UP23 Lab08

Date: 2023-05-15

Random Walk in a Forest

This lab aims to practice implementing program tracing using the introduced `ptrace` interface. Your mission is to obtain the FLAG from a dynamically generated executable. Please follow the introduction to the challenge server and the lab hints below to see how it works.

The Challenge Server

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

```
nc up23.zoolab.org 10965
```

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 from the server to solve the challenge.

The server dynamically generates a binary challenge on receipt of a solver uploaded from you. Suppose your uploaded solver is called `runner`, and the challenge executable generated on the server is called `chals`. The server then uses the following command to invoke your solver.

```
./runner ./chals
```

Note that `chals` is an `x86_64` executable, and your solver must be compiled as an `x86_64` executable. You can use the `ptrace` interface to interact with the `chals` program and control its program flow to achieve the goal.

Lab Hints

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

1. Because the challenge executables are dynamically generated on the server, we provide three possible samples.

   1. sample #1 (view (https://up23.zoolab.org/code.html?file=up23/lab08/sample/sample1.c) | download source (https://up23.zoolab.org/up23/lab08/sample/sample1.c) | download executable (https://up23.zoolab.org/up23/lab08/sample/sample1))
   2. sample #2 (view (https://up23.zoolab.org/code.html?file=up23/lab08/sample/sample2.c) | download source (https://up23.zoolab.org/up23/lab08/sample/sample2.c) | download executable (https://up23.zoolab.org/up23/lab08/sample/sample2))
   3. sample #3 (view (https://up23.zoolab.org/code.html?file=up23/lab08/sample/sample3.c) | download source (https://up23.zoolab.org/up23/lab08/sample/sample3.c) | download executable (https://up23.zoolab.org/up23/lab08/sample/sample3))

   Note that you cannot successfully compile the codes because we did not provide the required `liboracle` files. Nevertheless, the implementation of `liboracle` is irrelevant to the solution of this lab.

2. If you look at the challenge sample source codes, a challenge is composed of many `if-else` statements, which guide the program flow based on the values stored in the `magic` variable.

3. There are several `oracle_*` functions in the sample code, but it doesn't matter how they are implemented.

   > Please notice that the `oracle_*` functions implemented in the sample codes may differ from those implemented on the server. Your solution should not depend on the `oracle_*` functions.

4. For each selected program flow, it always executes the following functions: one `oracle_connect`, one `oracle_reset`, several `oracle_update`s, and one `oracle_get_flag`. If the selected program flow is correct, the function call to `oracle_get_flag` prints out the `Bingo!` message for you. Otherwise, the `oracle_get_flag` function simply prints out a dot (`.`).

5. The default value for the `magic` variable is `0000000000`, which is unlikely valid for the program. You have to find out the correct value that walks through the correct program flow.
6. To achieve the goal mentioned in the previous point, you can use the `ptrace` interface to control the program flow of a tracee. Generally, you can repeatedly fill different values into the `magic` variable and restart the `if-else` checks from the beginning until you find a correct `magic` value.
7. Note that the `oracle_get_flag` function returns zero on success or -1 on error.
8. You may also notice there is a `CC()` macro defined in the sample codes, which use used to insert `int3` instructions in the sample codes. You can leverage these checkpoints to implement your solution.
9. **IMPORTANT** Your solver is invokved in a sandboxed runtime. Please link your program with `-static-pie` option when you invoke the `gcc` compiler.
10. 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/lab08/submit_1310ed1cb40c16067911bdda36189abf.py) | download (https://up23.zoolab.org/up23/lab08/submit_1310ed1cb40c16067911bdda36189abf.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.
11. The challenge server only accepts machine codes generated for Intel x86_64 CPU.

Grading

- [40 pts] Your solver can solve sample challenges #1, #2, and #3 locally.
- [60 pts] You solver can solve the challenge on the server.

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