Project Lab 3

Time your Programs – 4 h

Camille MONIÈRE (course manager, A. Pr.) Mohamed EL BOUAZZATI (teacher, Dr.)
Adam HENAULT (teacher, Ph.D. Student)

November 2024

1 Time your Programs

1.1 Instructions

In this lab, you will have to use and to explore several programs, in order to understand important notions, and to be able to reformulate them. You will understand why *suid* programs are less and less used nowadays, and learn how much timing is an important notion in security.

The programs are given in the archive, as well as guidelines to compile, set them up, and use them.

Your tasks are[1]:

- Explain your analysis protocol (threats identified, ideas, assessment method...),
- Explain what the programs are supposed to do (be imaginative),
- Explain what are the issues,
- Explain the solution you can bring in terms of design,
- Patch the vulnerabilities in the code, and provide the commented patch.

On moodle, you will find a TGZ archive named `test-set.tar.gz`. This contains several projects. You will have to extract the tarball somewhere with "`tar -xvzf test-set.tar.gz`". Then you can begin working on it.

You must write patches and a report addressing the points above. Again, to help you choose the level of language, a bit of context is given in the next section.

The report have to be a PDF file. You are encouraged to write it using Markdown and convert to PDF using `pandoc` (like the current document). The result should be named `group_x_y_report.pdf`[2]. You will find a presentation of Markdown here, and explanations on the syntax here. To convert a Markdown file using `pandoc`, on a UBS machine, you can run:

```
pandoc --highlight-style=pygments --pdf-engine=pdflatex --standalone --to pdf \
    -o group_x_y_report.pdf  group_x_y_report.md
```

You may also use the Markdown template available on the forge[3].

The patch must be an all-in-one text file named `group_x_y_patch.txt`. To create it on Linux, if you have an initialized git repository in the extracted *test-set*, use the method you used in lab 2.

On Moodle, you have access to a *Project Lab 3 assignment*, where you can provide the report alongside the patch to us.

[1] These tasks **are not** questions. The report should be structured.
[2] Replace "x" and "y" by the number and letter of your group
[3] https://forgens.univ-ubs.fr/gitlab/moniere/template-markdown-pandoc

This lab is graded. The clarity of the report and its structure are evaluated, as well as the content itself. Yet again, exploiting is far from enough. You need to explain, diagnose, trace back issues and conceptual defects. The patch quality and the explanations about the fixes are thus both evaluated.

1.2 Context, Documentation and Hints

For the sake of experimentation, some rules need to be ignored, and some actions must be taken. The first two programs must be owned by the *root* user (uid `0`) and must have the *suid* bit set. To do that without endangering the university nor your own system, you have to run the programs in virtual system (or at least, in a container). You may use, for instance, the one entrusted to you on `caps-machine.univ-ubs.fr`.

First, connect to the server using `ssh <username>@caps-machine.univ-ubs.fr`. Then, start a tmux server using `tmux`. You are **strongly** encouraged to use git to synchronize files between the VM and your machines, using the forge for instance.

You can, for each project, run:

```
1  cd project_vX # Where X is '0' or '1'
2  cmake -B build -S . -DCMAKE_BUILD_TYPE=Release # or Debug
3  cmake --build build -j
```

The `cmake` at line 1 configure the program, while at line 2, it builds it. You should take a peak at what is actually done in the `CMakeLists.txt`, especially inside the `add_custom_command()` directive (line 25 in v0 and 28 in v1)...

You can also use the provided top-level makefile `project.mk` like so:

```
1  # To make both project in release
2  make -f project.mk
3  # To make both projects in debug
4  make -f project.mk BUILD_TYPE=Debug
5  # To clean
6  make -f project.mk clean
7  # To clear
8  make -f project.mk clear
```

1.2.1 Hints

Concerning exotic functions :

- `int poll(...)` function polls the file descriptor, waiting for input for 3000 ms. If nothing is available, it returns 0. If there is an error, it returns -1. Otherwise, it returns a positive value.
- `int access(...)` function checks that the effective user (the user actually using the program) has the rights to access a file using the specified mode (`R_OK`: read, `W_OK`: write, `F_OK`: file exists). If it does, `access()` returns 0. It returns 1 otherwise.
- `int fseek(...)` sets the file pointer somewhere in the file. `ftell()` returns the actual position of the file pointer. Combined, they may allow to compute the size of a file.
- Don't forget to read the manual ; `poll()` and `access()` are in section 2, `fseek()` and `ftell()` are in section 3.

Concerning the lab itself:

- Consider what the programs do, and try them on different files, with different permissions,
- the name of the lab should help you,
- search for symbolic links, and read about the `ln` command.

1.3 The Lore

*Note that the lore **is not** directions and **shall not** be substituted to previous directions.*

You are a C developer in a company in the field of home automation, in the software security division. You had a busy month, but it is not over yet.

Speaking of, you receive an e-mail from your co-worker:

> Hey <insert your name>,
>
> I have an assignment for next week, and I wanted your help. I have produced two programs: one securely copies a file, and the other calculates the hash of a file and stores it into another. Can you audit them, patch them if needed, and report everything to me (succinctly) ?
>
> I will provide you a test platform on the server.
>
> Can you send me patches and report for Thursday or Friday ? Thanks !
>
> Regards, Justin Time

1.4 Nota Bene

1. Markdown introduction: `https://www.markdownguide.org/getting-started/`
2. Markdown syntax: `https://www.markdownguide.org/basic-syntax/`