Lab 3: Timing Attacks

Karlstad University

Mahdi Akil – mahdi.akil@kau.se
Samuel Wairimu – samuel.wairimu@kau.se

1 Introduction

Timing attacks use time as a side-channel which leaks information about the data that a program is operating on. Such attacks are possible if the running time of a program is a function of the data *and* only some of that data is provided as input by the attacker. In this lab, you will write an exploit for a web server that does user authentication in non-constant time as follows:

1. The server computes a user's unique authentication tag based on a secure primitive for message authentication codes.
2. This tag is then byte-by-byte compared to a user-supplied authentication tag such that an error is returned *as soon as there is a mismatch*.

2 Preparation

To better understand timing attack vulnerabilities and authentication tags, we recommend that you look at the following two resources:

- Coda Hale: A Lesson in Timing Attacks (or, Don't use `MessageDigest.IsEquals`). https://codahale.com/a-lesson-in-timing-attacks/ (2009), accessed 2022-01-18.
- Jean-Philippe Aumasson: Serius Cryptography: A Practical Introduction to Modern Encryption, Chapter 7, No Starch Press (2018).

3 Lab Environment

We setup a web server that listens for HTTP GET requests on dart.cse.kau.se:12345/auth/<delay>/<user>/<tag>, where `<tag>` is a 16-byte hex-encoded tag for `<user>`.[1] The `<delay>` specifies how long the server will pause in ms after each byte-by-byte comparison, making it easier to exploit the vulnerability without an excessive amount of repetitions.

For example, access should be granted (HTTP 200 OK) quickly for:

dart.cse.kau.se:12345/auth/10/alice/c2d07d3c5ed87430f67f20ce9e711307

However, it should take a few seconds for:

dart.cse.kau.se:12345/auth/200/alice/0231d46f455b0e61c15924e9da02686d

Note the difference in delay (from 10 ms to 200 ms per comparison) and that the tags are valid (so there will be 16 byte comparisons).

If access is denied, the server returns HTTP 400 Bad Request (on malformed input) and otherwise HTTP 401 Unauthorized (indicating tag mismatch for `<user>` and `<delay>`).

4 Task

You may work alone or in pairs to authenticate for a delay less than 100 ms with `<user>` set to your own KauID(s). Do not use the KauID of any other student: this is considered cheating and treated as such. Note that any programming language may be used. If you use code by others, make sure to be clear about the sources: plagiarized code is plagiarism. Do not use code from other students in the course.

[1]Source code: https://github.com/rgdd/timing-server

5 Submission

Other than completing the task in Section 4 successfully, you must write a report that at least contains the following:

- Full name(s), email address(es), and KauID(s).
- The tag(s) you found to authenticate your kauID(s).
- A problem description and necessary assumptions (if any).
- Principles and limitations of your exploit code (at least 200 words).
- How to run your code on Linux (use an Ubuntu VM if you have to) and a rough estimate of the running time (e.g., seconds or minutes?).
- How to execute your code to perform a timing attack with your KauID(s) with a delay less than 100 ms.
- Encountered issues and challenges (if any).
- Lessons learned by this lab related to timing attacks (at least 200 words). For example, how do you mitigate them?

Submit your exploit code (attachment) and the report (PDF) via Canvas by the end of the course. Late labs may not be corrected until the next examination period.

Acknowledgements

The following people have contributed to this lab specification: Mahdi Akil, Rasmus Dahlberg, Leonardo Martucci, Tobias Pulls, Artem Voronkov, and Samuel Wairimu.