

Prerequisites

- (a) Disabling address space randomization

```
$ sudo sysctl -w kernel.randomize_va_space=0
```

- (b) Building the vulnerable x64 C++ program

```
$ g++ -o prog prog.cpp
```

- (c) Running the program

```
$ ./prog
```

Tasks

Your tasks are to explore the given C++ program and analyze it for potential heap vulnerabilities (Task 1); then, in Task 2 and Task 3, you will exploit some of these vulnerabilities.

Set the `BUF_SIZE` to be `200+x`, where `x` is the least significant two digits in your SFU ID. If these two digits are zeros, choose the next significant two digits. No other modification to the code is permitted.

Note 1:

GEF has numerous commands for heap analysis that you may find helpful:

<https://hugsy.github.io/gef/commands/heap/>

Note 2:

You may find it useful to *demangle* C++ symbols in GDB by running:

```
gef> set print asm-demangle on
```

Note 3:

If you cannot fully complete a task, include information and screenshots regarding what you have tried and the results. You may receive partial points for your attempt.

Task 1: Analyzing the Code and Program [10%]

Please answer the following questions. These answers will also help you with the rest of the tasks.

- A) Are there any dangling pointers that could result in double-free memory problems? If so, where?
- B) What is the size of a User / Admin object in the memory?
- C) Which *free bin* does a User/Admin object memory chunk go to after it is released? Why?
- D) What is the address of the start of the virtual table (vtable)?
- E) Which memory segment is this vtable located in? What are the permissions of this segment?
- F) What is the structure of this vtable?

Task 2: Double Free [40%]

Subtask 2.A) Your task is to identify a double free vulnerability in the code and exploit it to get arbitrary read access to the memory. Then, to demonstrate your success, you need to make the program print the `secret_key` on the screen. Explain the steps that you took and include the required GDB/GEF screenshots for each step to support your claim. Finally, you should also include a screenshot to show the successful print of the full `secret_key` string.

Hint 1:

After an object is de-allocated, the corresponding freed heap address range (chunk) is released to one of the *free bins*. If an attacker can somehow craft a payload that allocates the same freed heap chunk, an existing double-free vulnerability can be exploited.

Hint 2:

Recall that in C++ a destructor is called when the memory is being released. If this destructor happens to print something, can it be fooled to print something else instead?

Hint 3:

In C++, `std::string` is a class. If the text string is small enough, the string object can hold it internally without the need to allocate extra dedicated heap memory. This technique is called [*Small String Optimization*](#). It may help to familiarize yourself with how this object is stored in the memory.

Subtask 2.B) Can this double-free vulnerability be used to defeat ASLR? Explain.

Task 3: Use After Free (UAF) [50%]

In this task, we try to exploit a use-after-free vulnerability. We use this to control the flow of the program. For each subtask, explain the steps that you took and include the required GDB/GEF screenshots to support your claim. Finally, include screenshots to demonstrate the successful exploit.

Subtask 3.A) Your task is to open the shell by taking advantage of the existing `open_shell()` function. For this part, we try to directly call this function by exploiting the virtual table (vtable) and virtual pointer (vptr). Successful exploitation should result in opening a shell.

Hint 1:

Take another look at *Hint 1* for Task 2.

Hint 2:

You need to build a fake virtual table as part of your payload. If done correctly, the control flow of the program can be hijacked when, for example, `read_pass_file()` is called.

Subtask 3.B) For this part, we try to achieve the same task (i.e., opening the shell) but this time by calling the private `admin_access()` method. Successful exploitation should result in opening a shell.

Hint 1:

You may have noticed that the code has explicitly disabled the admin access by hardcoding the `admin_disabled` value to 1. As such, this value needs to be overwritten to activate access.

Submission

You need to submit:

- (1) All source code files that you developed and all the payloads that you have generated.
- (2) A detailed lab report.

The files should be compressed in a single (.zip) archive. The code should compile and run without any errors.