Project Part 1

CS3235: Memory Safety and Defense in Depth

Instructor : Prateek Saxena   Creator : Aditya Ranjan Jha

1. Overview

The first part of the project involves translating a vulnerable C program into safe Rust code, eliminating memory safety vulnerabilities. The instructions to setup and run are provided in the `README.md` file of the project directory.

1.1. Part A: Memory Safety Vulnerabilities in C

The first part of the assignment involves running a vulnerable C program and observing how various payloads exploit its memory safety issues. Instructions to run the program are provided in the `README.md` file of the assignment directory.

As you experiment with these payloads, you will observe crashes, memory corruption, and other undefined behaviors. After this, the next step is to translate the C code into **safe Rust** code that preserves the same external behavior. The trabslation should be function by function, (you can add extra helpers if needed) but you should not use external crates or unsafe code.

You should then test the same payloads (translated version) on your Rust program to check whether the vulnerabilities still persist.

During this process, you will likely discover that Rust's memory safety model—especially its ownership and borrowing rules—prevents many of the vulnerabilities from being expressed in the first place. For example, a direct translation might lead to compiler errors like the following:

```
59 | fn init_database() -> &mut UserDatabase {
   |                        ^ expected named lifetime parameter
   |
   = help: this function's return type contains a borrowed value, but there is no value for it
to be borrowed from
help: consider using the `'static` lifetime, but this is uncommon unless you're returning a
borrowed value from a `const` or a `static`
   |
59 | fn init_database() -> &'static mut UserDatabase {
   |                         +++++++
help: instead, you are more likely to want to return an owned value
```

In other cases, some vulnerabilities may still cause **runtime panics** (but not memory corruption), such as:

```
Running Test Payload OutOfBoundsPayload
Adding a user with name
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
thread 'main' panicked at src/database_fix_temporal.rs:66:9:
index out of bounds: the len is 50 but the index is 50
```

Your goal is to eliminate **all memory safety vulnerabilities** in the Rust version such that it:

- Preserves the original functionality.
- Handles all test payloads gracefully (e.g., by throwing exceptions or returning errors).
- Does not crash or exhibit undefined behavior.
- The final database state should be printed as the given in `sample_output.md` file.
- Does not use `unsafe` code or external crates.

2. Deliverables

The deliverables for this part of the assignment, along with the assigned rubric points, are:

1. **Rust translation** (instructions given in `README.md`): 5 points
2. **Report**: 5 points (max 2 pages, PDF format `report.pdf` in the base project directory)
   - Describe the source of the vulnerabilities in the original C code: 2 points
   - Describe the process of translating the C code to Rust (including challenges faced): 3 points

Total: 10 points

3. Submission Instructions

You should submit a zip file of this project directory to the Canvas submission portal.

3.1. Format

- Name your zip file as: `CS3235_Project1_<StudentID>.zip` (replace `<StudentID>` with your matriculation number).
- Submit individually; group submissions are not allowed.

3.2. Contents

The zip file should contain the following structure:

```
├── cleanup_docker
├── database
├── database.c
├── database-rust
│   ├── Cargo.lock
│   ├── Cargo.toml
│   ├── src
│       └── database_fix_full.rs
├── docker_compile_run.sh
├── Dockerfile
├── docker_rust_run.sh
├── README.md
├── report.pdf
└── run_docker
```

3.3. Report

- Include a `report.pdf` at the top level of the project directory.
- The report must contain a final section listing all references you used (documentation, websites, papers, etc.), including generative AI tools and any chat history you consulted.