Assignment 2: Reconstructing Execution From Memory Dumps

Context:

You are a cyber forensic investigator. You obtain a criminal's computer and locate a suspicious program that the criminal was running when he was arrested. The criminal was trying to send a secret message through this program, and that's what we want to know. Unfortunately, the program gets input (i.e., the secret message) from a file and then deletes the file right after it gets.

Luckily, you obtain a memory dump from the criminal's computer. Your goal is to identify the original message that the criminal wanted to send, from the memory dump. You have the program's binary too.

Description:

You are given a *binary program* (named **steg**) and *memory dump files* generated from the program. **steg** is a program that takes a text message (i.e., a secret message) and an image as input and generates a new image that embeds the input text message as output (i.e., the input image + the secret message). The memory dump files are created during the execution before the steg program creates the output image file.

You, as a forensic investigator, are asked to *understand what was the secret message* that the program was trying to embed from the dump.

Unfortunately, from merely looking at memory dump files, you realized that you couldn't find easy hints (e.g., interesting strings).
Now, you look at the binary program to understand why the memory dump files do not have explicit hints for the secret message. You realize that the program transforms the secret input message into a particular data structure, described below.

After analyzing the binary, you make the following observations:

- Steg reads the message and transforms the message into the **BUFENC** structure. During the transformation, steg erases the original input (i.e., the secret message in ASCII code). This is why the memory dump files do not have hints for the secret message in texts (i.e., ASCII code).

The code works as follow:

```
...
BUFENC* p = str2ptr(input);
// Where the memory dumps were obtained. You also need to inject the memory dump here to
replace “p”.
size_encrypt(size_txt,fp1,fp3);
stega_encrypt(fp1,fp3, p, size_txt); // you need to replace p with the injected memory
...
```

    - The code snippet is given to make your analysis easy. Note that in practice, you will not have access to such information (i.e., source code of the malware).
- Steg embeds the secret message into the input image through a function **stega_encrypt** that takes the **BUFENC** structure as input. By replacing the buffer containing **BUFENC**, you can change the secret message which will be embedded into the image.

From the above observations, you have the following idea:

- As the memory dump files contain the buffer of **BUFENCs** that contains the secret message we want to reveal, you want to inject the buffer into the existing binary program and let the program (steg) create a new image file.
- By analyzing the program execution, i.e., how the **BUFENC** structure is processed and originated from the original input, you would be able to infer the original hidden message.

Summary and Hints:

- You are given (1) **steg** binary program, (2) memory dump files.
    - ./steg.out: binary file
    - ./memory dump/*.dump files.
        - The filename format is "pid-starting_address-ending_address". For instance 2824-7fb2dd1eb000-7fb2dd1ed000.dump means that this is a memory dump for a process with pid 2824 and it is the dump of the memory block starting from 7fb2dd1eb000 and ending at 7fb2dd1ed000.
- The steg binary program is compiled by gcc. You can use any debuggers to understand it.
- You are asked to inject the memory content from memory dumps right after **str2ptr** function.
    - The implementation of str2ptr is provided: "str_to_ptr.cpp".
    - *Note that in practice, no such hints will be provided. So, please do not ask further about this file.*
- You are asked to change the argument of the **stega_encrypt** function which is taking a pointer of the **BUFENC** structure.
    - If you simply load the memory buffer and injects it without fixing the pointers included in the memory dump, the program will crash (e.g., segfault).
    - Your task is essentially fixing this error by changing the values in the memory dump files.
- The **steg** program is based on https://github.com/bapzz/Steganography-In-C
    - However, I made changes to the program. So, do not assume that all functions are identical.
    - *Note that in practice, no such hints will be provided. So, please do not ask further about this github project.*

What to do:

1. Download the Pin and install (i.e., compile) it – check the lecture slides for this.
2. Download the steg binary program and memory dump files.
3. Make your own pintool to (1) load the content of the memory dump files, (2) fix the pointers in the buffer so that the program would not crash with the injected contents from the memory dump files.
4. Obtain the output image file that embeds the secret message.
5. Recover the hidden message.
6. Write the report as described below.

Do not do:

1. Since the program uses a typical steganography algorithm covered in the class (also available in the public GitHub repo), it is possible to decode the code by simply analyzing the binary without developing your own pin tool. However, this is not an acceptable answer. You must use your pintool to inject the content from memory dumps to create the output image file.
2. Answers that do not leverage your pintool will not be considered.

Guide:

1. Read this document carefully. Seriously, it will save a lot of time.
2. When the program crashes, you would like to check where it crashes and why. To do so, logging the executed instructions and relevant registers' values would be particularly useful.
3. Stay tuned for more hints (will be released next week).

What to submit?

1. Your Pintool code. (**Submit a single .cpp file please**)
2. A report that includes
   (1) high-level descriptions of how your pintool works (around, **0.5 page**) (20%),
   (2) how did you inject the contents from memory buffer and find the correct offset that points to the buffer that holds instances of the **BUFENC** structure (around **0.25 page**) (15%),
   (3) why the program crashes after you inject the memory, how did you fix (around **0.5 page**) (30%)
   (4) what is the secret message and how did you recover? (around **1 page** with the details such as instructions for the activities) (35%),

- Please submit only two files: **(1) .cpp file**, and **(2) .pdf file**.

Extra credit:
Pin is a dynamic instrumentation tool. However, there are also static binary instrumentation tools that directly rewrite the instructions in the program such as PEBIL. You can use PEBIL or manually rewrite the binary by yourself to achieve the same effect as Pin. One possible solution would be you rewrite the binary program to inject a library call

that you can intercept with LD_PRELOAD trick and implement your instrumentation in your hook library. This is challenging (because there is little to no guidance). Hence, if you do so, you will get 50 out of 100 extra credit.