

Assignment 3: HIDe your password?

1 Introduction

As an additional opportunity to consolidate your skills, we offer three programming exercises. They make up 20% of your overall grade as outlined in the first exercise session. You are encouraged to discuss approaches and share experience with other students. But as it affects your grade, your submission must of course be your own original work, and will be checked for plagiarism.

The submissions are graded fully automatic once per hour after the grading machine is set up (usually a day or two after the release date above). Together with your achieved points you will be provided several logfiles which explain why your solution got this amount of points. You can upload a new solution as often as you like during the submission period, but we will ask questions in the case of a conspicuous high number of submissions. So don't just adapt your code to the grading framework. In the end, your best try counts. Should you find evidence that the grading does not follow the problem statement set out below, please contact us as soon as possible, so we can look into the problem and fix it before too many other students are affected.

2 Problem Description

As you probably know, USB ports can not only be used to plug in USB storage devices or XMC4500 boards configured to be a wicked CDC virtual serial interface. This time we will assume that you aim to infiltrate your victims Tiny Core Linux machine by dropping a textfile named after your matriculation number into the victims `$HOME` directory. Turning on his machine you notice that your geeky colleague has replaced linux' native authentication with his own very secure command line creation of which he claims that "it's the most secure authentication interface ever build. Very secure. Trust me, I'm like a smart person!"

Equipped with your XMC4500 board you start working on a program to crack his poorly designed security software and drop your message into his `$HOME` directory. After all, you don't want to be caught by attentive bystanders while your colleague swaggers about his seriously secure programming skills in the coffee lounge.

Together with this document you should have received a program evaluating a password. During evaluation, the program will report its state by setting some modifier keys of your keyboard. This program is vulnerable to timing side channel attacks, which means that you can intelligently crack the password by measuring the time it takes for the program to respond. However, as long as you're not Barry Allen you won't be able to detect those time differences with enough precision to exploit this side channel by hand¹. The program will generate a new

¹Even if you are Barry Allen, you will not be there for grading.

password each time you start it and report the password to your console output. You can use this to extensively test the authenticator. However, while grading the password will not be printed to screen, so don't bother trying to exploit this particular leak. The password may contain all alpha-numeric characters plus a few special characters:

```
1 const char pwchars[] = "  
  abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890!()-_  
  +=~;:,.<>[]{}/?&$";
```

To ease testing, the authenticator takes two optional command line arguments:

-t sets the timing side channel per character in ms. Default: 20ms.

-l sets the password length. Default: A random number between 10 and 20 characters.

-c disables program termination by Ctrl-C. Set on the grading system, but not default to ease testing.

Usage: ./authenticator [-t ms] [-l characters] [-c]

Please note: There is an `aarch64` version compiled for e.g. M1 processors. However, in order for this to work, you still require a running xserver, e.g. in a Linux VM.

2.1 Part A

In this deliverable, you are supposed to model an HID keyboard in order to breach the authenticator. Together with this problem description you should have received an `example_hid_project` that already implements a keyboard that types `HELLO WORLD` a few seconds after being plugged in. It uses the LUFA USB stack² that comes together with the XMCLib. Use this as a starting point and extend it to exploit the timing side channel of the authenticator to intelligently crack the password.

After cracking the victims authenticator, use your HID keyboard to create a textfile from the command line you are dropped into. The textfile has to contain your full name, be named after your matriculation number, and placed in the `$HOME` directory.

For grading, each run will be limited to 30 minutes, but we reserve the right to cut it off earlier if we detect that your submission completed successfully or ceases to pursue completion, e.g. there are no more attempts to crack the password.

2.1.1 Hints

- Get yourself familiar with HID devices (understanding the `example_hid_project` should be sufficient).
- Get familiar with the authenticator and its responses. Try to figure out when and why the authenticator sets/resets certain modifiers respectively keyboard LEDs.
- Constantly sending the same reports might not be supported by the OS (i.e. it may discard your input).

²You don't need to bother much what the LUFA exactly is. However, it could help you to find support online.

- The keycodes provided together with the LUFA library confer to the US keyboard layout. However, for grading the german layout will be used and, thus, some of the keycodes result in different key press events. You should have received a `.h` file defining the correct keycodes for the german keyboard layout, so you can use those. Don't forget to change your keyboard layout to a german one while testing. (The `example_hid_project` already uses the german keycodes.)
- When trying to assess the timing side channel you may use the CCU. However, there are other methods that may be easier and faster to implement (Hint^{HINT} systick).
- Use some basic statistics to determine if you achieved a sufficient signal to noise ratio in your time measurements and which character stands out.
- Note that the noise of your desktop computer may probably be larger than the one of the grading framework since the framework does nothing else than grading your submissions. So expect your solution to be able to detect smaller timing side channels on the grading system than on your desktop.
- After you breach the authenticator, you will be dropped to a command line interface. However, it is not guaranteed that you end up in the `$HOME` directory.
- The grading system reports your success via `/dev/spidev0.0` to the database. Whilst it is neither the recommended nor the intended way, you may search for vulnerabilities in the authenticator itself and send a `uint8_t` to set the number of correct characters to a value of your choice.
- Since we can only grade one submission at a time, **expect long waiting times** until you receive feedback for your submission! Do not use the grading system as a continuous testbench while development but only to verify that your local results match the grading system once you achieved a major milestone.

3 Submission

3.1 What to Submit

- A ZIP archive containing the `PartA` folder.

3.2 How to Submit

1. From your project directory (i.e. directory `PartA`) run `make deliverable`
2. Make will create a `.zip`-file in `../`.
3. Upload this archive via moodle