in C++). However, with recent compilers, the string comparison implementation may actually be resilient to timing attacks, so you may want to code your own string comparison for the password validation. For example:

```
bool check_password (const char * pwd)
{
    const char * c = "pwd123";
    while (*pwd == *c && *pwd != '\0' && *c != '\0')
    {
        ++pwd; ++c;
    }
    return *pwd == *c && *pwd == '\0';
}
```

Based on the time between calling the password validation function and returning from it, your

[1] 140+ students working on our common `eceubuntu` server and measuring execution timing makes things complicated!

program will determine (will "crack") the password (notice that you should measure the time until the response from the function arrives, before even looking at the response to see if it is true or false). In addition to determining each character of the password, the program should inform the user once the password has been determined (once the password validation function returns `true` for the attempted password.

Although you can measure the execution time with sub-nanosecond resolution (this is explained below), there is still "noise" in the measurement (timing variation due to hardware aspects such as cache, pipelines, dynamic clock frequency, etc., or to software aspects such as task switching beyond your program's control). Thus, you will need to measure multiple times to "average out" this measurement noise. In particular, you will need to run password attempts multiple times for each attempted password and compute statistics. For all attempted characters, you should keep the mean and confidence intervals (e.g., 75% or 95% confidence intervals), and as soon as one character's interval is above all other characters' intervals (without overlapping), then the program (or the user) decides that this is the character in the password at the corresponding position.

Notice that, like in assignment 1, you are free to write a program that does everything fully automated or just write a program or programs that display the statistics and any findings and the user makes decisions based on it.

To measure time with high resolution, you may use the CPU clock clycle counter. In C/C++ with the gcc compiler, you can use the following function (requires `#include <inttypes.h>`, in either C or C++).

```
static __inline__ uint64_t rdtsc()
{
    uint32_t hi, lo;
    __asm__ __volatile__ ("rdtsc" : "=a"(lo), "=d"(hi));
    return  ((uint64_t)lo) | (((uint64_t)hi) << 32);
}
```

To keep track of mean and variance on the fly (i.e., with O(1) storage), you can just keep a running total for the sum of the values, one for the sum of the square of the values, and one counter (please watch out for arithmetic overflow!). At any particular point in time, when you have N measurements, assuming that you have variables `N` for the counter, `sum_t` for the sum of the values and `sum_tt` for the sum of the squares, you take advantage of the following property:

$$\begin{aligned}
\hat{\mu} &= \frac{\text{sum\_t}}{N} \\
\hat{\sigma}^2 &= \frac{1}{N-1}\sum_k (t_k - \hat{\mu})^2 \\
&= \frac{1}{N-1}\left(\sum_k t_k^2 - 2\cdot\hat{\mu}\sum_k t_k + N\hat{\mu}^2\right) \\
&= \frac{1}{N-1}\left(\text{sum\_tt} - N\hat{\mu}^2\right)
\end{aligned}$$

For this assignment, you must write your programs in C or C++, and they must run on a Linux system with gcc/g++ (you may login into `eceubuntu` and try it).

Deliverables

You should submit source code and preferably also a Makefile (unless it is straightforward to compile; e.g., if your solution consists of a single source code file; in such case, you should still submit instructions on how to compile it).

The programs must compile and run on a Linux system with gcc/g++, and you can use eceubuntu to test.

You should also submit a short report (in the order of one or two pages) as either a text file or a PDF. *Please avoid MS Word documents.* Everything will be submitted through LEARN (you may submit all files separately, or pack all the files into a single .tar or .tgz or .tar.bz2 or .zip file).