

**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**  
**KHOA AN TOÀN THÔNG TIN**



**BÁO CÁO THỰC HÀNH**  
**HỌC PHẦN: KIỂM THỬ XÂM NHẬP**  
**MÃ HỌC PHẦN: INT14107**

**BÀI THỰC HÀNH**  
**LỖI VƯỢT GIỚI HẠN CẤU TRÚC DỮ LIỆU**

Họ và tên: Trần Thị Thu Phương

MSV: B21DCAT151

Nhóm lớp: 03

Giảng viên hướng dẫn: TS. Đinh Trường Duy

**HÀ NỘI 2025**

## Khởi động bài lab:

Chạy lệnh: *labtainer -r overrun* trong terminal của Labtainer

```
student@ubuntu:~/labtainer/trunk/scripts/labtainer-student$ labtainer -r overrun
latest: Pulling from labtainers/overrun.overrun.student
de3b12789954: Pull complete
a92cf332ea78: Pull complete
337c643669d0: Pull complete
dc0dde907a21: Pull complete
8f8a581f8062: Pull complete
90ca10620961: Pull complete
594f4e4aa64b: Pull complete
2d31e52f20a5: Pull complete
7e08acd5edf3: Pull complete
0e08b374be84: Pull complete
Digest: sha256:364b466a8ba0ac7f4641ab6c5cc923587558dda48198269f379345eb6cec1cde
Status: Downloaded newer image for labtainers/overrun.overrun.student:latest

Please enter your e-mail address: [B21DCAT151]B21DCAT151
Started 1 containers, 0 completed initialization, please wait...
```

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

- Thực hiện các yêu cầu sau:

## Kiểm tra lại code

Tại terminal mở ra, hãy xem chương trình *mystuff.c*. Sử dụng vi hoặc nano, hoặc chỉ nhập *less mystuff.c*.

```
ubuntu@overrun: ~  
File Edit View Search Terminal Help  
GNU nano 4.8 mystuff.c  
#include<stdio.h>  
#include<stdlib.h>  
#include<string.h>  
/*  
 * Program to illustrate data references that overrun intended bounds.  
 * Compile this with gcc -m32 -g -o mystuff mystuff.c  
 */  
/*  
 * Structure for holding my information.  
 */  
struct myData{  
    char public_info[20]; // publicly available stuff  
    char fav_color[9];  
    int pin; // my pin  
    int age; // my age  
};  
/*  
 * Initialize my information values.  
 */  
void setData(struct myData *data){  
    strcpy(data->public_info, "I yam what I yam.");  
};  
date  
Tran Thi Thu Phuong B21DCAT151  
Tue Apr 29 21:59:53 PDT 2025  
student@ubuntu:~/labtainer/trunk/scripts/labtainer-student$
```

## Cấu trúc myData

```
struct myData{  
    char public_info[20]; // publicly available stuff  
    char fav_color[9];  
    int pin; // my pin  
    int age; // my age  
};  
/*
```

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.

Câu hỏi đặt ra: nếu 19 là ký tự cuối cùng trong mảng, `data.public_info[20]` sẽ tham chiếu đến cái gì?

Trả lời: `data.public_info [20]` không tồn tại hợp lệ. Khi truy cập `data.public_info [20]`, nó sẽ không xác định giá trị hoặc có thể dẫn đến lỗi chương trình hoặc hành vi bất thường.

## 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.

```
void setData(struct myData *data){
    strcpy(data->public_info, "I yam what I yam.");
    strcpy(data->fav_color, "red");
    data->pin = 99;
    data->age = 61;
}
```

## 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?

```
while(1){
    printf("Enter an offset into your public data and we'll show you the character value.\n(or q to quit)\n");
    result = scanf("%d", &offset);
    if(result == 0){
        break;
    }
    printf("Hex value at offset %d (address 0x%p) is 0x%x\n", offset, &data.public_info[offset], data.public_info[offset]);
}
```

Trả lời: *my\_data.public\_info[20]* không nằm trong *public\_info* mà là phần đầu của *fav\_color*. Khi chạy chương trình và nhập offset = 20, ta sẽ thấy giá trị 0x72 ('r' trong ASCII). Điều này minh họa việc truy xuất ngoài phạm vi mảng, có thể dẫn đến lỗi bảo mật nếu không kiểm soát đúng.

```
20
20
Hex value at offset 20 (address 0x0xffe81a54) is 0x72
Enter an offset into your public data and we'll show you the character value.
(or q to quit)
```

## Biên dịch và chạy chương trình

Sử dụng lệnh này để biên dịch 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
```

và 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*.

```
ubuntu@overrun: ~  
File Edit View Search Terminal Help  
ubuntu@overrun:~$ ls  
mystuff.c  
ubuntu@overrun:~$ nano mystuff.c  
ubuntu@overrun:~$ gcc -m32 -g -o mystuff mystuff.c  
ubuntu@overrun:~$ ./mystuff  
Address of public data:      0x0xffffd5a4  
Address of secret PIN:      0x0xffffd5c4  
  
Public data is I yam what I yam.  
Hex value of PIN is 0x63  
  
Enter an offset into your public data and we'll show you the character value.  
(or q to quit)  
█
```

```
student@ubuntu: ~/labtainer/trunk/...  
File Edit View Search Terminal Tabs Help  
student@ubu... x student@ubu... x  
Tran Thi Thu Phuong B21DCAT151  
Tue Apr 29 21:59:53 PDT 2025  
student@ubuntu:~/labtainer/trunk/scri
```

```
ubuntu@overrun:~$ ./mystuff  
Address of public data:      0x0xffe81a78  
Address of secret PIN:      0x0xffe81a94  
  
Public data is I yam what I yam.  
Hex value of PIN is 0x63
```

Có bao nhiêu byte phân tách hai trường *public\_info* và *pin*?

→  $0xffffd5c4 - 0xffffd5a4 = 32$  bytes

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.

### Khám phá với gdb

Chạy chương trình trong trình gỡ lỗi GDB:

*`gdb mystuff`*

Sử dụng lệnh *list* để xem mã nguồn.

```

ubuntu@goverrun:~$ gdb mystuff
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from mystuff...
(gdb) l
45      /* Initialized my_data */
46      setData(&my_data);
47
48      /* Display address of my_data fields */
49      printf("Adress of public data:\t\t0x%p\nAddress of secret PIN:\t\t0x%p\n", &my_data.p
public_info[0], &my_data.pin);
50      printf("\n\n");
51
52      /* Display values of my_data fields */
53      printf("Public data is %s\n", my_data.public_info);
54      printf("Hex value of PIN is 0x%x\n", my_data.pin);
(gdb) l
55      printf("\n\n");
56      showMemory(my_data);
57  }
58  int main(int argc, char *argv[])

```

Đặt một điểm ngắt trong hàm *showMemory* trên dòng in giá trị tại offset đã cho. (Sử dụng list *showMemory* để xem mã nguồn cho hàm đó.)

*break showMemory*

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*

```
(gdb) break showMemory
Breakpoint 1 at 0x1285: file mystuff.c, line 27.
(gdb) run
Starting program: /home/ubuntu/mystuff
Address of public data: 0x0xffffd564
Address of secret PIN: 0x0xffffd584

Public data is I yam what I yam.
Hex value of PIN is 0x63

Breakpoint 1, showMemory (data=...) at mystuff.c:27
27 void showMemory(struct myData data){
(gdb) x/10x &data
0xffffd530: 0x61792049 0x6877206d 0x49207461 0x6d617920
0xffffd540: 0xf7fe002e 0x00646572 0xf7e10212 0xf7fbf3fc
0xffffd550: 0x00000063 0x0000003d
(gdb)
```

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

### Thử nghiệm thêm

Đặt một điểm ngắt ở cuối hàm *handleMyStuff*, tức là trên dòng của dấu ngoặc nhọn cuối cùng bên phải *()* trong hàm đó.

```
(gdb) break handleMyStuff
Breakpoint 4 at 0x56556319: file mystuff.c, line 41.
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/ubuntu/mystuff

Breakpoint 4, handleMyStuff () at mystuff.c:41
41 void handleMyStuff(){
1: x/i $pc
=> 0x56556319 <handleMyStuff>: endbr32
(gdb)
```

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*



```

Breakpoint 1, handleMyStuff () at mystuff.c:41
41      void handleMyStuff(){
(gdb) display /i $pc
1: x/i $pc
=> 0x56556319 <handleMyStuff>: endbr32
(gdb) stepi
0x5655631d      41      void handleMyStuff(){
1: x/i $pc
=> 0x5655631d <handleMyStuff+4>:      push    %ebp
(gdb)
0x5655631e      41      void handleMyStuff(){
1: x/i $pc
=> 0x5655631e <handleMyStuff+5>:      mov     %esp,%ebp
(gdb)

```

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*.

```

0x5655631e      41      void handleMyStuff(){
1: x/i $pc
=> 0x5655631e <handleMyStuff+5>:      mov     %esp,%ebp
(gdb)
0x56556320      41      void handleMyStuff(){
1: x/i $pc
=> 0x56556320 <handleMyStuff+7>:      push    %ebx
(gdb)
0x56556321      41      void handleMyStuff(){
1: x/i $pc
=> 0x56556321 <handleMyStuff+8>:      sub     $0x34,%esp
(gdb)
0x56556324      41      void handleMyStuff(){
1: x/i $pc
=> 0x56556324 <handleMyStuff+11>:
    call   0x56556130 <__x86.get_pc_thunk.bx>
(gdb)
0x56556130 in __x86.get_pc_thunk.bx ()
1: x/i $pc
=> 0x56556130 <__x86.get_pc_thunk.bx>: mov     (%esp),%ebx
(gdb)
0x56556133 in __x86.get_pc_thunk.bx ()
1: x/i $pc
=> 0x56556133 <__x86.get_pc_thunk.bx+3>:      ret
(gdb)

```

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. Hiện 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.



```

=> 0x56556329 <__x86.get_pc_thunk.0x+5>:      ret
(gdb) x $esp
0xffffd55c:      0x56556329
(gdb) nexti
0x56556329 in handleMyStuff () at mystuff.c:41
41      void handleMyStuff(){
1: x/i $pc
=> 0x56556329 <handleMyStuff+16>:      add     $0x2ca3,%ebx
(gdb)

```

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* mà sinh viên đã xem trong gdb, nó sẽ quay trở lại địa chỉ mà sinh viên đã viết.

```

ubuntu@overrun:~$ ./mystuff
Address of public data:      0x0xffffd5a4
Address of secret PIN:      0x0xffffd5c4

Public data is I yam what I yam.
Hex value of PIN is 0x63

Enter an offset into your public data and we'll show you the character value.
(or q to quit)
32
32
Hex value at offset 32 (address 0x0xffffd590) is 0x63
Enter an offset into your public data and we'll show you the character value.
(or q to quit)

```

### Kết thúc bài lab:

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

*stoplab overrun*


- Khi bài lab kết thúc, một tệp lưu kết quả được tạo và lưu vào một vị trí được hiển thị bên dưới stoplab.
- Sinh viên cần nộp file *.lab* để chấm điểm.

### Kết quả checkwork:

```
student@ubuntu:~/labtainer/trunk/scripts/labtainer-student$ checkwork
Results stored in directory: /home/student/labtainer_xfer/overrun
Labname overrun

Student      |      gdb_commands      |      showed_pin      |      viewed_ret      |
=====      | =====      | =====      | =====      |
B21DCAT151   |      3      |      Y      |      Y      |
What is automatically assessed for this lab:
  showed_pin: Display the secret PIN by providing an out-of-bounds index
  viewed_ret: Viewed a ret instruction in the disassembly
  gdb_commands: How many gdb commands issued by the student
student@ubuntu:~/labtainer/trunk/scripts/labtainer-student$ echo "Tran Thi Thu Phuong B21DCAT151"; date
Tran Thi Thu Phuong B21DCAT151
Wed Apr 30 03:27:49 PDT 2025
student@ubuntu:~/labtainer/trunk/scripts/labtainer-student$
```

Kết quả nộp trên seclab



Bài tập

Trạng thái

Lịch sử

Bảng xếp hạng

Hướng dẫn

Trần Thị Thu Phương

B21DCAT151

Hồ sơ

Lớp học

Đăng xuất

Lịch sử nộp bài

ID	Thời gian	Bài tập	Kết quả
20900	2025-04-30 17:30:46	Lỗi vượt giới hạn cấu trúc dữ liệu	AC