UNIVERSITÄT PASSAU
Fakultät für Informatik und Mathematik

Institute of IT-Security
and Security Law

3rd IT Security Summer School Madagascar 2020

Cache-Based Side Channel Attacks

November 25, 2020

Prof. Hans P. Reiser, Johannes Köstler

Objectives

- Learn about cache-based side channel attacks.
- Implement simple Flush & Reload attacks.
- Implement simple Prime & Probe attacks.
- Visualize and analyze the results.

Resources

- C Programming Reference: `https://www.programiz.com/c-programming`
- Flush & Reload Attack: `https://eprint.iacr.org/2013/448`
- Practical Flush & Reload: `https://www.researchgate.net/publication/289952669_Cross-Tenant_Side-Channel_Attacks_in_PaaS_Clouds/`
- Practical Prime & Probe: `http://palms.ee.princeton.edu/system/files/SP_vfinal.pdf`

Report

- Explain the necessary steps.
- Document your source code.
- Include the measurement plots in your report.

INTRODUCTION

The introduction recaps some basic C programming, which forms the basis for the practical tasks in this project. Create a first C program (`hello.c`) with the following functionality:

1. Implement a `main` function that

   - stores a pointer to the first command line argument in a global variable `nameptr`
   - stores a pointer to dynamically allocated memory with `malloc()`, sufficiently large to store a copy of first command line argument, in a global variable `ucnameptr`
   - copies an upper-case version of the content referenced by `nameptr` to the dynamically allocated memory referenced by `ucnameptr`
   - calls a function `printhello(2)`(see below).

   Note: the use of global variables is, in many cases, bad programming style and most of the time should be avoided in real C programs.

2. Add a function `printhello(int n)` that

   - stores a reference to a constant string `"Hello "` in local variable `helloptr`
   - executes `n` iterations of a loop
   - in each iteration, prints a line containing the contents of the strings referenced by `helloptr` , `nameptr`, a colon (`":"`) and the string referenced by `ucnameptr`

3. Create a `Makefile` with targets

   all compiles and links everything, two separate steps for compiling and linking, producing executable file hello

   clean removes all automatically generated files

4. Extend the printhello function (at the end) with an additional instrumentation print statement that prints

   - addresses of content referenced by variables `ucnameptr`, `nameptr`, and `helloptr`
   - address of variables `nameptr` and `helloptr` in memory
   - address of functions `printhello` and `printf` in memory

5. In case your operating system is Linux you can investigate the memory layout of our program. Integrate code at at the end of the main function that:

   - extracts the addresses of variables `nameptr` and `helloptr` in memory
   - obtains the process identifier (PID) of the running process using the system call `getpid()` and reads the content of pseudo file `/proc/<pid>/maps` (`<pid>` is the return value of `getpid()`) by, for example, executing the external command “**`cat`**” with the `system()` library function).

```
char cmd[50];
snprintf(cmd, "cat /pro/%d/maps", getpid());
system(cmd);
```

For each of the addresses printed by the `printhello` function, find out to which memory section in the process map it corresponds to. Explain these results.