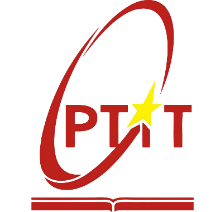
**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**

**KHOA AN TOÀN THÔNG TIN**

-------🙞🙜🕮🙞🙜-------



**Báo cáo bài thực hành**

**Phân tích và khắc phục lỗi trong lập trình C/C++ với GDB**

sinh viên thực hiện:

B20DCAT59 Nguyễn Cảnh HIếu

Giảng viên hướng dẫn: TS.Phạm Hoàng Duy

MỤC LỤC

[MỤC LỤC 1](#_Toc185192520)

[DANH MỤC CÁC HÌNH VẼ 2](#_Toc185192521)

[DANH MỤC CÁC BẢNG BIỂU 2](#_Toc185192522)

[1.1 Giới thiệu chung về bài thực hành 3](#_Toc185192523)

[1.2 Nội dung và hướng dẫn bài thực hành 3](#_Toc185192524)

[1.2.1 Mục đích 3](#_Toc185192525)

[1.2.2 Yêu cầu đối với sinh viên 3](#_Toc185192526)

[1.2.3 Nội dung thực hành 3](#_Toc185192527)

[1.3 Phân tích yêu cầu bài thực hành 4](#_Toc185192528)

[1.4 Thiết kế bài thực hành 4](#_Toc185192529)

[1.5 Cài đặt và cấu hình các máy ảo 6](#_Toc185192530)

[1.6 Tích hợp và triển khai 7](#_Toc185192531)

[***1.6.1*** Docker Hub 7](#_Toc185192532)

[***1.6.2*** Github 8](#_Toc185192533)

[1.7 Thử nghiệm và đánh giá 8](#_Toc185192534)

[TÀI LIỆU THAM KHẢO 15](#_Toc185192535)

DANH MỤC CÁC HÌNH VẼ

[Hình 1 Giao diện Labedit của bài lab 6](#_Toc186667698)

[Hình 2 Cài đặt phần Result 6](#_Toc186667699)

[Hình 3 Dockerfiles của máy gdb-cpp 7](#_Toc186667700)

[Hình 4 Treataslocalcủa gdb-cpp 7](#_Toc186667701)

[Hình 5 Đẩy các vùng chứa lên dockerhub 7](#_Toc186667702)

[Hình 6 Đã đẩy lên docker 8](#_Toc186667703)

[Hình 7 IP của máy gdb-cpp 8](#_Toc186667704)

[Hình 8 Nội dung file stack2.c-1 9](#_Toc186667705)

[Hình 9 Nội dung file stack2.c-2 9](#_Toc186667706)

[Hình 10 Biên dịch stack2.c 9](#_Toc186667707)

[Hình 11 Checkwork nhiệm vụ 1 10](#_Toc186667708)

[Hình 12 Phân tích gỡ lỗi với gdb 10](#_Toc186667709)

[Hình 13 Đặt break ở các hàm 11](#_Toc186667710)

[Hình 14 breakpoint số 1 11](#_Toc186667711)

[Hình 15 Kiểm tra giá trị input và buffer 11](#_Toc186667712)

[Hình 16 Giá trị buffer sau khi sử dụng “strcpy(buffer,input);” 12](#_Toc186667713)

[Hình 17 Sửa lỗi Buffer Overflow 12](#_Toc186667714)

[Hình 18 Checkwork nhiệm vụ 2 12](#_Toc186667715)

[Hình 19 Breakpoint số 2 13](#_Toc186667716)

[Hình 20 Phát hiện lỗi array out of bounds 13](#_Toc186667717)

[Hình 21 Checkwork nhiệm vụ 3 14](#_Toc186667718)

[Hình 22 Breakpoint số 3 14](#_Toc186667719)

[Hình 23 Phát hiện lỗi use after free 14](#_Toc186667720)

[Hình 24 Checkwork nhiệm vụ 4 15](#_Toc186667721)

DANH MỤC CÁC BẢNG BIỂU

[Bảng 1. Bảng Result 5](#_Toc186668380)

* 1. Giới thiệu chung về bài thực hành

Bài thực hành " Phân tích và khắc phục lỗi trong lập trình C/C++ với GDB " được thiết kế nhằm giúp sinh viên hiểu rõ hơn về một số lỗ hổng phổ biến trong ngôn ngữ C/C++ và nắm vững cách sử dụng GDB để phân tích phát hiện lỗ hổng. Bài thực hành này tập trung vào việc phát hiện, phân tích và khắc phục các lỗi phổ biến trong lập trình C, liên quan đến vùng nhớ và quản lý bộ nhớ. Với sự hỗ trợ của công cụ gỡ lỗi GDB, sẽ tìm hiểu nguyên nhân và giải quyết các vấn đề như Segmentation Fault, Buffer Overflow, và Array out of Bounds. Lập trình C/C++ là ngôn ngữ mạnh mẽ nhưng yêu cầu quản lý bộ nhớ trực tiếp, điều này dễ dẫn đến các lỗi khó tìm và gây ảnh hưởng lớn đến hiệu năng, bảo mật của chương trình. Thông qua bài thực hành, bạn sẽ có cơ hội hiểu sâu hơn về cách chương trình hoạt động ở mức thấp, đồng thời học cách sử dụng GDB để gỡ lỗi hiệu quả.

* 1. Nội dung và hướng dẫn bài thực hành
     1. Mục đích

Hiểu rõ về một số lỗi liên quan đến vùng nhớ trong lập trình C/C++. Làm quen với các chức năng cơ bản và nâng cao của GDB. Thực hành phân tích lỗi thông qua việc kiểm tra stack, heap, và giá trị các biến trong chương trình. Áp dụng các kỹ thuật gỡ lỗi để viết mã an toàn và hiệu quả hơn.

* + 1. Yêu cầu đối với sinh viên

Có kiến thức cơ bản về hệ điều hành Linux, GDB, lập trình C/C++.

* + 1. Nội dung thực hành
* Khởi động bài lab:

Vào terminal, gõ:

*Labtainer* *oss\_an\_gdb\_n3\_hieu\_1 -r*

*(chú ý: sinh viên sử dụng mã sinh viên của mình để nhập thông tin email người thực hiện bài lab khi có yêu cầu, để sử dụng khi chấm điểm)*

Sau khi khởi động xong sẽ có terminal ảo **gdb-cpp** xuất hiện. Thực hiện đọc mã nguồn c, rồi tiến hành biên dịch với gcc “*gcc -g -fno-stack-protector -o stack2 stack2.c”* cần có tùy chọn *-fno-stack-protector* để vô hiệu hóa cơ chế bảo vệ ngăn xếp. tiếp theo dùng gdb để phát hiện các lỗ hổng và sửa lại code trong mã nguồn.

* Kết thúc bài lab:

Trên terminal đầu tiên sử dụng câu lệnh sau để kết thúc bài lab:

*stoplab oss\_an\_gdb\_n3\_hieu\_1 -r*

Khi bài lab kết thúc, một tệp zip lưu kết quả được tạo và lưu vào một vị trí được hiển thị bên dưới stoplab.

* Khởi động lại bài lab:

Trong quá trình làm bài sinh viên cần thực hiện lại bài lab, dùng câu lệnh:

*startlab oss\_an\_gdb\_n3\_hieu\_1 -r*

* 1. Phân tích yêu cầu bài thực hành

Bài thực hành gồm một máy ảo gdb-cpp. Để hoàn thành bài thực hành cần thực hiện 4 nhiệm vụ chính. Cần đọc hiểu và biên dịch mã nguồn C có sẵn ở trong máy(stack2.c), tiếp theo sử dụng GDB để thử hiện phân tích và phát hiện các lỗi hổng Buffer Overflow, Array Out Of Bounds và Segmentation Fault.

* 1. Thiết kế bài thực hành

Trên môi trường máy ảo Ubuntu được cung cấp, sử dụng docker để tạo ra một container: container mang tên “**gdb-cpp**” là một máy tính ảo chạy hệ điều hành Linux đóng vai trò chính trong bài thực hành.

Cấu hình docker gồm có:

* Config: lưu cấu hình hoạt động của hệ thống
* Dockerfiles: mô tả cấu hình của container:
* Các nhiệm vụ cần phải thực hiện để thực hành thành công:
* Thực hiện đọc và biên dịch mã nguồn c “*gcc -g -fno-stack-protector -o stack2 stack2.c*”
* Tiếp theo, thực hiện phân tích và phát hiện lỗi với gdb. Cần đặt break point ở các hàm trong mã nguồn “*break buffer\_overflow\_function*”, “*break array\_out\_of\_bounds\_function*”, “*break use\_after\_free\_function*”

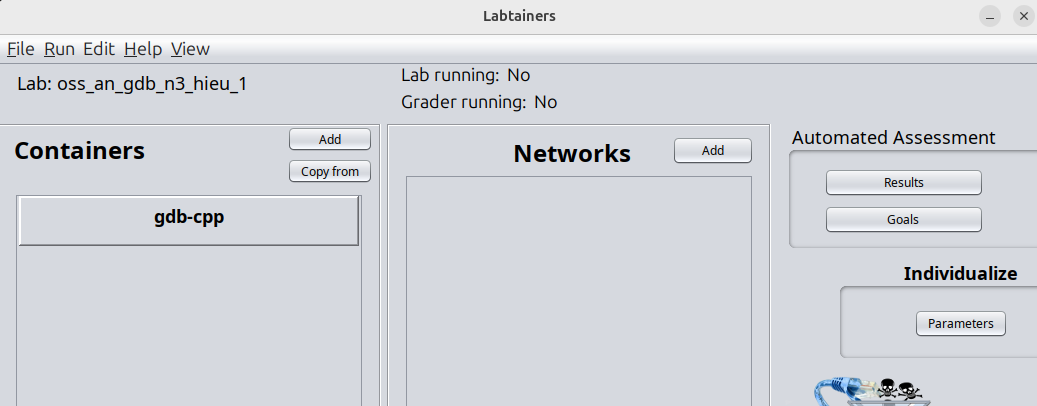
Kết thúc bài lab và đóng gói kết quả.

Để đánh giá được sinh viên đã hoàn thành bài thực hành hay chưa, cần chia bài thực hành thành các nhiệm vụ nhỏ, mỗi nhiệm vụ cần phải chỉ rõ kết quả để có thể dựa vào đó đánh giá, chấm điểm. Do vậy, trong bài thực hành này hệ thống cần ghi nhận các thao tác, sự kiện được mô tả và cấu hình như bảng 1:

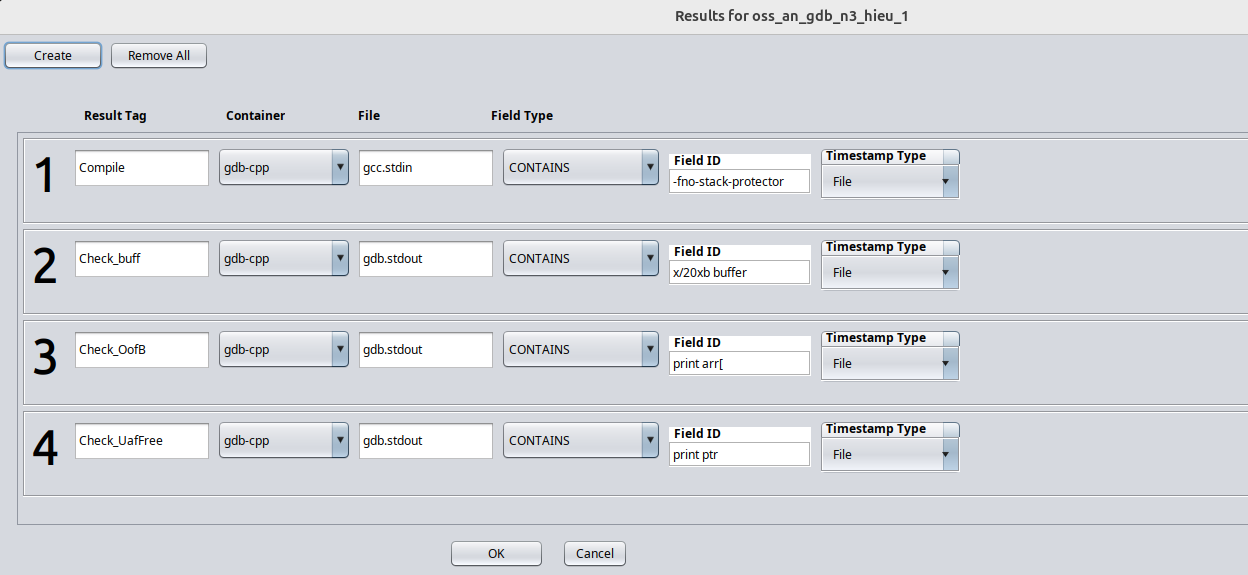
1. Bảng Result

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Result Tag | Container | File | Field Type | Field ID | Timestamp Type |
| Compile | gdb-cpp | gcc.stdin | CONTAINS | -fno-stack-protector | File |
| Check\_buff | gdb-cpp | gdb.stdout | CONTAINS | x/20xb buffer | File |
| Check\_OofB | gdb-cpp | gdb.stdout | CONTAINS | print arr[ | File |
| Check\_UafFree | gdb-cpp | gdb.stdout | CONTAINS | print ptr[ | File |

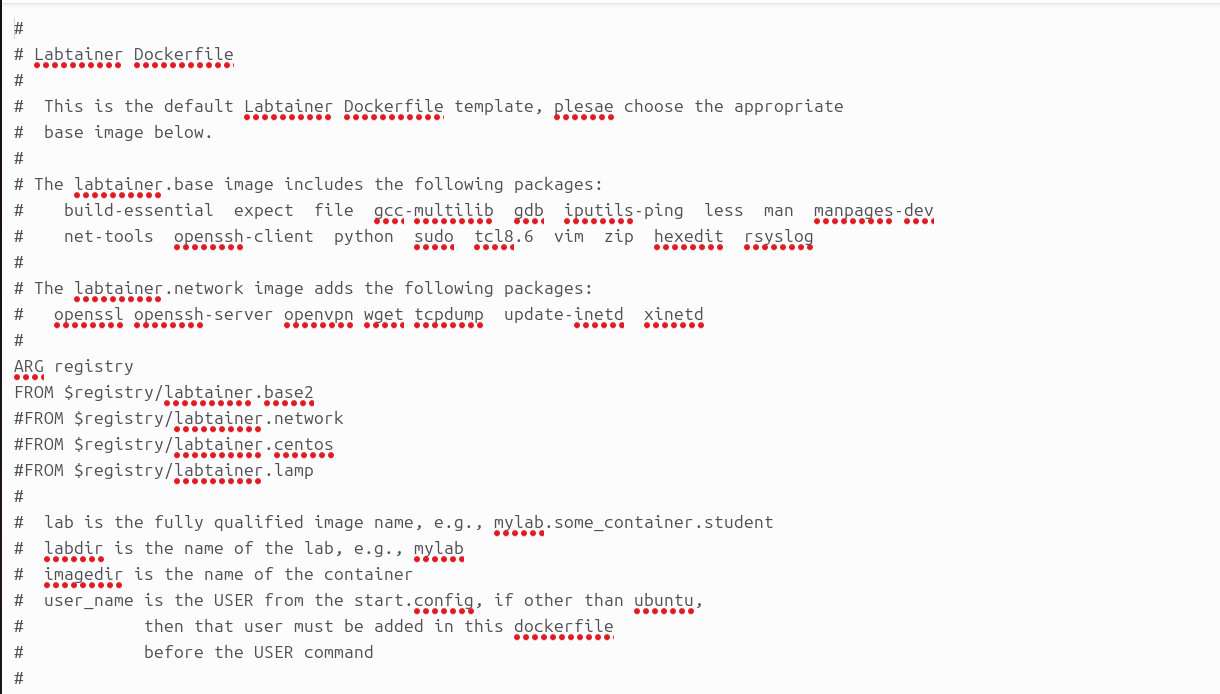
* Compile: Đọc hiểu và biên dịch thành công mã c trong máy gdb-cpp
* Check\_buff: Dùng GDB để phân tích lỗi Buffer Overflow được đặt ở trong hàm buffer\_overflow\_function sau đó tìm cách gỡ lỗi.
* Check\_OofB: Dùng GDB để phân tích lỗi array out of bounds được đặt ở trong hàm array\_out\_of\_bounds\_function sau đó tìm cách gỡ lỗi.
* Check\_UafFree: Dùng GDB để phân tích lỗi use after free được đặt ở trong hàm use\_after\_free\_function sau đó tìm cách gỡ lỗi.
  1. Cài đặt và cấu hình các máy ảo



1. Giao diện Labedit của bài lab



1. Cài đặt phần Result



1. Dockerfiles của máy gdb-cpp



1. Treataslocalcủa gdb-cpp
   1. Tích hợp và triển khai

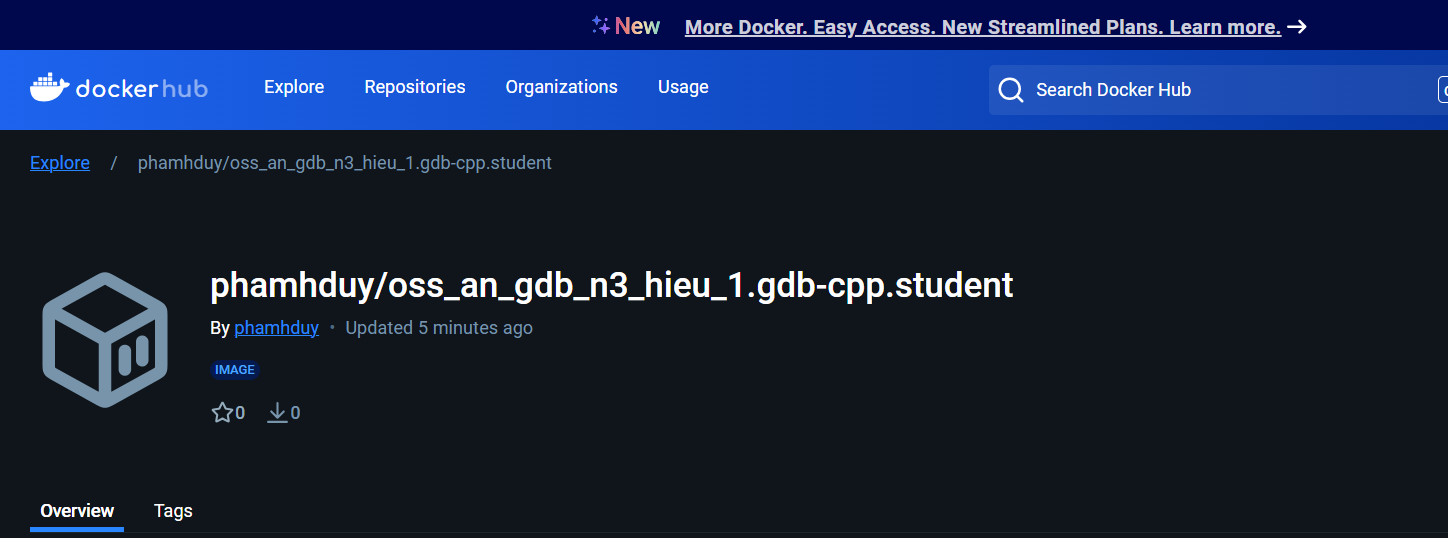
Bài thực hành đã được triển khai như sau:

***Docker Hub***

https://hub.docker.com/r/phamhduy/oss\_an\_gdb\_n3\_hieu\_1.gdb-cpp.student



1. Đẩy các vùng chứa lên dockerhub

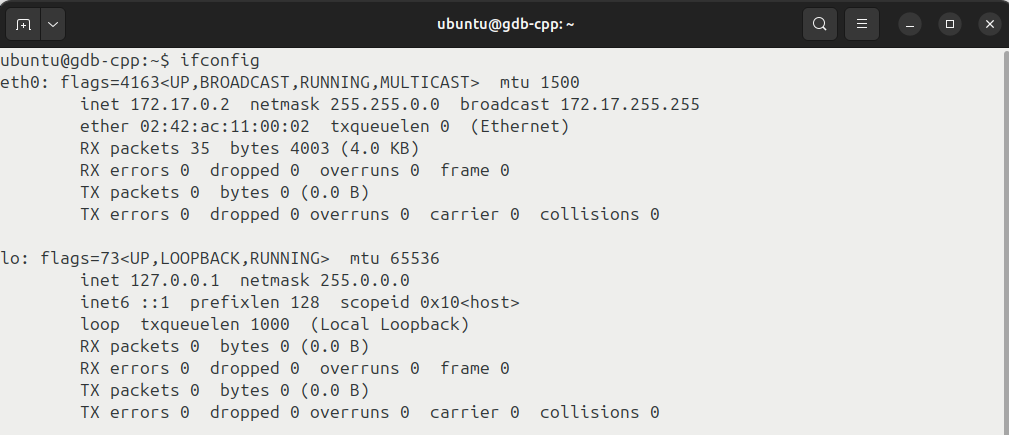


1. Đã đẩy lên docker

***Github***

* 1. Thử nghiệm và đánh giá

Bài thực hành đã được xây dựng thành công, dưới đây là hình ảnh minh họa về bài thực hành:



1. IP của máy gdb-cpp

* ***Nhiệm vụ 1:*** Đọc hiểu chức năng mã nguồn c được lưu ở stack2.c và tiến hành biên dịnh bằng gcc



1. Nội dung file stack2.c-1



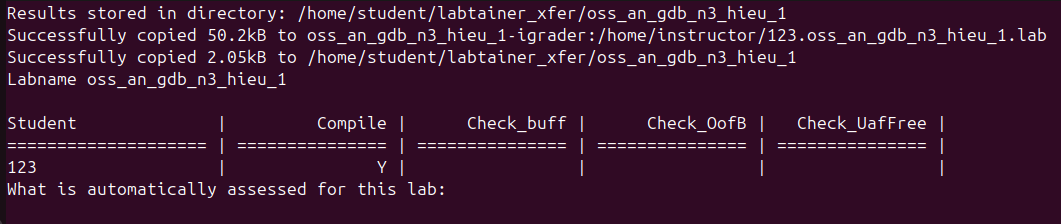
1. Nội dung file stack2.c-2

Thực hiện biên dịch file stack2.c với gcc: *gcc -g -fno-stack-protector -o stack2 stack2.c*



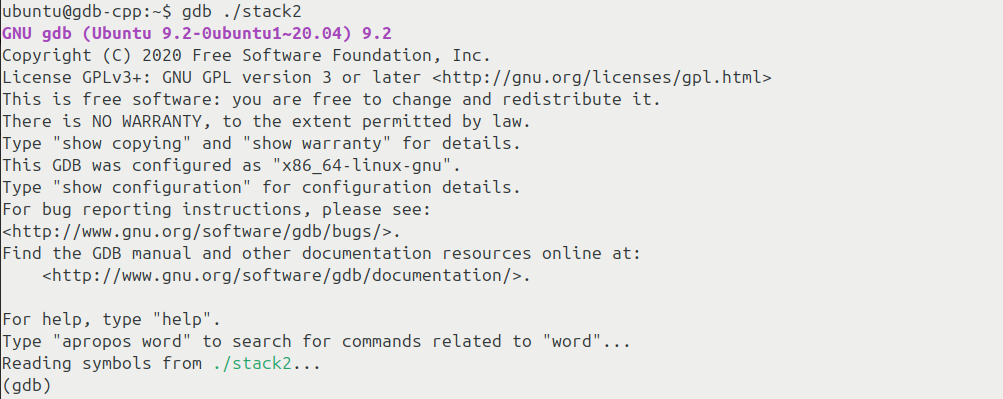
1. Biên dịch stack2.c

* -g: Tùy chọn này để yêu cầu trình biên dịch bao gồm thông tin gỡ lỗi trong tệp thực thi.
* -fno-stack-protector: Tùy chọn này vô hiệu hóa bảo vệ ngăn xếp. Bảo vệ ngăn xếp là một tính năng bảo mật giúp ngăn chặn tràn bộ đệm ngăn xếp.
* -o stack2: Tùy chọn này chỉ định tên của tệp đầu ra. Trong trường hợp này, chương trình biên dịch sẽ được đặt tên là stack2.



1. Checkwork nhiệm vụ 1

Tiếp theo thực hiện phân tích và gỡ lỗi với gdb

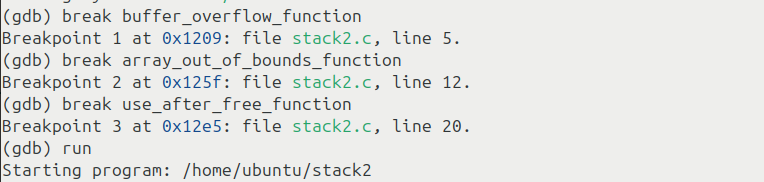


1. Phân tích gỡ lỗi với gdb

Đặt break point theo thứ tự ở các hàm trong mã C.

* *break buffer\_overflow\_function*
* *break array\_out\_of\_bounds\_function*
* *break use\_after\_free\_function*

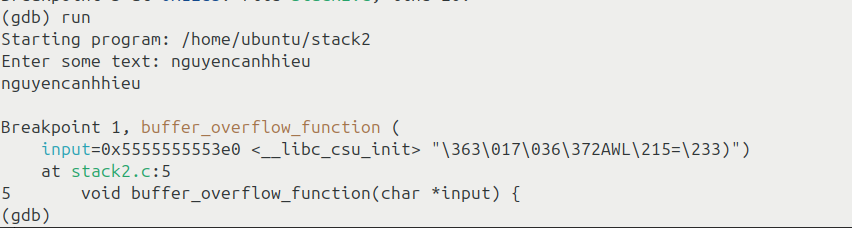
Thực hiện chạy chương trình bằng *run*



1. Đặt break ở các hàm

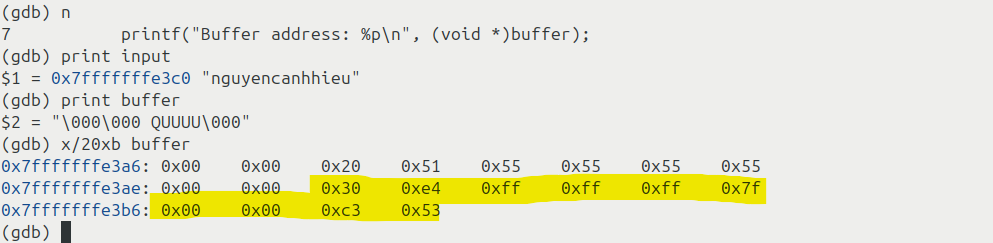
* ***Nhiệm vụ 2:*** Phân tích, gỡ lỗi Buffer Overflow

Nhập chuỗi đầu vào cho chuỗi input. Cần nhập trên 10 ký tự để có thể phát hiện lỗi tràn bộ đệm, sau đó chương trình sẽ dùng lại ở breakpoint số 1 hàm buffer\_overflow\_function

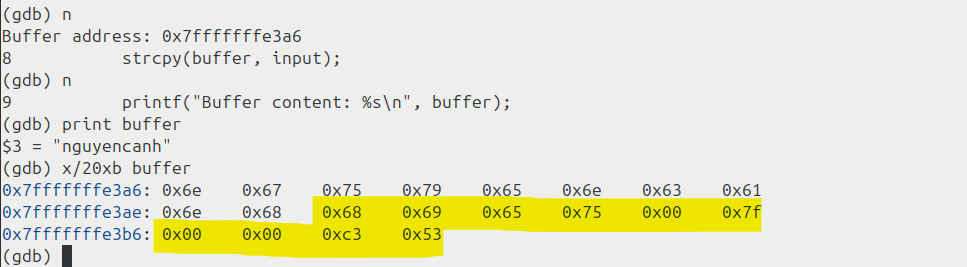


1. breakpoint số 1

Tiếp theo sử dụng lệnh next để đi sâu vào hàm buffer\_overflow\_function, sau đó in giá ra giá trị của input và buffer hiện tại. Sử dụng Lệnh x/20xb buffer để hiển thị 20 đơn vị của bộ nhớ bắt đầu từ địa chỉ của biến buffer.



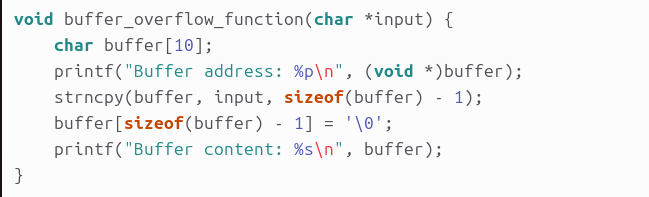
1. Kiểm tra giá trị input và buffer

Buffer được khai báo trong mã nguồn với kích thước là 10. Nên khi sử dụng “*x/20xb buffer*” thì chỉ có 10 đơn vị đầu tiên là giá trị khi mà biến buffer được in ra, cần quan sát 10 đơn vị tiếp theo khi đến bước tiếp theo sử dụng “*strcpy(buffer,input);”* **

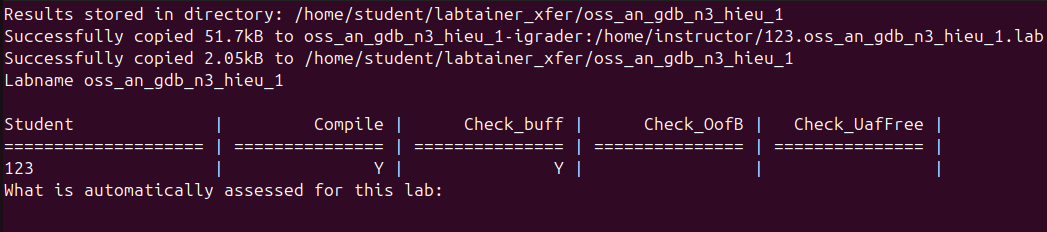
1. Giá trị buffer sau khi sử dụng “strcpy(buffer,input);”

Quan sát thấy khi sử dụng “*print buffer”* thì chỉ in ra tối đa 10 ký tự như đã được khai báo nhưng khi sử dụng “*x/20xb buffer”* dễ dàng thấy được các giá trị từ đơn vị 11 cũng đã thay đổi. Khi dữ liệu ghi vào một mảng (buffer) vượt quá kích thước được cấp phát, nó có thể ghi đè lên các vùng nhớ khác trên ngăn xếp.

Cách sửa lỗi : Thay vì sử dụng strcpy, nên dùng snprintf hoặc strncpy để tránh việc sao chép chuỗi quá dài vào bộ đệm. Snprintf, strncpy đảm bảo không sao chép quá nhiều ký tự vào bộ đệm, bằng cách chỉ sao chép tối đa số ký tự cho phép.



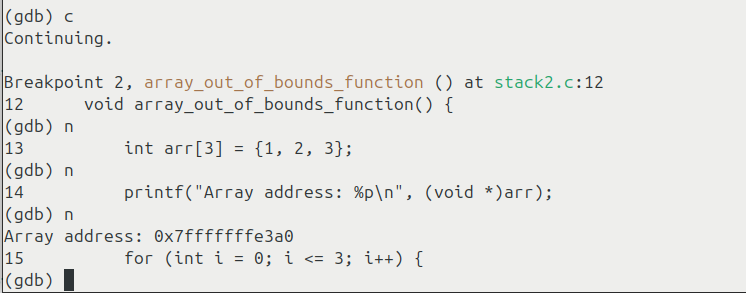
1. Sửa lỗi Buffer Overflow



1. Checkwork nhiệm vụ 2

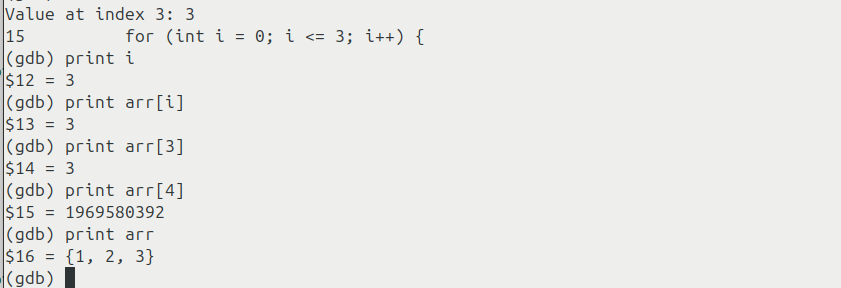
* ***Nhiệm vụ 3:*** phân tích, gỡ lỗi array out of bounds

Sự dụng continue để tiếp tục chương trình đến breakpoint số 2 hàm *array\_out\_of\_bounds\_function* và *next* để đi sâu và trong hàm.

******

1. Breakpoint số 2

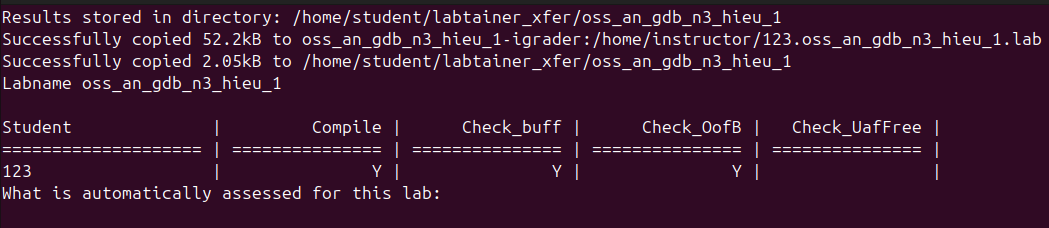
Quan sát thấy mảng arr được khai báo với có 3 phần tử nhưng ở vòng for thì điều kiện i<=3. Tiếp tục sử dụng next cho đến khi i có giá trị bằng 3.



1. Phát hiện lỗi array out of bounds

Có thể thấy mảng arr chỉ có 3 phần tử nhưng khi i có giá trị là 3 là thực hiện print arr[i] thì vẫn nhận được 1 giá trị ngẫu nhiên, tiếp theo thấy tương tự khi print arr[4] cũng nhận được giá trị ngẫu nhiên

Khi cố tình in ra giá trị của một phần tử trong mảng với chỉ số vượt quá giới hạn của mảng (ví dụ arr[3], arr[4] trong trường hợp mảng có kích thước 3), bạn sẽ gặp phải truy cập ngoài giới hạn mảng, gây ra hành vi không xác định (undefined behavior). Để tránh việc truy cập ngoài giới hạn mảng, nên luôn đảm bảo rằng chỉ số mảng luôn nằm trong phạm vi hợp lệ. Kiểm tra kích thước mảng: Trước khi truy cập vào một phần tử mảng, hãy chắc chắn rằng chỉ số bạn sử dụng không vượt quá kích thước mảng.



1. Checkwork nhiệm vụ 3

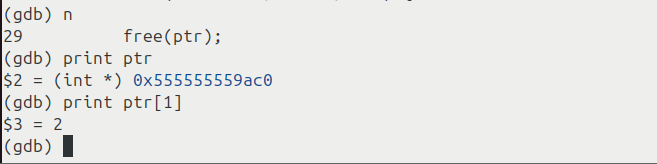
* ***Nhiệm vụ 4:*** phân tích, gỡ lỗi use after free

Sự dụng continue để tiếp tục chương trình đến breakpoint số 3 hàm use\_after\_free\_function và next để đi sâu và trong hàm.



1. Breakpoint số 3

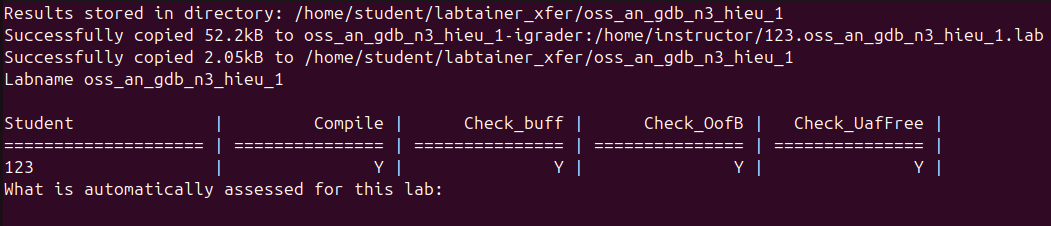
Tiếp tục next cho đến “*free(ptr);”*. Hàm free() trong C được sử dụng để giải phóng bộ nhớ đã được cấp phát động trước đó.



1. Phát hiện lỗi use after free

Có thể thấy sau khi thực hiện giải phóng bộ nhớ nhưng khi thực hiện print ptr[1] thì vẫn có giá trị được in ra màn hình. Khi truy cập vào bộ nhớ đã được giải phóng có thể gây ra lỗi chương trình, Hành vi không xác định, rò rỉ bộ nhớ và lỗi dữ liệu

Cách khắc phục sau khi gọi free(arr), luôn đặt arr = NULL để tránh truy cập ngoài ý muốn.



1. Checkwork nhiệm vụ 4

TÀI LIỆU THAM KHẢO