Lab 2 “Secure Coding”

Objectives

In this lab you will:

1. Practice reading program code and finding potential security breaches in it.
2. Explain how exactly do your findings threaten security of the analyzed code.
3. Categorize each finding as threatening either *integrity*, *confidentiality*, or *availability*.
4. Fix each finding in the code; explain how your modification fixes the problem.
5. Introduce hardenings to avoid the risks created by similar problems.

Description

You are given an archive with two files, `hash.c` and `hash.h`, that contain a naive hash table implementation. The code contains programming mistakes that make it vulnerable in some way. You will need to:

1. Copy `hash.c` and `hash.h` to `hash_fixed.c` and `hash_fixed.h`, respectively.
2. Find as many security related programming mistakes as you can in `hash.c` and `hash.h`.
3. For every mistake you find (comment directly in the code for each step):
    a. Clearly locate the mistake in the code.
    b. Explain why it is a mistake and how it affects security of the code.
    c. Categorize the mistake as affecting *integrity*, *confidentiality*, or *availability.*
    d. Fix the mistake in `hash_fixed.c` and `hash_fixed.h`.
    e. Explain why do you expect your fix to remove the problem.
    f. Introduce a hardening to avoid the risks created by the mistake.
4. Summarize the hardening instructions from 3.f, for every problem, in file `hardenings.txt`.

Expected outcome

We expect you to submit a .zip archive with the following structure:

<your_name>_lab_2.zip
|__hash.c
|__hash.h
|__hash_fixed.c
|__hash_fixed.h
|__hardenings.txt

Explanation:

- `hash.c` and `hash.h` will contain the original executable code accompanied with your comments explaining why the code is buggy (steps 3.a-3.c in the Description).
- `hash_fixed.c` and `hash_fixed.h` will contain the fixed code accompanied with your comments explaining how your corrections fix the problems (steps 3.d-3.e in the Description).
- `hardenings.txt` will contain **reproducible** hardening instructions to avoid the risks created by similar problems in **arbitrary** C code.

Consistency criteria

The following are the minimum requirements that should be met by your submission:

1. Archive name and contents structure that **exactly** matches the one described above.
2. The code compiles.
3. The measures from `hardenings.txt` must:
    a. protect the system in which the program runs from potential coding mistakes;
    b. be **realistically** reproducible on a regular Ubuntu machine;
    c. work for arbitrary C code (you can assume the 1 x C file + 1 header file setting);
    d. be self-contained – I should not have to google your instructions!

Resources

Start with the CWE Top 25 enumeration of the most dangerous software weaknesses as of 2022. Chances are that the mistakes you detect fall into one of these categories. If you click an item in the list (say, Use After Free), you will see a detailed explanation of how to detect and prevent the corresponding issue, together with technical examples.