

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

**BÁO CÁO BÀI THỰC HÀNH**

**Kiểm thử xâm nhập**

**Lỗi vượt giới hạn cấu trúc**

**Giảng viên: Đinh Trường Duy**

**Nhóm lớp: 02**

**Sinh viên: Hoàng Trung Kiên**

**Mã sinh viên: B20DCAT098**

**Hà Nội – 2024**

**Mục lục**

[1. **Mục đích**. 3](#_Toc164409922)

[2. **Yêu cầu đối với sinh viên**. 3](#_Toc164409923)

[3. **Nội dung thực hành**. 3](#_Toc164409924)

[4. **Checkwork**. 10](#_Toc164409925)

# 1. **Mục đích**.

Minh họa việc chạy quá giới hạn dự định của cấu trúc dữ liệu trong một chương trình C.

# 2. **Yêu cầu đối với sinh viên**.

Có kinh nghiệm lập trình C cơ bản và có kiến thức về các cấu trúc dữ liệu đơn giản.

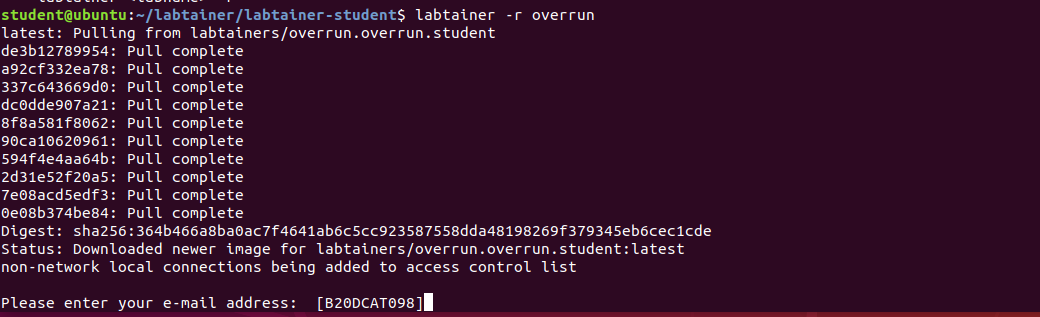
Biết cách biên dịch và thực thi một chương trình C bất kỳ.

Biết cách sử dụng chương trình gỡ lỗi GDB.

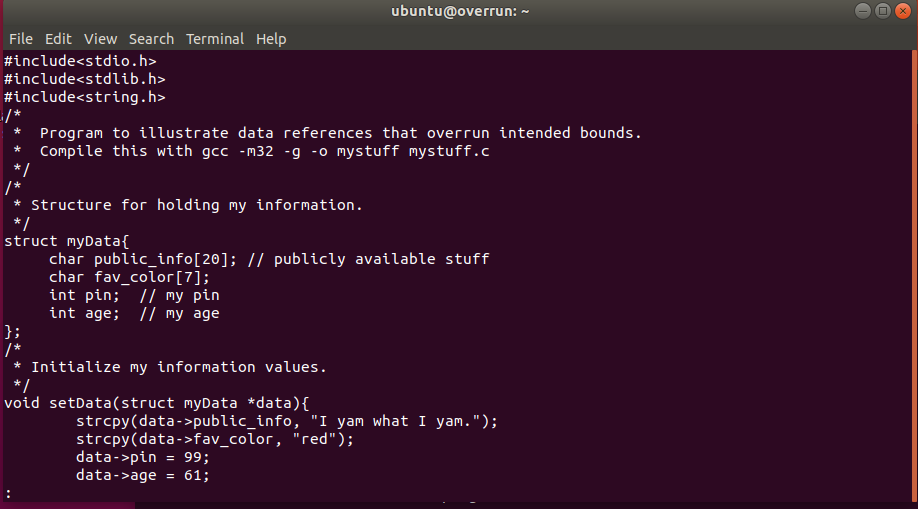
# 3. **Nội dung thực hành**.

Khởi động lab:

Chạy lệnh: labtainer -r overrun trong terminal của Labtainer



Kiểm tra lại code: less mystuff.c

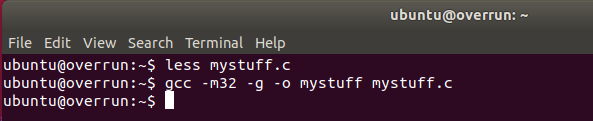


Cấu trúc myData: Nhìn vào struct myData. Trong chương trình khai báo biến my\_data là một struct kiểu myData. Lưu ý rằng mảng ký tự public\_info có 20 phần tử. Ta có thể tham chiếu đến các phần tử của mảng bằng cách sử dụng chỉ mục. Ví dụ: my data.public\_info[4] đề cập đến ký tự thứ 5 trong mảng và my data.public\_info[19] đề cập đến ký tự cuối cùng trong mảng.

Địa chỉ của các trường: Sau khi chương trình khởi tạo biến my\_data kiểu struct, nó sẽ hiển thị địa chỉ của phần bắt đầu trường public\_data và trường pin, đồng thời nó hiển thị các giá trị bộ nhớ của các trường đó

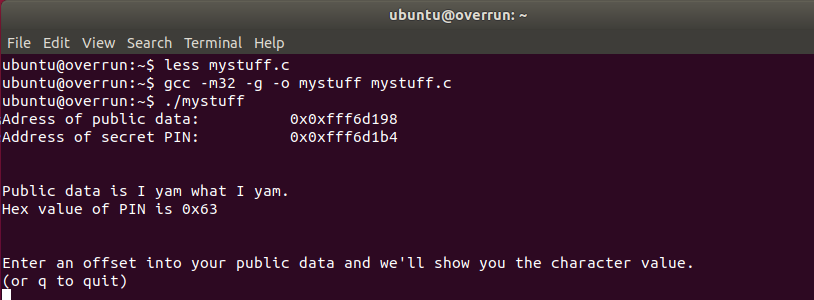
Nội dung bộ nhớ: Chương trình có một vòng lặp cho phép người dùng xem các giá trị hex của các ký tự riêng lẻ trong trường public\_info. Chính vòng lặp này sẽ cho chúng ta khám phá câu hỏi được hỏi trước đó: my\_data.public\_info[20] đề cập đến điều gì?

Biên dịch và chạy chương trình: gcc -m32 -g -o mystuff mystuff.c

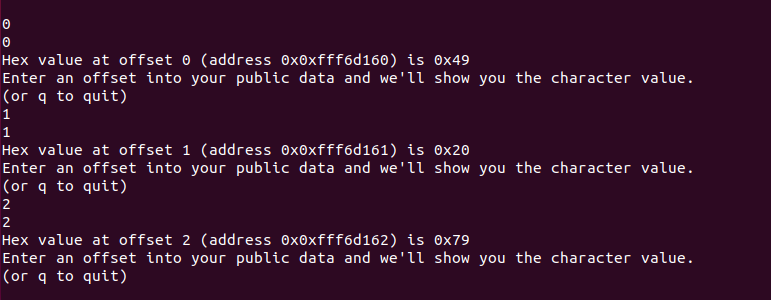


Lưu ý rằng -m32 tạo ra một mã nhị phân 32 bit và -g sẽ chứa các ký hiệu trong file nhị phân, cho phép khám phá quá trình thực thi của chương trình bằng cách sử dụng gdb.

Chạy chương trình: ./mystuff



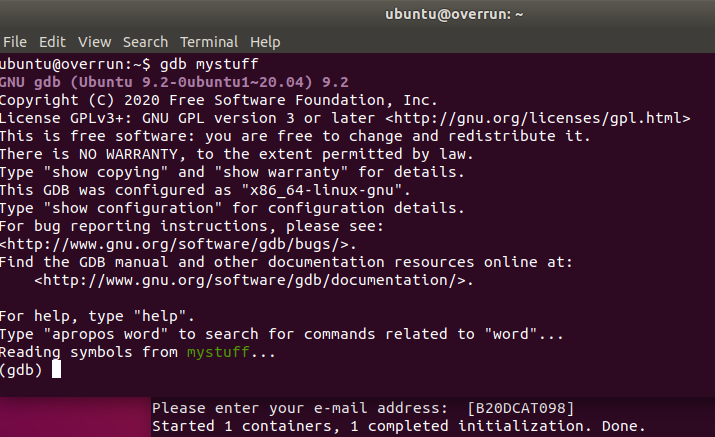
xem các giá trị được hiển thị ở các offset khác nhau trong (và hơn thế nữa) trường public\_info. Lưu ý địa chỉ hiển thị của trường public\_info và địa chỉ của trường pin. Có bao nhiêu byte phân tách hai trường public\_info và pin?



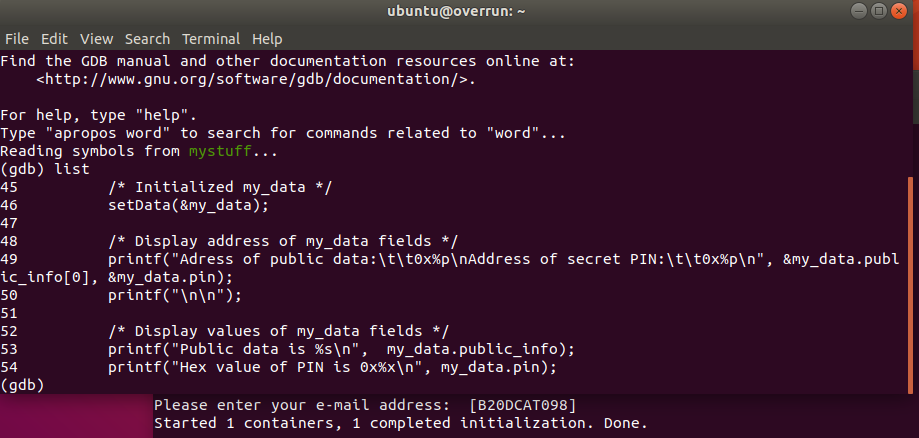
0x0xfffcb7f4 - 0x0xfffcb7c4 = 0x20 = 32 (byte)

Sử dụng chương trình để hiển thị giá trị hex của trường pin. Lưu ý rằng kích thước bộ đệm biến fav\_color là số lẻ thì trình biên dịch sẽ đệm bộ đệm để biến tiếp theo bắt đầu trên ranh giới từ 4 byte

Chạy chương trình trong trình gỡ lỗi GDB: gdb mystuff

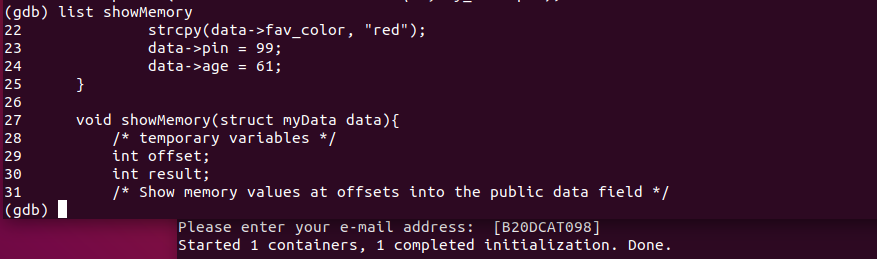


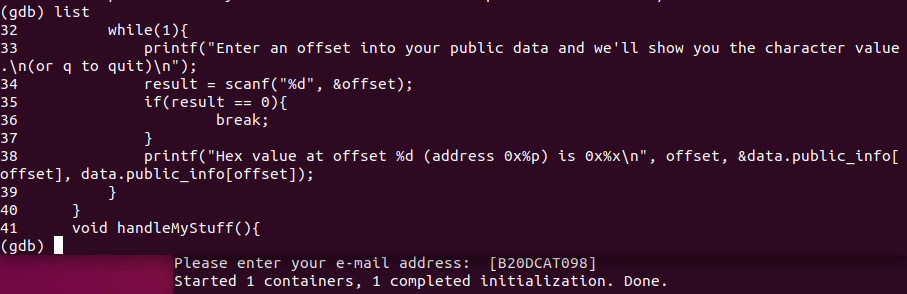
Sử dụng lệnh list để xem mã nguồn.



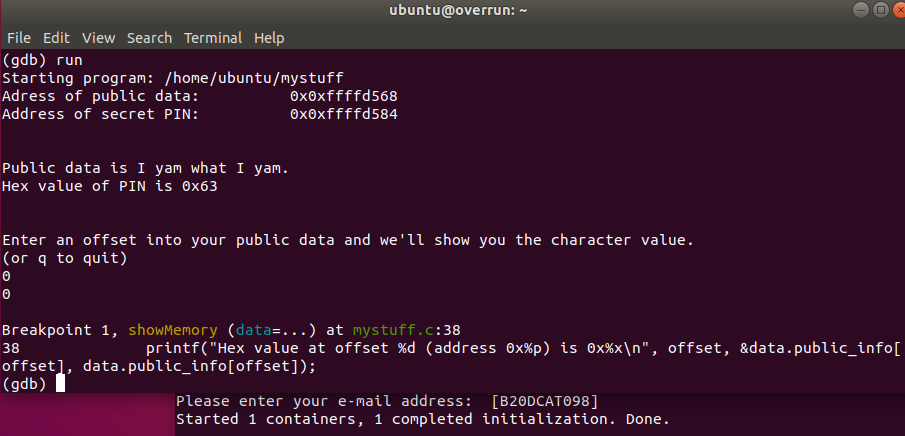
Đặt một điểm ngắt trong hàm showMemory trên dòng in giá trị tại offset đã cho: break <dòng>

Sử dụng list showMemory để xem mã nguồn cho hàm đó

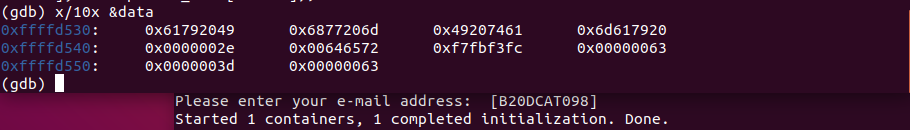




Break 38 và sau đó chạy chương trình từ bên trong gdb: run

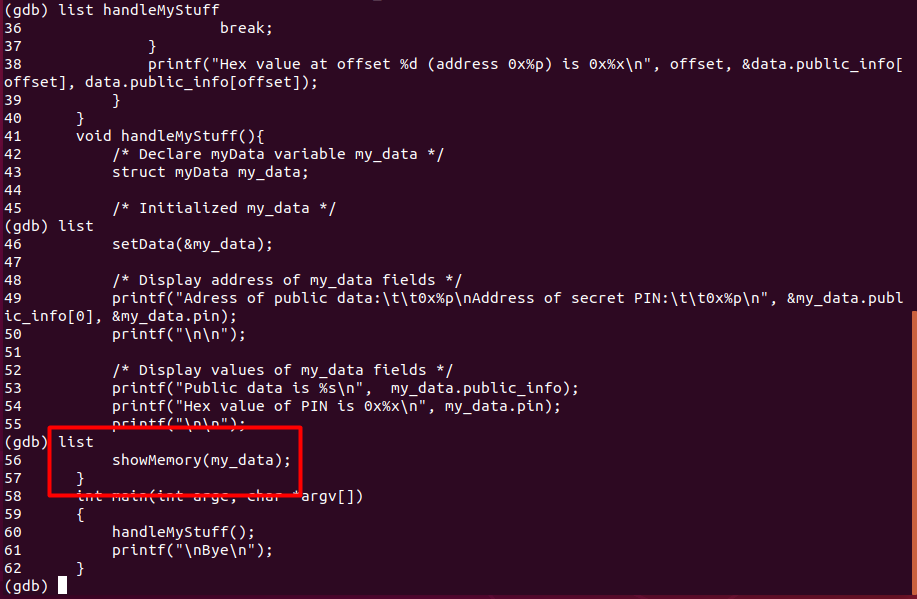


Khi chương trình chạm điểm ngắt, hiển thị 10 word (40 byte) trong bộ nhớ hệ thống dưới dạng giá trị hex bắt đầu từ cấu trúc dữ liệu: x/10x &data

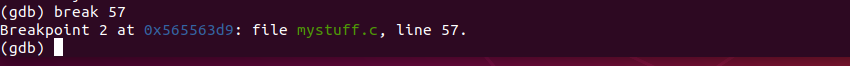


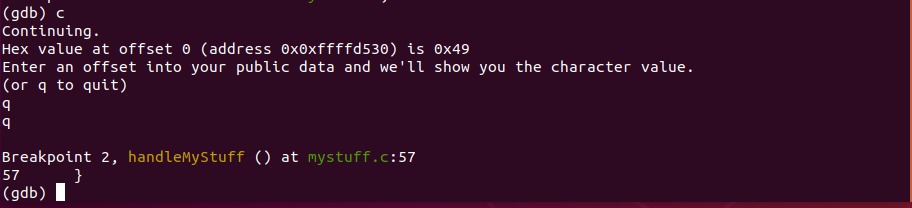
Nội dung bộ nhớ có tương ứng với những gì sinh viên đã quan sát trong khi chạy chương trình không?: có

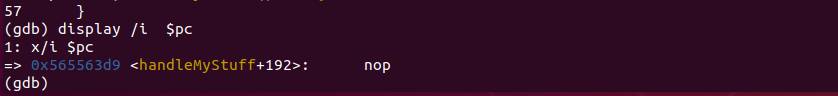
Đặt một điểm ngắt ở cuối hàm handleMyStuff, tức là trên dòng của dấu ngoặc nhọn cuối cùng bên phải (}) trong hàm đó. Sau đó tiếp tục với lệnh c. Tại lời nhắc cho offset tiếp theo, hãy nhập q. Sau đó, khi chương trình chạm điểm ngắt, hãy hiển thị chương trình đã dịch ngược bằng cách sử dụng: display /i $pc stepi

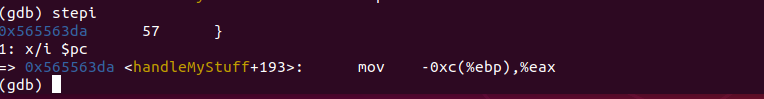


Break 57

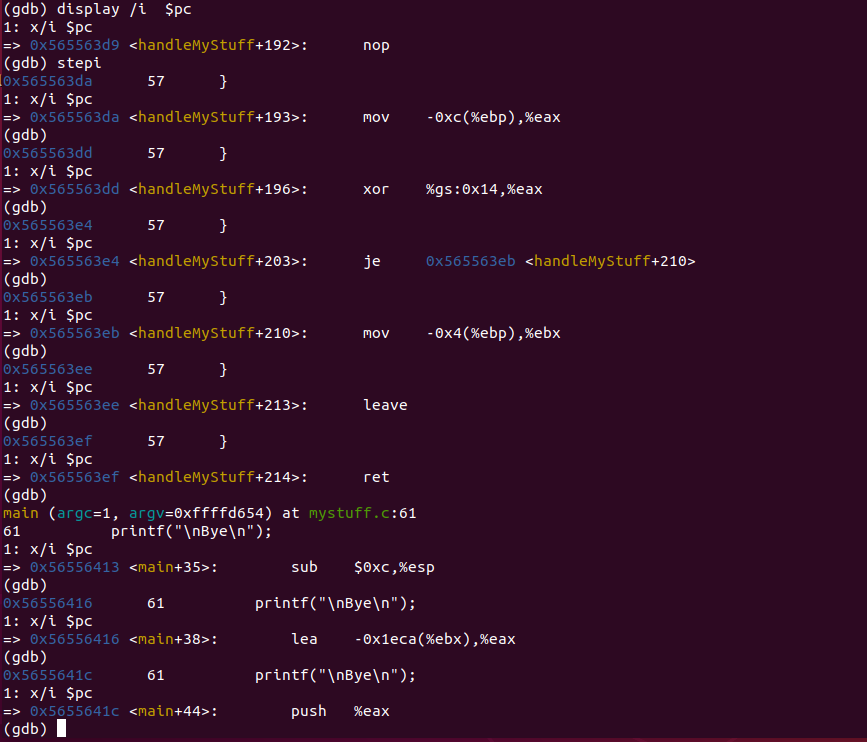








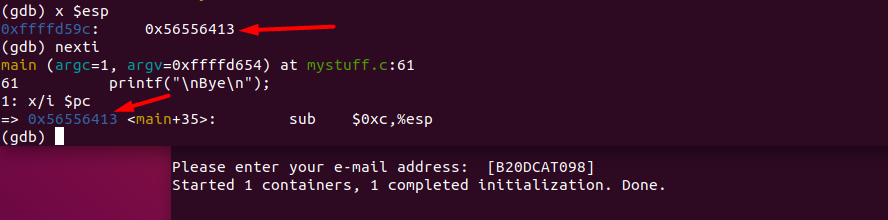
Và từng bước để dịch ngược phần còn lại của hàm *handleMyStuff* bằng cách nhấn liên tục phím *Enter* cho đến khi chương trình chuyển sang lệnh *ret*.



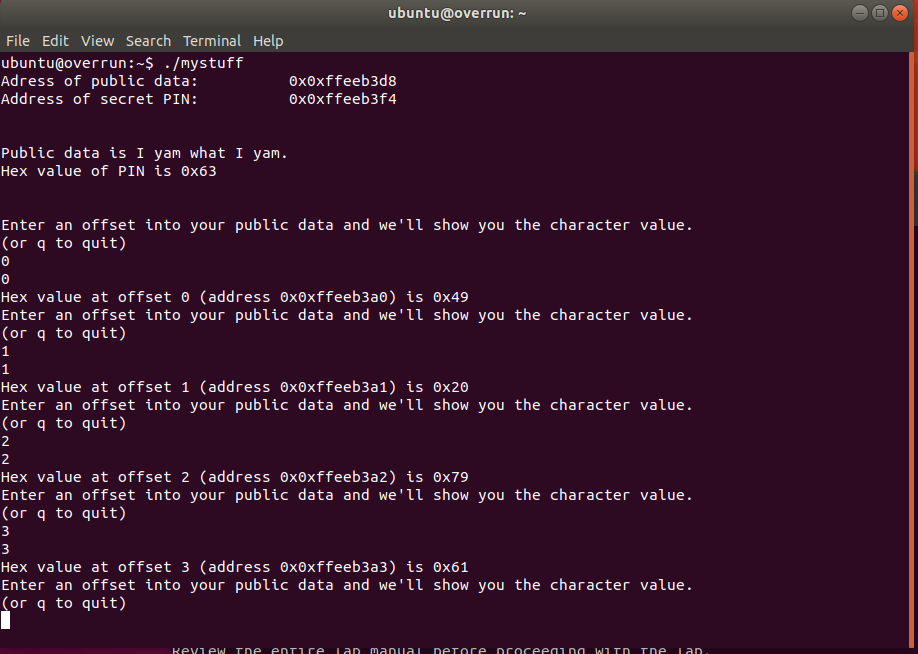
Đây là điểm trong chương trình mà tại đó hàm handleMyStuff sẽ trở lại hàm chính. Lệnh ret chỉ đạo bộ xử lý chuyển đến lệnh tại địa chỉ chứa trong con trỏ ngăn xếp hiện tại. Hiển thị nội dung bộ nhớ được trỏ đến bởi thanh ghi ngăn xếp bằng cách sử dụng:x $esp

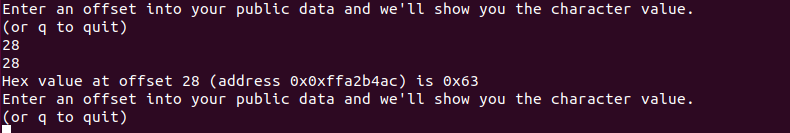


Giá trị được hiển thị sẽ trở thành địa chỉ lệnh tiếp theo, ta có thể xác nhận bằng một nexti nữa. Ghi lại con trỏ lệnh hiện tại. Hãy xem lại địa chỉ ngăn xếp chứa giá trị trả về này. Lưu ý rằng nó cao hơn địa chỉ của cấu trúc dữ liệu được quan sát trong hàm showMemory. Tính toán và ghi lại sự khác biệt giữa hai địa chỉ.



Chạy lại chương trình bên ngoài trình gỡ lỗi và sử dụng nó để hiển thị giá trị địa chỉ trả về, mỗi lần một byte. Xác nhận rằng địa chỉ là những gì sinh viên đã quan sát thấy trong gdb. Tưởng tượng rằng chương trình cho phép chúng ta sửa đổi các mục riêng lẻ trong mảng public\_info. Khi chương trình truy cập vào lệnh ret đã xem trong gdb, nó sẽ quay trở lại địa chỉ đã viết.





# 4. **Checkwork**.

