Assignment 4 — Side Channel Attacks

2018-10-16

In this assignment you will mount a side-channel attack against an implementation of modular exponentiation, and will fix the implementation to make it resistant to microarchitectural side channel attacks. The application to attack is `num.c`, which you can find using `tar xf assignment4.pdf num.c`. It uses the (fixed) `ybn` library to calculate a single modular exponentiation with hard-coded base, exponent and modulus.[1] We recommend that you use the Mastik toolkit (https://cs.adelaide.edu.au/~yval/Mastik/) for the attack, but this is not a requirement. You may use any other software, but must credit the source.

Task 1 (40%)

Implement a side-channel attack that leaks information on the exponent in the provided binary. Analyse the side-channel information to identify the success rate, i.e. the average number of exponent bits correctly leaked from each execution of the attack.

Note that because the exponent is hardcoded in the binary (and because you have access to the source code) using a side-channel attack is not the most efficient method of extracting the exponent. The assignment simulates a situation where you do not have such a level of access to the binary. You should use a side-channel attack!

Task 2 (20%)

Edit the source code to change the value of the variable `num` in `main()` to 4, 3, 2, 1, and 0. Build the binary `num` with ybn and repeat the attack against the compiled binaries. Identify the smallest value (from the list) that you can attack, and describe the attack and the results.

Task 3 (40%)

Modify `ybn_modexp()` to mitigate all microarchitectural side-channel attacks, under the assumption that all of the other functions in `ybn.c` have constant-time implementations.[2] Note that defending only against the attack you implemented is not enough.

Submission

You should submit a `.tar` or a `.tgz` archive. The archive should contain a single directory, whose name is your student a-number. The directory should contain the following files:

- `ybn.c` - your (fixed) version of `ybn`.

[1] We will provide our fixed version of `ybn` on Sunday, after all of the late submissions of Assignment 3 are due. Until then you can use your Assignment 3 submission.

[2] The provided implementations are not constant time, but you do not need to worry about this.

- `answers.pdf` - A PDF file that describes the attacks and results of Task 1 and Task 2, and also describes the changes you made to `ybn_modexp()` in order to defend against side-channel attacks.
- `results.log` - A log file that contains the raw side-channel information from one attack on the original binary. If you use the `FR-trace` utility, this file is the output it produces. Otherwise, you need to include some method of producing a meeningful log file in whatever code you write for implementing the attack.
- Any software you wrote for the attack or for analysing the side channel results.

Notes on Mastik

We have only tested Mastik with Intel Core processors. Mastik does not work well with AMD processors, may have problems with non-Core processors (e.g. we are aware of problems on some Atom or Celeron processors). It does not work at all with ARM processors. The machines in the computer labs are ok. Last, Mastik may fail in some virtual machine configurations. It is highly recommended that you run Mastik from a Linux host. If your machine is running another operating system, you may want to create a dual-boot configuration or get a live Linux distribution.

When building Mastik on your machine, you want to have the development packages of `libbfd`, `libelf`, and `libdwarf` installed. On RedHat-like distributions (CentOs, Fedora, etc.) install the packages `binutils-devel` and `libdwarf-devel`. On Debian releases (e.g. Ubuntu) install `binutils-dev`, `libdwarf-dev`, and `libelf-dev`.