Saarland University
Systems Security
Prof. Dr. Thorsten Holz

CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

Systems Security

Winter Term 2023/2024

Assignment 1 / November 8th, 2023
Due November 21th, 2023, 23:59

Remote Environment for Practical Tasks

We have prepared a *remote exercise framework* (REF) equipped with all tools needed to solve practical tasks. For information on how to connect, carefully read "Assignments" in CISPA CMS before working on the tasks. Submitted solutions will be tested in this environment and graded accordingly. Thus, you should complete (or at least check) your solutions on the remote system.

Task 1: Bandit (optional)

This exercise will refresh (or establish) Linux basics needed to complete future assignments in this course. Your task is to play the wargame *Bandit* (`https://overthewire.org/wargames/bandit/`) at least up to (and including) level 14. For each level, submit the corresponding password.

Task 2: Binary Analysis (10 points)

Your very first task will help you familiarize yourself with some tools useful for analyzing binaries. Assume the following setting: You want to analyze the program `intro`, however, its C source code has been lost. You know there must be a `secret` function in it. As this is the very first exercise, it will contain more detailed instructions than following exercises will *(especially italic comments like this will provide additional information)*. To get started, check the task name specified below this exercise in *Remote login (it is `intro` for this exercise). At this point, you should have worked through the `Remote Exercise Framework (REF)` PDF (cf.* Materials *in CMS) - if you did not, go there now before continuing. If something is not working, do not hesitate to contact us. Once you are set up, connect to the initial task and work through the following tasks:*

a) First, let us extract the `secret` function. Use `objdump` to extract the assembly code of the function `secret` *(To find out which flag of `objdump` prints the program's assembly code, call `objdump --help` or, more verbose, `man objdump`)*. Make `objdump` use the Intel syntax. Submit the result within your submission PDF file (Use a monospaced environment such as `verbatim`, `lstlisting`, or `minted`). Note that you are not required to understand the assembly code at this point (but we will dive into this soon!).

b) *Statically* analyzing *(this means looking only at the code and reasoning about it)* the code is out of scope for now. Let us just hop into the program and look at it *dynamically (the opposite of static analysis: we enrich our analysis by running the program with diverse values and observe its behavior)* with `GDB`, the GNU debugger. Use some resource *(either the ones we linked in the `Remote Exercise Framework (REF)` PDF or search the Internet! This course will expect you to do some research on your own)* to familiarize yourself with the following `GDB` commands: `break`, `stepi`, `nexti`, `continue`, `run`, and `finish`.

   1) Once you know what these commands do, set a breakpoint onto the function `secret` and run the program with input 5.
   2) Then, step through the function.

3) What value is placed in the register `eax` <u>after</u> executing the fourth assembly instruction in the function?

4) What value is there after executing the fifth instruction?

5) If we tell you that the function's return value is stored in the register `eax`, can you tell us which value is returned for the input 5?

Find out how to run `GDB` commands from a file and place your commands from above in `commands`. Submit within the remote environment by typing `task check` to run sanity tests and then `task submit`.

c) Great we made the first steps! So the next question is whether we can generalize this knowledge: Can you find a function $f(n)$ that has the same functionality *(the format $f(n) = n+1$ would be the solution if the secret function returns always your input value + 1)*? Either try to understand the assembly code or simply use the code as oracle, for example for input values 6, 10, 20. In other words, can you *reverse engineer* the function? Submit your solution in pseudocode in the PDF file.

d) Alright, with that done - let's look around in the program.. Use `readelf` to extract the *program sections* from the program. If you do not know what a program section is, use a search machine to learn more about this concept (and review the slides, where program sections were briefly explained). Submit your solution in the PDF file.

e) For the following C code snippet, state in which section (or location) the elements referred to by `<A>`, ..., `<D>` are located when the binary is executed (and why). Submit your solution in the PDF file.

```
1  static int MAX_OBSERVED_VALUE;    // <A>
2  static int THRESHOLD = 250;       // <B>
3
4  int test_function() {             // <C>
5      int some = 5;                 // <D>
6      if (some > THRESHOLD)
7          MAX_OBSERVED_VALUE = some;
8      return 0;
9  }
```

Remote login: `intro`

Task 3: Environment (7 points)

In this task, you will investigate *environment variables* on Linux and their purpose. To do so, analyze the program `test` and its source code `test.c`. For this task, you should be familiar with debugging and `GDB` in particular[1].

a) Explain in two sentences what *environment variables* are used for on Linux.

b) Run `test` for a few inputs and observe the addresses printed for `arr`. Then, open the program in `GDB` and re-run it for the same inputs. Compare the addresses and explain why `arr`'s address differs depending on whether you run `test` within or outside of `GDB`. To this end, compare the environment variables set in `GDB` to those set outside of `GDB` and investigate where environment variables are stored.

c) Create a `.gdbinit` to minimize the differences. Submit your `.gdbinit` within the remote exercise framework. Also, explain with which path you have to call any program outside of `GDB` such that the addresses within and outside of `GDB` are the same.

Hints:

- You can download files from the remote environment, e. g., with
  `scp -P 2223 <task_name>@syssec.pwning.academy:/home/user/<filename> .`

Taskname for remote: `environment`

[1] Some introductory material can be found online: https://youtu.be/D8imWEgyS1g and https://youtu.be/bWH-nL7v5F4

Task 4: Bash Introduction (8 points)

Write a simple C program to dump all command line arguments passed to your program (one argument per line). Submit your solution in a file called `solution.c` via REF. Additionally, explain the following (and submit your solution to these questions in the PDF file):

a) What is `argv` and `argc` – where do they reside in program memory?

b) What does `argv[0]` contain?

c) Run your program for the following inputs: `ABC`, `ABC DEF`, `"ABC DEF"`, `\"ABC DEF\"`. What is the impact of the whitespace? Why do we use quotation marks? What happens if they are escaped (`\"`)?

Taskname for remote: `bash`