UP23 Lab06

Date: 2023-05-01

Assembly Sort Challenge

This lab aims to practice writing assembly codes. Your mission is to implement a sort algorithm in assembly language that sorts a given array containing n `long` integers. To have more fun, we implement a live scoreboard to show solution ranks based on the *correctness*, *running time*, and *code size*. Please try your best to implement efficient assembly codes.

The Challenge Server

The challenge server can be accessed using the `nc` command:

```
nc up23.zoolab.org 10950
```

Upon connecting to the challenge server, you must first solve the Proof-of-Work challenge (ref: pow-solver (https://md.zoolab.org/s/EHSmQ0szV)). Then you can follow the instructions to submit (1) the shellcode to run on the server and (2) optionally submit your token for the live scoreboard, and (3) the assembled machine codes used to sort the array.

The live scoreboard is available here (https://sort.zoolab.org/)

Lab Hints

Here are some hints for you. You can solve the challenge locally and then verify your solution on the challenge server.

1. We will invoke your uploaded machine code from offset zero. It is called from the server with two arguments, the pointer to the array and the number of `long` integers in the array. The prototype of the function is

   ```
   typedef void (*sort_funcptr_t)(long *numbers, int n);
   ```

   We do not have another array for placing sorted numbers. You can reorder the elements in the array `numbers` directly.

2. Your code cannot invoke any system call in your implementation. If you need to allocate memory spaces, allocate them on the stack.

3. We have runtime constraints for your uploaded machine code.

   - Your code size cannot exceed 512 bytes.
   - The maximum running time is 120 seconds.

4. To simplify the code submission process, you can use our provided `pwntools` python script to solve the pow and submit your shellcode. The submission script is available here (view (https://up23.zoolab.org/code.html?file=up23/lab06/submit_ea7c7bdbbbf63d90e647337bdf623049.py) | download (https://up23.zoolab.org/up23/lab06/submit_ea7c7bdbbbf63d90e647337bdf623049.py)). You have to place the `pow.py` file in the same directory and invoke the script by passing the path of your compiled solver executable as the first parameter to the submission script. The usage of the submission script is as follows.

   ```
   ./submit.py filename.s [scoreboard-token]
   ```

   The above command assumes you implement your assembly codes in `filename.s`. If you plan to submit your score to the live scoreboard, please pass your *scoreboard-token* as the third parameter to the submission script.

5. The challenge server only accepts machine codes generated for Intel x86_64 CPU.

Grading

Before we illustrate how to grade your implementation, we define the *correctness rate*. We use the following codes to measure the correctness rate of your output:

```
int check_numbers(long * numbers, long *sorted, int n) {
    int i;
    for(i = 0; i < n; i++) {
        if(numbers[i] != sorted[i]) {
            printf("** check failed.\n");
            break;
        }
    }
    if(i == n) {
        printf("** check passed. Good Job!\n");
    }
    return i;
}
```

The correctness rate is obtained by $\frac{i}{n}$.

- [60 pts] Your machine code achieves 0.1 correctness rate.
- [20pts] You get additional points if your program achieves a correctness rate higher than 0.1. Suppose your correctness rate is r (ranging from 0.0 to 1.0). The equation used to calculate the additional points linearly based on the correctness rate is $\frac{r-0.1}{0.9} \times 20$.
- [15 pts] You can get additional points based on your rank on the live scoreboard. We plan to classify all the unique identities on the leaderboard into three classes. The top class gets all 15pts, the second class gets 10 pts, and the last class gets 5 pts.

> We have an execution time limit for your challenge. You have to solve the challenge within 120s.