In this project you have to demonstrate your ability to perform a cache side channel attack on a system in which the attacker and the victim use a common shared library. Such attacks can be feasible in shared computing systems where programs of different users use common standard shared libraries, and also in PaaS cloud environments where code of different tenants runs on the same operating system with the same shared libraries.

To minimize noise due to other processes on a PC that use the same system shared library, we have a slightly simplified setup with a custom shared library that only you and the victim process use.

Part 1: Preparation: shared library, victim, probing

You will find the code of the shared library (`libnumberpic`) and a sample implementation of two victim processes that use the shared library on the git repository at

`https://gandra.fim.uni-passau.de/csec_git/csec_exercises/project1_template`

It contains a `Makefile` for building both the shared library and two sample victim processes. The shared library renders numbers as png images and has the following API:

```
bitmap_t *numberpic_mkbitmap(int x, int y);
void placenum(int nr, bitmap_t *bitmap, int x, int y);
pixel_t *pixel_at(bitmap_t *bitmap, int x, int y);
int save_png_to_FILE(bitmap_t *bitmap, FILE *f);
```

The header file also defines the types `pixel_t` and `bitmap_t`. The function `placenum` can be used to draw a digit (0..9) at a specific position within a bitmap previously allocated with `numberpic_mkbitmap()`. The library contains an array of constant pixel arrays that define the visual representation of the digits (variable `numbers`).

The program "`test`" is a simple program that generates an image (`test.png`) containing the digits '1' and '4'.

The program "web" is a simple web server that takes two arguments: port number and files directory. If you run the program with "./web 8888 ." in the src folder, it accepts connections on port 8888 and serves files in the current directory (there is a sample index.html). On URLs addressing an image `ABCD.png`, it renders an image (png file) containing the digits A, B, C, D.

Implement, in a file `probe.h` , the inline functions for probing a memory location (measuring and returning the memory access time with `RDTSC/RDTSCP`) and for evicting a memory location from CPU cache (`CLFLUSH`) – see also Lab 1.3. You should be able to explain why you used exactly the instruction(s) you used, including possibly the use of some fence instructions.

```
inline __attribute__((always_inline)) int reload(char *addr) { ... }
inline __attribute__((always_inline)) void flush(char *addr) { ... }
```

Part 2: Basic side channel probing

(a) Checking access times

In one console, run the first test program with "`./test 1`".

Write a simple probe program that – while the test program is running – periodically measures the access time of the memory pointed to by numbers[5] and numbers[7]. Make 1000 measurements, store them in a .csv file, and write a Python program (using pyplot, which you can install in the lab with "pip install --user matplotlib") that creates a graphical plot of the measurement results (X-Axis: iteration counter; Y-Axis: measured access time, use two different colors for ploting the measurements of [5] and [7] in a single graph).

Repeat the measurements on three different configurations:

- Two different PCs (chose between three possibilities: A: a PC in lab 0.028; B: your own PC that you bring with you in the discussion; C: a virtual machine in our private cloud).
- For one of the PCs, make two measurements: probe and victim running on (A) the same CPU core; and (B) different CPU cores. Use "`taskset`" to enforce on which core the probe and the victim is running.

(b) Basic side channel

Write a simple probe program (`probe.c`) that detects if some process uses the `placenum` function of the shared library.

- If no other processes access the shared library, it should not output anything.
- If another process calls `placenum()`, it should print "Access" on stdout (if there is more than one call to placenum() within one second, it may print "Access" one or multiple times).
- If another process calls other functions of the shared library, printing and not printing "Access" is an acceptable behaviour.

You can test your program by starting `probe` and then concurrently

(1) run "test 1" or "test 0"

(2) run "web 8888 ." and point your browser to "http://localhost:8888/1357.png" (or some other number instead of 1357). Press "reload" in the browser...

Part 3: Advanced side channel probing

(a) Digit probing

In this part, the attack target is the web server application "`web`". On each request for png file "ABC.png", it invokes `placenum()` for the digits A, B, C, ... to generate a picture shown on the web browser on the fly. The goal of the attacker is to find out (a) when a web client requests such a picture (solved already by part 2), and (b) finding out what digits are shown in the image.

Implement an extended attacker program `autoprobe.c` that continuously monitors the shared library's memory. It shall detect if a client accesses the web server to generate a png file with digits, and print which digits are shown in the image.

Client accesses: http://localhost:8888/1579.png

Expected output: "`Access: [1], [5], [7], [9]`"

The information must be retrieved using a cache side channel. autoprobe may not obtain the information by listening to network traffic, examining the web servers log file, or other means that do not use a cache side channel.

(b) Enhanced information exfiltration from web form

If you access http://localhost:8888/index.html, you find a web form in which you can type in a number, and after submitting the page (requires JavaScript) will show you a small animation with the number you typed in.

Enhance the autoprobe program of part (a) such that

- it takes a command line argument (integer value `T`, in seconds)
  (without command line argument, autoprobe shall maintain the behaviour implemented in part (a))
- it monitors the shared library for `T` seconds, and tries to guess the correct number entered in the web form.

Submission and discussion

A git repository for your code has been created at: (replace <XX> with your group number)

`https://gandra.fim.uni-passau.de/csec_git/csec_exercises/group_<XX>_project_1`

Submission of code, results and documentation is done via the git repository. You need to commit and push your files to the repository before the submission deadline, and create a tag "final" (`git tag final; git push origin final`).

We expect the following file structure (in addition to the files provided with the shared library) within the repository:

```
src/autoprobe.c
src/probe.c
src/probe.h
src/Makefile
src/plotter.py
results/graphA_1core.png  [or .pdf]
results/graphA_2core.png  [or .pdf]
results/graphB.png        [or .pdf]
doc/doc.txt         [or doc.pdf with documentation for all parts]
```

You should use git for collaboration. We use the authors of git commits as an indication for which group member contributed what to a submission. The file `doc.txt` / `doc.pdf` must indicate your group number and the names of all contributing students.

Each student must be able to explain at least two of the parts 1, 2a, 2b, 3a and 3b autonomously, and for all parts, one group member must be able to explain it. Discussions will take place on Thursday June 6.

You will have to register for a time slot for discussing/evaluating your group's project. Discussion registration details will be announced on Stud.IP.

Submission deadline on git is on **June 4, 23:59h**. This is a hard deadline. Late submissions and submission by other means than via the git repository will not be accepted.