**Bài thực hành 5: Overrun**

**TÊN\_BÀI\_LAB: overrun**

***Bài thực hành này dựa trên lab overrun trong Labtainer framework của trung tâm Cybersecurity and Cyber Operations, Naval Postgraduate School.***

1. ***Mục đích***

* Bài thực hành này minh họa việc chạy quá giới hạn dự định của cấu trúc dữ liệu trong một chương trình C.
* Sử dụng cho sinh viên đã có kinh nghiệm lập trình ngôn ngữ C cơ bản và đã quen thuộc với các cấu trúc dữ liệu đơn giản. Chương trình GDB được sử dụng để khám phá chương trình đang thực thi, bao gồm cả gỡ lỗi.

1. ***Chuẩn bị thực hành***
   1. *Tìm hiểu lý thuyết*

* Tìm hiểu về gdb
  1. *Tài liệu tham khảo*
  2. *Chuẩn bị môi trường*
* Phần mềm ảo hóa: VMWare Workstation.
* Môi trường Labtainer:
  + Download file ova: <https://nps.box.com/shared/static/2chwo31xgxm2hs4hewp2n4nblroyagwz.ova>
  + Chạy VMWare, rồi: File / Open / LabtainerVM-VMWare.ova
* Khởi động bài lab:
  + Vào terminal, gõ:

*cd ~/labtainer/trunk/scripts/labtainer-student*

*labtainer TÊN\_BÀI\_LAB*

*(chú ý: sinh viên sử dụng email <tên\_tài\_khoản>@stu.ptit.edu.vn của mình để nhập thông tin email người thực hiện bài lab khi có yêu cầu, để sử dụng khi chấm điểm)*

* Dừng bài lab:
  + Vào terminal, tại thư mục:

*~/labtainer/trunk/scripts/labtainer-student*

Gõ:

*stoplab*

* File kết quả bài lab dùng để nộp cùng với file báo cáo nằm trong thư mục:

*/home/student/labtainer\_xfer/TÊN\_BÀI\_LAB*

* Trong trường hợp lỗi máy giữa chừng và cần thực hiện lại bài lab, gõ lệnh:

*labtainer -r TÊN\_BÀI\_LAB*

Nếu có lỗi khác, có thể thử khởi động lại máy ảo và chạy lại.

1. ***Nội dung thực hành***
   1. ***Kiểm tra lại code***

* Chạy lệnh: *labtainer overrun* trong terminal của Labtainer
* Tại terminal mở ra, hãy xem chương trình mystuff.c. Sử dụng vi hoặc nano, hoặc chỉ nhập *less mystuff.c*.
  + 1. ***Cấu trúc myData***

Nhìn vào struct myData. Trong chương trình khai báo biến my\_data là một struct 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 một 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.

Nếu 19 là ký tự cuối cùng trong mảng, data.public\_info[20] sẽ tham chiếu đến cái gì?

* + 1. ***Địa chỉ của các trường***

Sau khi chương trình khởi tạo cấu trúc my\_data, nó sẽ hiển thị địa chỉ của phần bắt đầu trường public\_data và trường pin. Và nó hiển thị các giá trị bộ nhớ của các trường đó.

* + 1. ***Nội dung bộ nhớ***

Sau đó, chương trình đi vào một vòng lặp cho phép người dùng hiển thị 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?

* 1. ***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. Có bao nhiêu byte phân tách hai trường?

Sử dụng chương trình để hiển thị giá trị của trường pin. Lưu ý rằng trình biên dịch sẽ đệm bộ đệm để biến tiếp theo bắt đầu trên ranh giới từ 4 byte.

* 1. ***Khám phá với gdb***

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. (Sử dụng list showMemory để xem mã nguồn cho hàm đó.) Và sau đó chạy chương trình từ bên trong gdb:

breakpoint <số dòng>

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 bạn đã quan sát trong khi chạy chương trình không?

* 1. ***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. 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

Và từng bước để dịch ngược phần còn lại của hàm handleMyStuff bằng cách nhấn liên tục phím Return 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 bạn 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 bạn đã xem trong gdb, nó sẽ quay trở lại địa chỉ mà bạn đã viết.

1. ***Kết quả cần đạt được***

* Chạy được tất cả các bước như yêu cầu, giải thích/phân tích ngắn gọn các bước và kết quả

1. ***Yêu cầu nộp file kết quả***

* Cần nộp 2 file:
  + 1 file trong thư mục: /home/student/labtainer\_xfer/TÊN\_BÀI\_LAB (tên file dạng *email ptit của sinh viên đã nhập.TÊN\_BÀI\_LAB.lab*)
  + 1 file báo cáo của sinh viên ở dạng pdf, tên file dạng: *Bài thực hành 2\_Họ tên\_Mã sinh viên.pdf (số thứ tự theo đúng thứ tự bài lab)*
    - Trang bìa (ghi rõ môn học, bài thực hành, mã sv và họ và tên)
    - Nội dung mô tả các bước và phân tích sơ bộ + minh chứng kết quả dạng ảnh