POLITECNICO DI TORINO

DIPARTIMENTO DI ELETTRONICA E TELECOMUNICAZIONI

ACADEMIC YEAR 2023/2024

Operating systems

Laboratory experiences

Authors:
Bonino Andrea - 314709
Fasolis Gabriele - 314903
Mattiauda Mattia - 315096
Mondino Ettore - 317755
Tartaglia Federico - 319857

Professor:
Stefano Di Carlo

October 25, 2023

Contents

LAB 3 - Application Program with SHA-256

In this lab experience you will have to program an application that can take full advantage of the encoding device and the library you created (alternatively you can find a working library for the device in the lab solutions). The application has to be written in C language and needs to be cross compiled along with the *Sha256_library* as shown in the previous lab experience with the `Test_sha256.c` file.

1 Program specification

In the material folder you can find the program `UnsafeRegistration.c`. Open it and try to understand how the code works: try to compile it and run it with a series of test inputs.
The generated executable simulates a **login system** which manages multiple users registered with name, password and phone number (which can be used in case a user forgets the password).

As you can see the program is working in all its parts and the databases with usernames, credentials etc. are properly stored. However, **the program is not safe**! Imagine if this same program was used in a company to allow the employees identification and let them access a private area to upload their projects and gather sensitive information about the company; do you notice any security vulnerabilities that could be exploited by someone inside or outside the company?

> **Tip:** The database is currently composed of a bunch of vulnerable text files which contain usernames, passwords and telephone numbers. These information need to be hashed by the SHA-256 algorithm in case a malicious user accesses the files in which these sensitive data are stored.

Try to edit the provided .c file and turn it into `SafeRegistration.c` by fixing the security issue. To do this, use the SHA-256 algorithm by interfacing the application with the device through the library *Sha256_library*.

> **Tip:** Some files must be hashed, but at the same time the access by the user who enters username and password (as well as the possibility of changing password only if the second factor corresponds to the correct one) must be guaranteed and verified.

Once you have prepared the `SafeRegistration.c` file, place in the `files` directory, where you have to insert the following files:

- `Sha256_library.h` and `Sha256_library.c`
- `SafeRegistration.c`

You have to modify `myhello.bb` in this way:

```
DESCRIPTION = "Simple helloworld application"
LICENSE = "MIT"
LIC_FILES_CHKSUM = "file://${COMMON_LICENSE_DIR}/MIT;md5=0835ade698e0bcf8506ecda2f7b4f302"
SRC_URI = "file://Sha256_library.c  file://Sha256_library.h file://Test_sha256.c"
S = "${WORKDIR}"
do_compile() {
    set CFLAGS -g
    ${CC} ${CFLAGS} Sha256_library.c Sha256_library.h Test_sha256.c ${LDFLAGS} -o TestSha256
    unset CFLAGS
}
do_install() {
     install -d ${D}${bindir}
     install -m 0755 TestSha256 ${D}${bindir}
}
```

2 About SHA-256

SHA-256 (Secure Hash Algorithm 256-bit) cannot be reverted because it is a one-way cryptographic **hash function**. When you apply SHA-256 to data, it produces a fixed-length output (256 bits), which appears random and is computationally **infeasible to reverse back** to the original data. This property is crucial for data integrity and security, as it ensures that you cannot determine the original input solely from its hash value.

Notice that, even when SHA-256 is used, the `SafeRegistration.c` login system may be still **vulnerable** to attacks. **Brute forcing SHA-256** means trying all possible input values until you find one that produces the same hash as the target. Given that SHA-256 has a large output space (256 bits), it is computationally infeasible to perform a brute force attack on it in practice, especially for long and complex inputs. In our case, though, the attacker could know part of the input that will be hashed: the telephone number is a private information, but it can easily fall into the attacker's hands!
Knowing part of the key (input of the hashing function) reduces the number of possible combinations that need to be tried during a brute force attack (it improves efficiency also in other more sophisticated kinds of attacks). This can significantly speed up the attack because the attacker doesn't need to test all possible keys; they can focus on a subset of the keys that match the known part.