**TRƯỜNG ĐẠI HỌC KINH TẾ  
KHOA THỐNG KÊ - TIN HỌC**

**----🙣🕮🙡----**



**TIỂU LUẬN HỌC PHẦN**AN TOÀN VÀ BẢO MẬT HỆ THỐNG THÔNG TIN

**TẤN CÔNG TRÀN BỘ ĐỆM**

***Giảng viên: ThS.Trần Thị Thu Thảo***

**Lớp: 47K14**

**Nhóm 11:** Nguyễn Thị Kim Anh

Mai Thị Hoàng Diểm

Nguyễn Đỗ Ánh Như

Mai Thị Tuyết Trâm

***Đà Nẵng, 30/10/2023***

# 

# ĐÁNH GIÁ THÀNH VIÊN

|  |  |
| --- | --- |
| **Tên thành viên** | **Mức độ hoàn thành** |
| Nguyễn Thị Kim Anh | 100% |
| Mai Thị Hoàng Diểm | 100% |
| Nguyễn Đỗ Ánh Như | 100% |
| Mai Thị Tuyết Trâm | 100% |

**MỤC LỤC**

[ĐÁNH GIÁ THÀNH VIÊN 1](#_Toc149650774)

[I. GIỚI THIỆU TỔNG QUAN 3](#_Toc149650775)

[1. Định nghĩa 3](#_Toc149650776)

[2. Nguyên nhân 4](#_Toc149650777)

[3. Tác hại 5](#_Toc149650778)

[4. Các kiểu lỗi thường gặp 5](#_Toc149650779)

[II. CÁC KIỂU KHAI THÁC LỖI BUFFER OVERFLOW 7](#_Toc149650780)

[1. Khai thác lỗi Buffer Overflow trên stack 7](#_Toc149650781)

[2. Lỗi Buffer Overflow trên Heap 7](#_Toc149650782)

[3. Một số cách khai thác khác: 7](#_Toc149650783)

[4. Các phương pháp ngăn chặn Buffer Overflow 7](#_Toc149650784)

[III. DEMO KHAI THÁC 12](#_Toc149650785)

[1. Stack overflow 12](#_Toc149650786)

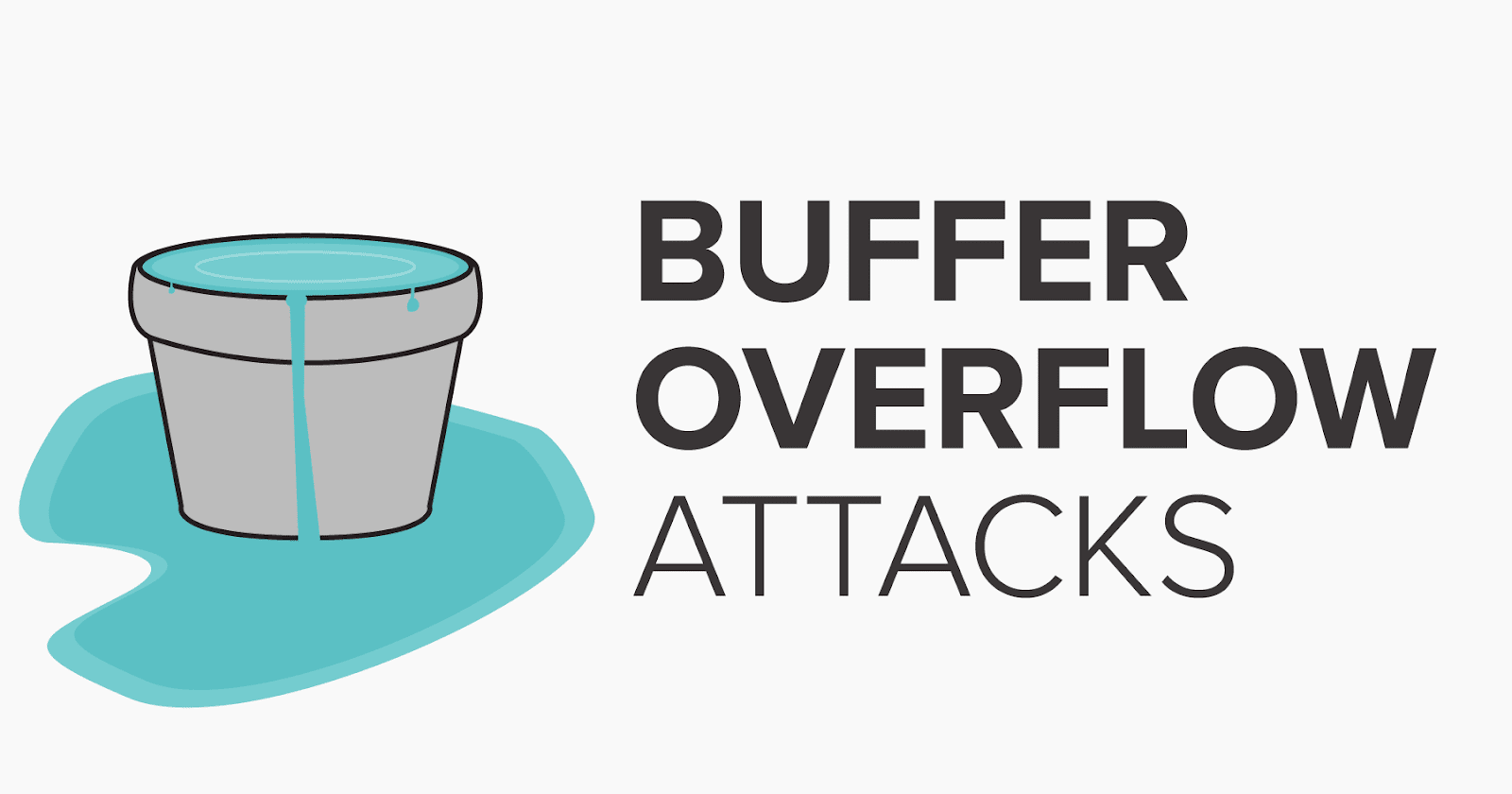
[2. Format string overflow 16](#_Toc149650787)

[3. Heap overflow 19](#_Toc149650788)

[4. Fix lỗi tràn bộ đệm 23](#_Toc149650789)

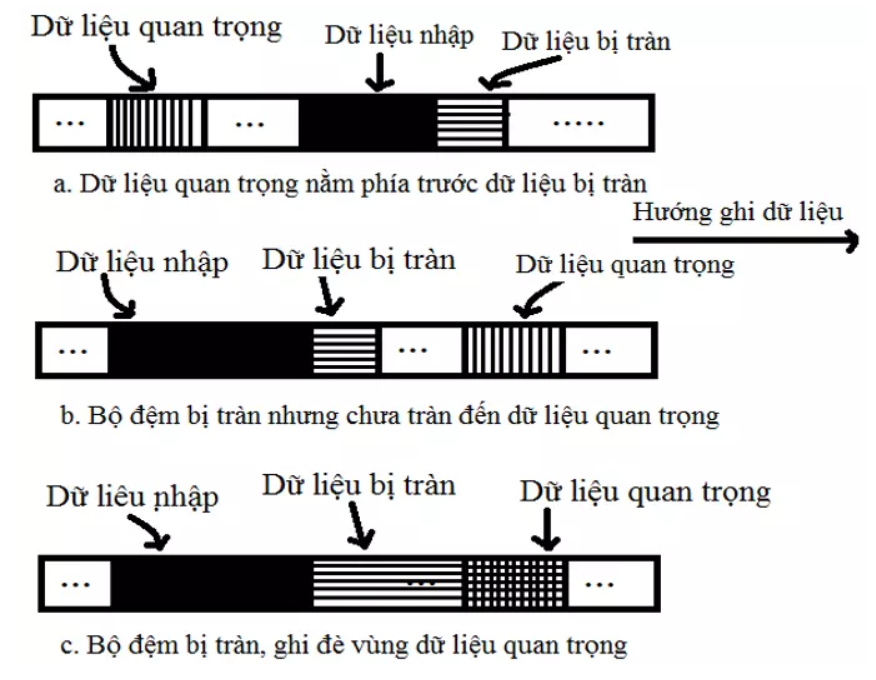
[IV. TÀI LIỆU KHAM KHẢO 30](#_Toc149650790)

# GIỚI THIỆU TỔNG QUAN



## Định nghĩa

* Lỗi Buffer overflow hay tiếng Việt gọi là lỗi tràn bộ nhớ đệm / lỗi tràn bộ đệm là một điều kiện bất thường khi tiến trình lưu trữ dữ liệu vượt ra ngoài biên của bộ nhớ đệm có chiều dài cố định làm dữ liệu bị ghi đè lên các bộ nhớ liền kề.
* Lỗi này liên tục xảy ra do người dùng gửi một lượng lớn tài liệu tới server ứng dụng, điều này làm cho tài liệu bị bắt phải đè lên những vị trí bộ nhớ liền kề đó.
* Đây là một lỗi lập trình hoàn toàn có thể gây ra một ngoại lệ truy nhập bộ nhớ máy tính và chương trình bị kết thúc, hoặc khi người dùng cố ý phá hoại, hoàn toàn có thể tận dụng lỗi này để phá vỡ bảo mật an ninh mạng lưới hệ thống .



## Nguyên nhân

Lỗi Buffer Overflow xảy ra khi một chương trình ghi vào một vùng nhớ đệm (buffer) với một số lượng dữ liệu lớn hơn kích thước của vùng đệm, nguyên nhân do:

* Phương thức kiểm tra bên (boundary) không được thực hiện đầy đủ hoặc là được bỏ qua.
* Nhập quá nhiều dữ liệu mà không kiểm tra kích thước của vùng nhớ đệm, dẫn đến việc vượt quá kích thước của vùng nhớ.
* Lỗi lập trình: Các lập trình viên không kiểm tra và xác thực dữ liệu đầu vào. Khi đó, người dùng nhập vào một dữ liệu lớn hơn kích thước của vùng nhớ đệm, làm cho chương trình ghi vào vùng nhớ khác và ghi đè lên các dữ liệu khác hoặc gây ra lỗi.
* Sử dụng các hàm không an toàn: Sử dụng các hàm không an toàn, chẳng hạn như hàm strcat(), strcpy(), sprintf(), bcopy(), gets(), canf()  trong C, có thể dẫn đến lỗi Buffer Overflow. Các hàm này không kiểm tra độ dài của chuỗi trước khi sao chép vào vùng đệm.
* Các ngôn ngữ lập trình như là ngôn ngữ C, C++ tiềm ẩn các lỗi mà hacker có thể khai thác . Vì ngôn ngữ này thường bỏ qua tính bảo mật để đổi lấy sự hiệu quả, đồng thời không kiểm soát truy cập bộ nhớ.

## Tác hại

Lỗi tràn bộ đệm xảy ra khi một ứng dụng cố gắng ghi dữ liệu vượt khỏi phạm vi bộ đệm (giới hạn cuối hoặc cả giới hạn đầu của bộ đệm). Lỗi tràn bộ đệm có thể khiến ứng dụng ngừng hoạt động, gây mất dữ liệu hoặc thậm chí giúp kẻ tấn công kiểm soát hệ thống hoặc tạo cơ hội cho kẻ tấn công thực hiện nhiều thủ thuật khai thác khác nhau.

## Các kiểu lỗi thường gặp

Hiện nay có nhiều kiểu tấn công Buffer overflow khác nhau. Các kỹ thuật khai thác lỗ hổng buffer overflow thay đổi tùy theo hệ điều hành (OS) và ngôn ngữ lập trình. Tuy nhiên, mục tiêu luôn là thao tác với bộ nhớ máy tính để làm suy yếu hoặc kiểm soát việc thực thi chương trình. Một số loại Buffer Overflow attack phổ biến nhất:

* **Stack Overflow attack** (Tràn bộ đệm dựa trên ngăn xếp): ngăn xếp là cấu trúc dữ liệu trừu tượng làm việc theo nguyên lí LIFO(last in first out) tấn công tràn bộ đệm dựa trên ngăn xếp là một loại tấn công bảo mật trong đó kẻ tấn công gửi một lượng dữ liệu lớn hơn hoặc nằm ngoài giới hạn của một vùng nhớ được cấp phát trong ngăn xếp. Điều này dẫn đến việc ghi đè lên các dữ liệu quan trọng khác trong ngăn xếp, như địa chỉ trả về và các địa chỉ quan trọng khác, từ đó kiểm soát luồng thực thi của chương trình.
* **Heap Overflow attack:** heap dùng để lưu trữ các biến cục bộ của chương trình hoặc lưu trữ các đối tượng của con trỏ khi được cấp phát tấn công tràn bộ đệm dựa trên heap  hiện tượng xảy ra khi 1 một phần bộ nhớ được gán cho heap và dữ liệu ghi vào bộ nhớ đó không được kiểm tra. việc khai thác được thực hiện bằng cách làm hỏng dữ liệu theo những cách cự thể khiến ứng dụng bị đè lên các cấu trúc bên trong như con trỏ
* **Integer Overflow attack:** là một lỗi xảy ra khi một biến kiểu số nguyên (integer) tràn qua giới hạn của kiểu dữ liệu của nó. Điều này xảy ra khi giá trị của biến vượt quá phạm vi tối đa mà kiểu dữ liệu số nguyên có thể biểu diễn. Trong hầu hết các ngôn ngữ lập trình, các giá trị số nguyên thường được cấp phát một số bit nhất định trong bộ nhớ.

*Ví dụ:*  kiểu số nguyên 4 byte (32 bit) trong ngôn ngữ C có giá trị từ (-231) đến (231 – 1). Giả sử bây giờ bạn muốn lưu trữ giá trị 231 vào kiểu số nguyên thì điều gì sẽ xảy ra? Hầu hết các ngôn ngữ hay trình biên dịch đều không gặp lỗi chương trình, mà chỉ đơn thuần thực hiện phép tính mô-đun.Cùng lắm chỉ khiến chương trình bị crash, và không để lại lỗ hổng bảo mật nào.Nhưng đây lại là cơ hội tốn để hacker thực hiện Buffer Overflow attack.Thậm chí, việc tràn số nguyên còn có thể dẫn đến một số hậu quả nghiêmtrọng hơn nữa, chẳng hạn như thao túng tính toán tài chính, ảnh hưởng đến những khách hàng.

* **Format string overflow** (tràn định dạng chuỗi): là một lỗi bảo mật trong lập trình máy tính. Nó xảy ra khi một chương trình cho phép người dùng nhập dữ liệu mà sau đó được định dạng và hiển thị mà không kiểm tra cẩn thận. Khi người dùng nhập dữ liệu chứa các định dạng chuỗi, nó có thể gây ra tràn bộ đệm và làm hỏng dữ liệu hoặc thực hiện các hành động độc hại trong chương trình.
* **Unicode Overflow:**  tạo một Buffer Overflow bằng cách chèn các ký tự Unicode vào input dự kiến của các ký tự ASCII. (ASCII và Unicode đều là các tiêu chuẩn mã hóa để máy tính có thể hiển thị văn bản. ASCII chỉ có kí tự của các nước phương Tây, nhưng Unicode hỗ trợ hầu hết mọi ký tự ở trên thế giới. Do đó, ký tự Unicode thường lớn hơn ASCII nhiều).

# CÁC KIỂU KHAI THÁC LỖI BUFFER OVERFLOW

## Khai thác lỗi Buffer Overflow trên stack

* Ghi đè một biến địa phương nằm gần bộ nhớ đệm trong stack để thay đổi hành vi của chương trình nhằm tạo thuận lợi cho kẻ tấn công.
* Ghi đè địa chỉ trả về trong một khung stack (stack frame). Khi hàm trả về, thực thi sẽ được tiếp tục tại địa chỉ mà kẻ tấn công đã chỉ rõ, thường là tại một bộ đệm chứa dữ liệu vào người dùng.
* Nếu không biết địa chỉ của phần dữ liệu người dùng cung cấp, nhưng biết rằng địa chỉ của nó được lưu trong một thanh ghi, thì có thể ghi đè lên địa chỉ trả về một giá trị địa chỉ của một opcode mà opcode này sẽ có tác dụng làm cho thực thi nhảy đến phần dữ liệu người dùng.
* Cụ thể: nếu địa chỉ đoạn mã độc hai muốn chạy được ghi trong một thanh ghi R, thì một lệnh nhảy đến vị trí chứ opcode cho một lệnh jump R, call R (hay một lệnh tương tự với hiệu ứng nhảy đến địa chỉ ghi trong R) sẽ làm cho đoạn mã trong phần dữ liệu người dùng được thực thi.

## Lỗi Buffer Overflow trên Heap

* Hiện tượng lỗi tràn bộ đệm xảy ra tại khu vực dữ liệu Heap hay cũng chính là hiện tượng tràn Heap, hacker có thể khai thác lỗ hổng này bằng các kỹ thuật (khác với các lỗi tràn stack).
* Bộ nhớ heap là bộ phận thường để chứa dữ liệu của chương trình, nó được cấp phát tự động bởi các ứng dụng thời gian chạy.
* Hacker thực hiện tấn công bằng cách phá những dữ liệu này để làm ứng dụng ghi đè lên các dữ liệu của nội bộ (ví dụ như các con trỏ của danh sách liên kết).

## Một số cách khai thác khác:

* Khai thác dựa vào các lỗ hổng phần mềm thông qua ngôn ngữ lập trình (phần mềm thường được viết bằng ngôn ngữ C).
* Khai thác các trang web có tương tác người dùng nhưng không ràng buộc dữ liệu nhập như các trường hợp username, password,...

## Các phương pháp ngăn chặn Buffer Overflow

* Các nhà phát triển ứng dụng có thể ngăn chặn lỗi Buffer Overflow bằng cách xây dựng các biện pháp bảo mật vào mã phát triển của họ, sử dụng các ngôn ngữ lập trình cấp cao có tích hợp sẵn tính năng bảo vệ, đồng thời, thường xuyên kiểm tra mã để phát hiện và sửa lỗi.
* Một trong những cách để ngăn chặn Buffer Overflow là tránh sử dụng các hàm thư viện tiêu chuẩn chưa được kiểm tra giới hạn, bao gồm get, scanf và strcpy.
* Nhìn chung, 3 phương pháp bảo vệ Buffer Overflow phổ biến là: Address space randomization (ASLR), Ngăn chặn thực thi dữ liệu và Structured exception handler overwrite protection (SEHOP).

#### Phương pháp Address space randomization (ASLR)

* Phương pháp Address Space Randomization (ASLR) là một kỹ thuật trong lĩnh vực bảo mật máy tính nhằm ngăn chặn việc khai thác các lỗ hổng bảo mật dựa trên việc dự đoán địa chỉ bộ nhớ.
* ASLR hoạt động bằng cách thay đổi vị trí của các thành phần quan trọng trong không gian địa chỉ bộ nhớ khi chương trình được thực thi. Thay vì sử dụng những địa chỉ bộ nhớ cố định, ASLR sẽ làm cho các thành phần như các segment code, cấu trúc stack, vùng nhớ heap... nằm ở các vị trí ngẫu nhiên trong không gian địa chỉ bộ nhớ.
* Mục tiêu của ASLR là làm cho việc khai thác lỗ hổng bảo mật khó khăn hơn đối với kẻ tấn công. Nếu kẻ tấn công không biết chính xác vị trí của các thành phần quan trọng, việc thực hiện một cuộc tấn công thành công sẽ khó hơn nhiều. Khi các thành phần di chuyển ngẫu nhiên sau mỗi lần khởi động chương trình, kẻ tấn công sẽ gặp khó khăn trong việc tìm hiểu vị trí đúng của các thành phần và xây dựng các kĩ thuật khai thác phức tạp hơn.
* ASLR là một trong những biện pháp bảo mật quan trọng để ngăn chặn và làm trì hoãn các cuộc tấn công khai thác lỗ hổng bảo mật trên hệ điều hành và các ứng dụng trên máy tính.

#### Ngăn chặn thực thi dữ liệu

* Phương pháp Ngăn chặn thực thi dữ liệu (Data Execution Prevention - DEP) là một công nghệ bảo mật được sử dụng để ngăn chặn việc thực thi mã độc từ các vùng nhớ không an toàn trong quá trình chạy chương trình.
* Nguyên tắc hoạt động của DEP là xác định các vùng nhớ chứa mã thực thi (executable memory) và các vùng nhớ chỉ chứa dữ liệu (non-executable memory). Khi một kỹ thuật tấn công như Buffer Overflow hay chèn mã độc vào bộ nhớ diễn ra, DEP sẽ ngăn chặn việc thực thi các mã độc từ các vùng nhớ chỉ chứa dữ liệu, do đó giảm khả năng bị tấn công thành công.
* DEP có thể hoạt động ở hai chế độ: DEP thiết lập cho toàn bộ hệ thống (DEP toàn diện) hoặc DEP chỉ áp dụng cho các chương trình cụ thể. Trong chế độ DEP toàn diện, nó sẽ áp dụng các biện pháp ngăn chặn thực thi dữ liệu trên toàn hệ thống, trong khi chế độ DEP cho chương trình cụ thể chỉ áp dụng DEP cho một số chương trình đặc biệt mà người dùng chọn.
* DEP là một trong những công nghệ bảo mật quan trọng để ngăn chặn các cuộc tấn công như Buffer Overflow.

#### Structured exception handler overwrite protection (SEHOP)

* Phương pháp Structured Exception Handler Overwrite Protection (SEHOP) là một công nghệ bảo mật được sử dụng để ngăn chặn tấn công sử dụng lỗi ghi đè bộ xử lý ngoại lệ (Exception Handler) trong quá trình thực thi chương trình.
* Một Structured Exception Handler (SEH) là một cơ chế trong các hệ điều hành Windows để xử lý các ngoại lệ (exceptions) xảy ra trong quá trình chạy chương trình. Tuy nhiên, kẻ tấn công có thể tận dụng các lỗi ghi đè bộ xử lý ngoại lệ để thực thi mã độc hoặc kiểm soát luồng thực thi.
* SEHOP tạo ra một khu vực bảo vệ xung quanh bộ xử lý ngoại lệ trong bộ nhớ. Khi một ngoại lệ xảy ra, SEHOP kiểm tra tính toàn vẹn của các cấu trúc dữ liệu của bộ xử lý ngoại lệ để phát hiện bất kỳ sự tấn công nào ghi đè SEH. Nếu phát hiện có sự thay đổi bất thường, nó sẽ kích hoạt một cơ chế bảo vệ và ghi lại thông tin về lỗi để phục vụ cho việc điều tra và ngăn chặn tấn công.
* SEHOP là một biện pháp bảo mật hiệu quả để ngăn chặn các cuộc tấn công sử dụng lỗi ghi đè bộ xử lý ngoại lệ.

#### Các phương pháp khác

##### *a*. Xử lý bộ đệm

- Việc xử lý bộ đệm trước khi đọc hay thực thi có thể làm thất bại các cuộc khai thác lỗi tràn bộ đệm nhưng vẫn không ngăn chặn được một cách tuyệt đối.

* Việc xử lý bao gồm:
* Chuyển từ chữ hoa thành chữ thường.
* Loại bỏ các ký tự đặc biệt và lọc các xâu không chứa kí tự là chữ số hoặc chữ cái.
* Ngoài ra, vẫn còn có các kỹ thuật để tránh việc lọc và xử lý này:
* Alphanumeric code: mã gồm toàn chữ và số.
* Polymorphic code: mã đa hình.
* Self-modifying code: mã tự sửa đổi.
* Tấn công kiểu return – to – libc.

Vì vậy, để tránh các nguy cơ bị khai thác lỗi buffer overflow chúng ta cần sử dụng các biện pháp phòng tránh hiệu quả hơn.

##### b. Lựa chọn ngôn ngữ lập trình

- Ngôn ngữ lập trình có một ảnh hưởng lớn đối với sự xuất hiện lỗi tràn bộ đệm

- Ngôn ngữ lập trình C và C++ là hai ngôn ngữ lập trình thông dụng, nhưng hạn chế của nó là không kiểm tra việc truy cập hoặc ghi đè dữ liệu thông qua các con trỏ. Cụ thể nó không kiểm tra dữ liệu copy vào một mảng có phù hợp với kích thước của mảng hay không:

* Cyclone: một biến thể của C, giúp ngăn chặn các lỗi tràn bộ đệm bằng việc gắn thông tin về kích thước mảng với các mảng.
* Ngôn ngữ lập trình sử dụng nhiều kỹ thuật đa dạng để tránh gần hết việc sử dụng con trỏ và kiểm tra biên do người dùng xác định.
* Nhiều ngôn ngữ lập trình khác cung cấp việc kiểm tra tại thời gian chạy. Việc kiểm tra này cung cấp một ngoại lệ hay một cảnh báo khi C hay C++ ghi đè dữ liệu ví dụ như Python, Ada, Lisp, ...
* Ngoài ra, các môi trường Java hay .Net cũng đòi hỏi kiểm tra biên đối với tất cả các mảng.

##### c. Sử dụng thư viện an toàn

* Sử dụng các thư viện được viết tốt và đã được kiểm thử dành cho các kiểu dữ liệu trừu tượng mà các thư viện này thực hiện tự động việc quản lý bộ nhớ, trong đó có kiểm tra biến có thể làm giảm sự xuất hiện và ảnh hưởng của các hiện tượng tràn bộ đệm.
* Trong các ngôn ngữ này, xâu ký tự và mảng là hai kiểu dữ liệu chính; do đó, các thư viện ngăn chặn lỗi tràn bộ đệm tại các kiểu dữ liệu này có thể cung cấp phần chính của sự che chắn cần thiết. Dù vậy, việc sử dụng các thư viện an toàn một cách không đúng có thể dẫn đến tràn bộ đệm và một số lỗ hổng khác. Các thư viện an toàn gồm có: The Better String Library, Arri Buffer API, Vstr.

##### d. Chống tràn bộ đệm trên Stack

* Stack-smashing protection là kỹ thuật dùng để phát hiện các hiện tượng tràn bộ đệm phổ biến nhất. Kỹ thuật này kiểm tra xem stack đã bị sửa đổi hay chưa khi một hàm trả về. Nếu stack đã bị sửa đổi, chương trình kết thúc bằng một lỗi segmentation fault.
* Chế độ Data Execution Prevention (cấm thực thi dữ liệu) của Microsoft bảo vệ các con trỏ và không cho chúng bị ghi đè.
* Có thể bảo vệ stack bằng cách phân tán stack thành 2 phần, một phần dành cho dữ liệu và một phần dành cho các bước trả về của hàm. Sự phân chia này được dùng trong ngôn ngữ Forth. Tuy nó không phải một cách thiết kế theo tiêu chí an toàn. Nhưng đây cũng không phải một giải pháp hoàn chỉnh đối với tràn bộ đệm, khi các dữ liệu không phải địa chỉ trả về vẫn có thể bị ghi đè.

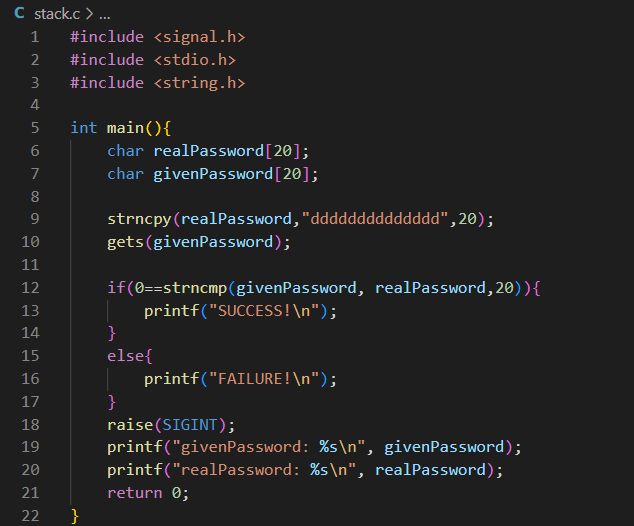
##### e. Kiểm tra sâu đối với gói tin

* Biện pháp kiểm tra sâu đối với gói tin (deep packet inspection-DPI) có thể phát hiện việc cố gắng khai thác lỗi tràn bộ đệm từ xa ngay từ biên giới mạng. Các kĩ thuật này có khả năng ngăn chặn các gói tin có chứa chữ ký của một vụ tấn công đã biết hoặc chứa các chuỗi dài các lệnh No-Operation (NOP- lệnh rỗng không làm gì).
* Việc rà gói tin không phải là một phương pháp hiệu quả vì nó chỉ có thể ngăn chặn các cuộc tấn công đã biết và có nhiều cách để mã hóa một lệnh NOP. Các kẻ tấn công có thể đã sử dụng mã[alphanumeric,](https://vi.wikipedia.org/w/index.php?title=Alphanumeric_code&action=edit&redlink=1)metamorphic,shellcode tự sửa và phát hiện để tránh rà gói tin

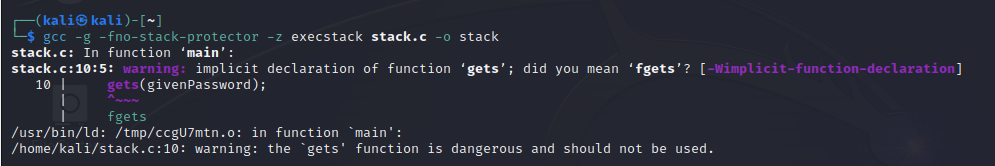
# DEMO KHAI THÁC

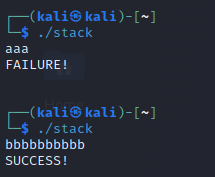
## Stack overflow

**Bước 1:**  Chạy chương trình C có lỗi buffer overflow, chương trình chứa lỗi tại hàm gets để đọc đầu vào từ người dùng mà không kiểm tra giới hạn của biến givenPassword. Nếu đầu vào của người dùng quá dài có thể dẫn đến tràn bộ nhớ.

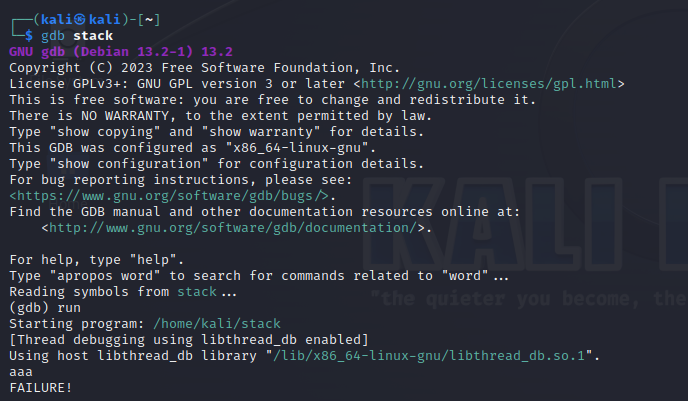
****

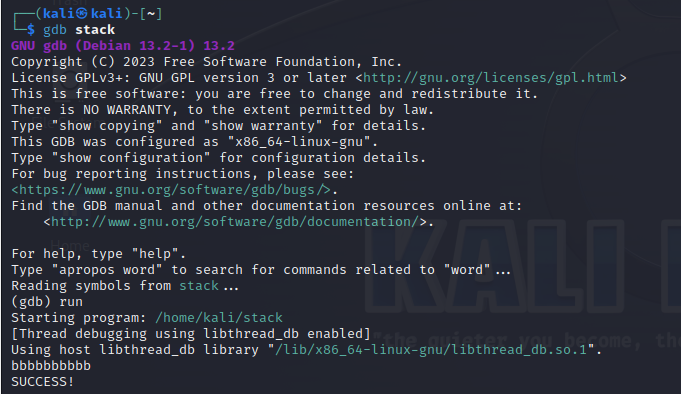
**Bước 2:** Tiến hành biên dịch mã nguồn C từ file stack.c thành file có tên là stack. ‘-fno-stack-protector -z execstack’ dùng để tắt bảo vệ stack. Sau khi biên dịch thành công, bạn có thể chạy chương trình bằng cách gọi ./stack trong dòng lệnh.

****

****

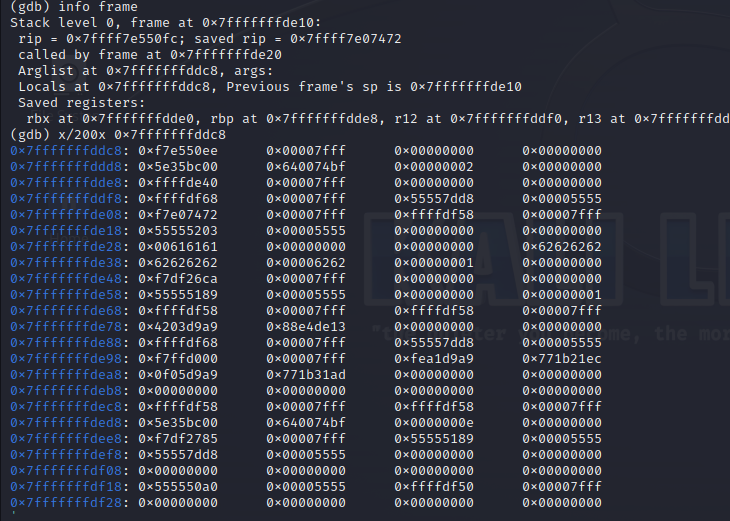
**Bước 3:** Gdb stack là trình gỡ lỗi, xem chương trình đang làm gì, bộ nhớ như thế nào khi đang chạy.

****

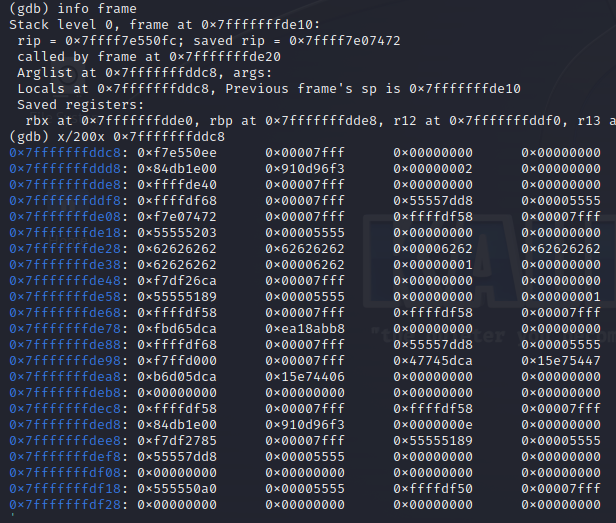
****

**Bước 4:** Gdb khung thông tin ‘info frame’ cho phép tìm vị trí trong bộ nhớ và các biến cục bộ sẽ nằm trên ngăn xếp. Sau khi biết được vị trí biến cục bộ thì in ra vùng bộ nhớ của nó

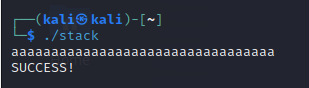
* Khung thông tin nhập sai mật khẩu

****

* Khung thông tin nhập đúng mật khẩu

****

**Bước 5:**  Ta có thể thấy xuất hiện lỗi tràn bộ đệm khi các địa chỉ bị tràn và bị trùng với nhau. Ghi đè giữa 2 biến cục bộ là realPassword và givenPassword nằm gần bộ nhớ đệm trong stack để thay đổi hành vi của chương trình nhằm tạo thuận lợi cho kẻ tấn công.

****

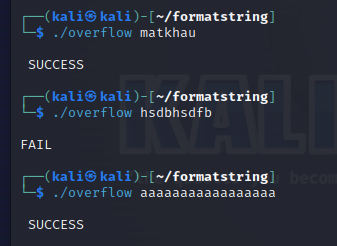
## Format string overflow

**Bước 1**: Chạy một chương trình C đơn giản có lỗ hổng tràn bộ nhớ đệm.

Trong chương trình đã có một lỗi sai tại hàm strcpy, vì hàm này không kiểm tra đầu vào có kích thước lớn hơn mảng đã cấp phát hay không tạo lỗ hổng để hacker khai thác làm tràn bộ đệm.

**Mục tiêu:** Làm tràn bộ đệm của chương trình nhằm khiến chương trình chạy sai (cụ thể: nhập mật khẩu sai nhưng khiến màn hình vẫn xuất hiện thông báo “SUCCESS”)

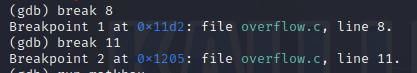


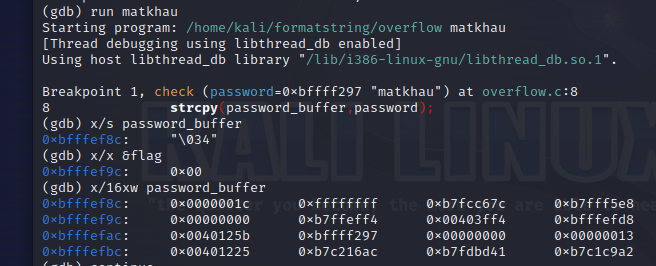


**Bước 2**: Kiểm tra, nếu mật khẩu đúng thì thông báo ‘SUCCESS’, nếu sai thì thông báo ‘FAIL’

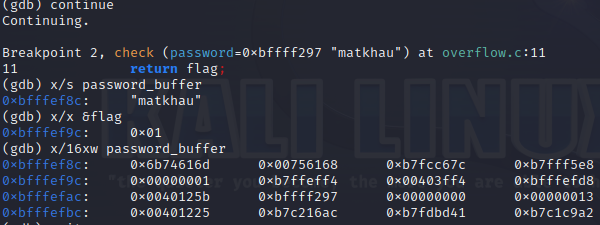
**Bước 3**: Dùng gdb để debug

**Bước 4**: Đặt break tại 2 dòng 8 và 11 để kiểm tra. Tại breakpoint đầu tiên, chạy với chuỗi như hình, kiểm tra con trỏ ta thấy rằng nó đang chứa giá trị ngẫu nhiên ở địa chỉ **0xbfffef8c** và biến **flag** đang ở địa chỉ **0xbfffef9c** và có giá trị 0x0



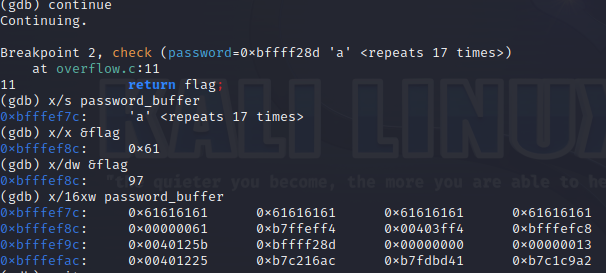
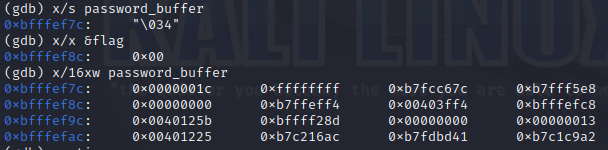
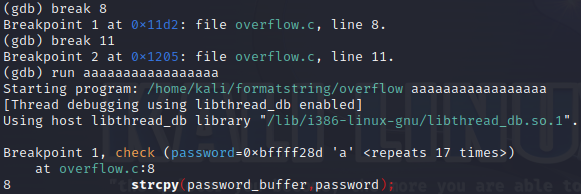


**Bước 5**: Tiếp tục với ***Breakpoint*** tiếp theo, biến flag đang có giá trị 0x01 vì ta chạy với mật khẩu đúng với mật khẩu trong chương trình C



**Bước 6**: Làm tương tự các bước 4 và 5 để chạy với chuỗi như hình. Ở đây xuất hiện một thông tin quan trọng: là “password\_buffer” đã tràn tới đến phần bộ nhớ của “flag” và sửa đổi “flag” sang giá trị 0x61. Do vậy chương trình sẽ coi giá trị của biến này là một số nguyên và giá trị của nó là 97.

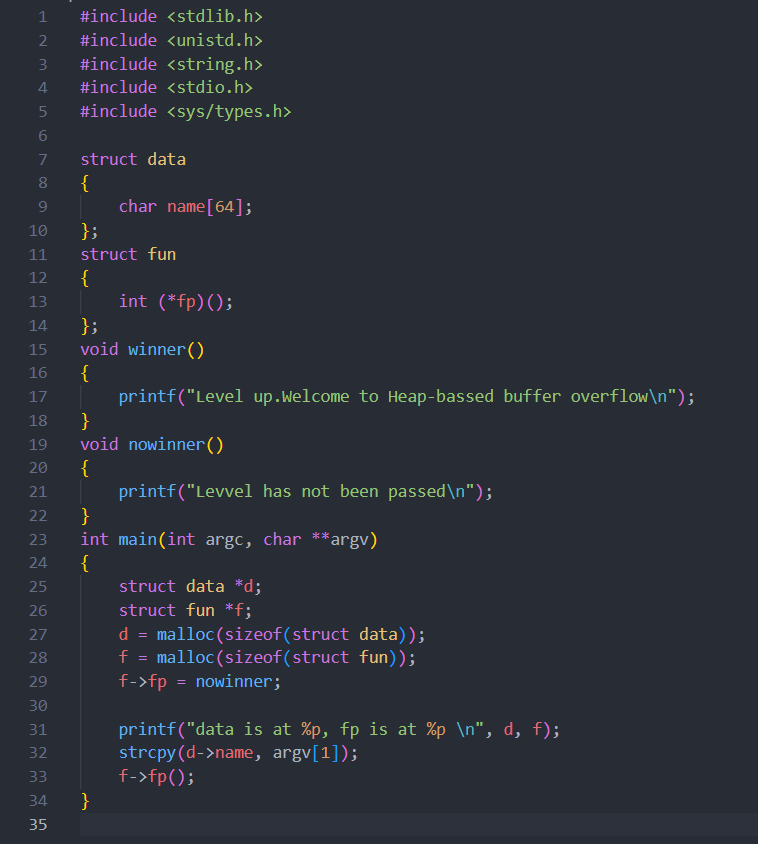
Sau quá trình overflow, hàm “check” sẽ trả về giá trị 97 thay vì 0. Lúc này màn hình sẽ thông báo “SUCCESS”



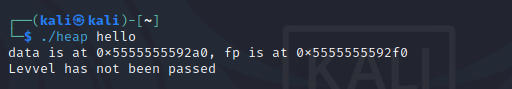
## Heap overflow

**Bước 1**: Chạy chương trình C có lỗi buffer overflow, chương trình chưa lỗi tại hàm strcpy vì không kiểm tra kích thước đầu vào

**Mục tiêu**: Thay đổi địa chỉ trỏ đến của con trỏ f->fp từ hàm nowinner thành địa chỉ của hàm winner



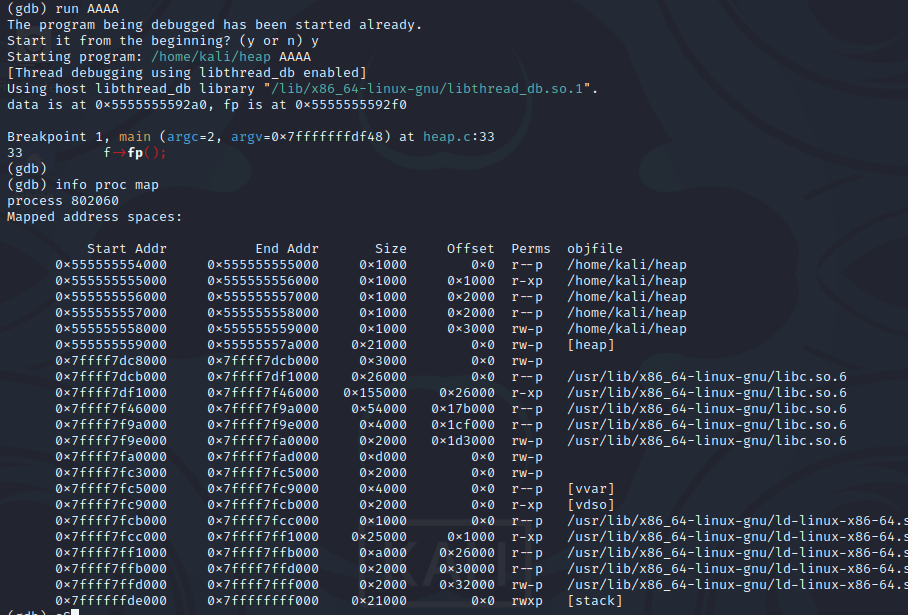
**Bước 2**: Chạy chương trình, chương trình chạy đúng sẽ luôn trả về thông báo trong hàm nowinner



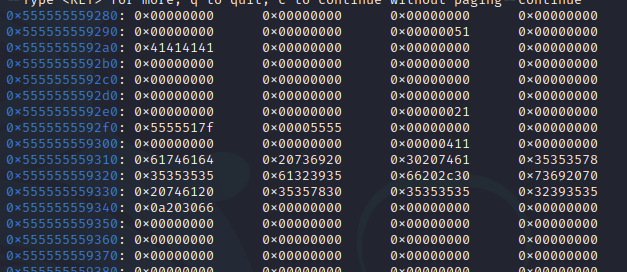
**Bước 3**: Dùng list để xem chương trình trong gdb, thực hiện đặt break point tại dòng 33, dòng chưa lệnh gán con trỏ f cho hàm fp lúc này con trỏ sẻ chứa địa chỉ của con trỏ fp



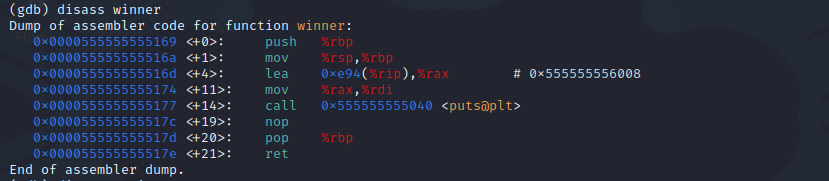
**Bước 4**: Chạy chương trình bằng lệnh run, dùng info proc map để xem địa chỉ của heap



**Bước 5**: Dùng x/500x strart\_addr\_heap để xem các biến d và f được lưu trong heap, có thể thấy 0x5555555592a0 là địa chỉ của hàm winner và 0x5555555592f0 địa chỉ hàm nowinner



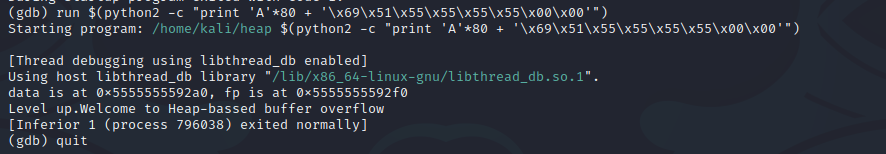
**Bước 6**: Xem hợp ngữ của winner



**Bước 7:** Chúng ta có thể thấy địa khoảng cách giữa 2 con trỏ f và fp được lưu trong heap là 80 chúng ta sẻ thực hiện nhập 1 đoạn dữ liệu có độ lớn là 80 và địa chỉ của hàm winner ghi đè vào vị trí của hàm nowinner để chương trình chạy ra thông báo trong winner, chạy lệnh python

**$(python2 -c "print 'A'\*80 + '\x69\x51\x55\x55\x55\x55\x00\x00'")**

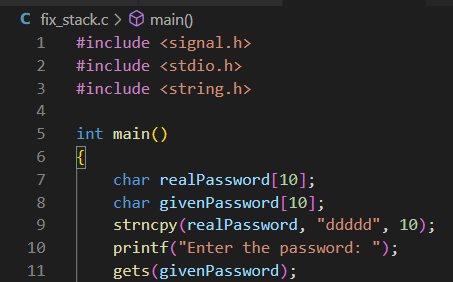
**với '\x69\x51\x55\x55\x55\x55\x00\x00'** là địa chỉ của lệnh winner được viết dưới dạng  dãy bytes cho máy tính hiểu

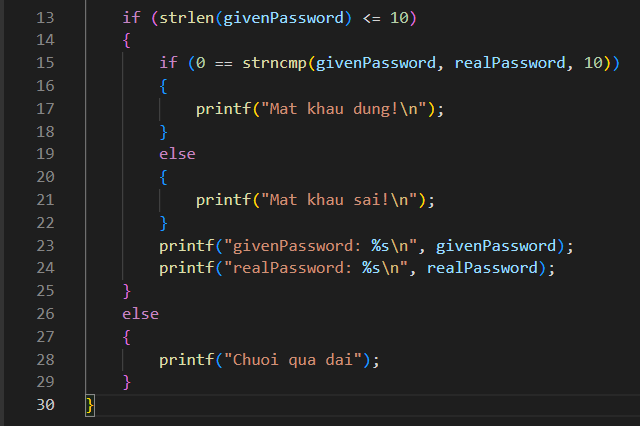


## Fix lỗi tràn bộ đệm

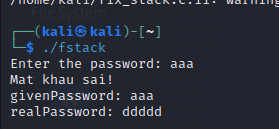
***4.1. Trên stack***

**Bước 1**: Ta thực hiện kiểm tra độ dài chuỗi nhập vào bằng hàm strlen(), nếu độ dài chuỗi nhập vào nhỏ hơn hoặc bằng 10 thì được nhập vào.

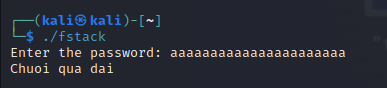




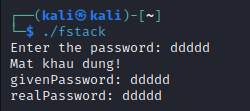
**TH1**: Độ dài chuỗi nhập vào nhỏ hơn hoặc bằng 10 nhưng không đúng thì kết quả in ra trên màn hình là “Mat khau sai!”



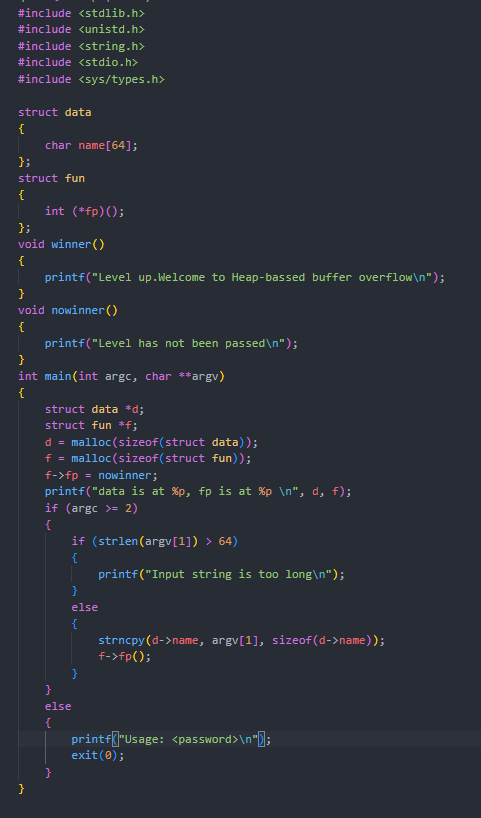
**TH2:** Độ dài chuỗi lớn hơn 10 thì kết quả in ra trên màn hình “Chuoi qua dai”



**TH3:** Mật khẩu nhập đúng thì kết quả in ra trên màn hình “Mat khau dung!”

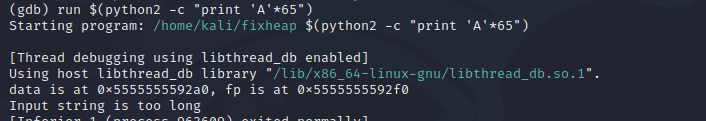


***4.2. Trên heap***

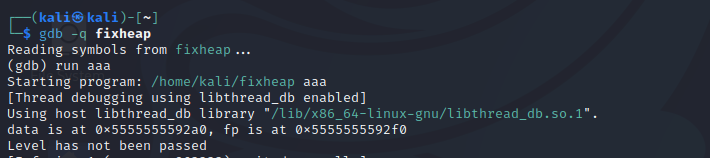


Ta thực hiện kiểm tra độ dài chuỗi nhập vào bằng strlen(), nếu độ dài chuỗi nhỏ hơn hoặc bằng 64 thì được nhập vào, trong câu lệnh strncpy(d->name, argv[1], sizeof(d->name)); trong câu lệnh này thêm hàm sizeof(d->name) để lấy chuỗi nhập vào có độ dài bằng chìu dài tối đa của biến name là 64.

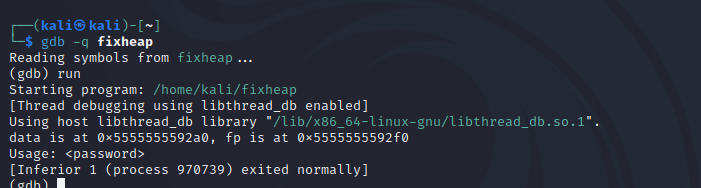
**TH1:** Không nhập quá số lượng thì sẽ in ra dòng input “string too long”



**TH2:** Trường hợp nhập đúng



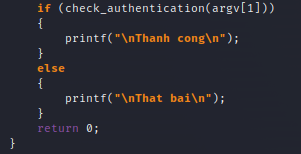
**TH3:** Trường hợp không nhập gì thì sẽ hiện dòng chữ Usage: <password>



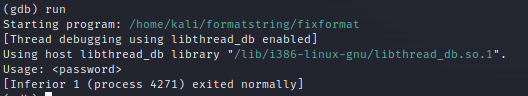
***4.3. Fix lỗi format string***

Ta thực hiện kiểm tra đầu vào. Nếu độ dài đầu vào bé hơn hoặc bằng mảng password\_buffer thì dùng hàm strncpy sao chép mật khẩu vào ‘password\_buffer’ với giới hạn độ dài là sizeof(password\_buffer)

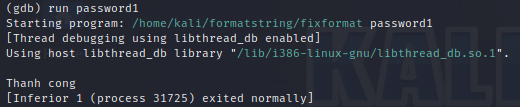




**TH1:** Nếu không nhập gì thì sẽ hiện dòng chữ Usage: <password>

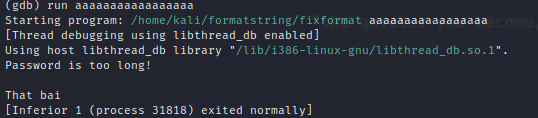


**TH2:** nếu nhập đúng mật khẩu thì hiện ‘Thanh cong’



**TH3:** nếu nhập dài hơn 16 kí tự hay nhập sai thì hiện ‘Password is too long!’

‘That bai’



# TÀI LIỆU KHAM KHẢO

[*https://viblo.asia/p/ky-thuat-tan-cong-buffer-overflow-3Q75wmjMZWb  
https://www.youtube.com/watch?v=GY6sOCHSlJk&t=1511s*](https://viblo.asia/p/ky-thuat-tan-cong-buffer-overflow-3Q75wmjMZWb)

[*https://vietnix.vn/buffer-overflow-la-gi/#stack-overflow-attackhttps://viblo.asia/p/ky-thuat-tan-cong-buffer-overflow-3Q75wmjMZWb*](https://vietnix.vn/buffer-overflow-la-gi/#stack-overflow-attackhttps://viblo.asia/p/ky-thuat-tan-cong-buffer-overflow-3Q75wmjMZWb)[*https://www.rapid7.com/blog/post/2019/02/19/stack-based-buffer-overflow-attacks-what-you-need-to-know/*](https://www.rapid7.com/blog/post/2019/02/19/stack-based-buffer-overflow-attacks-what-you-need-to-know/)[*https://vi.wikipedia.org/wiki/L%E1%BB%97i\_tr%C3%A0n\_b%E1%BB%99\_nh%E1%BB%9B\_%C4%91%E1%BB%87m*](https://vi.wikipedia.org/wiki/L%E1%BB%97i_tr%C3%A0n_b%E1%BB%99_nh%E1%BB%9B_%C4%91%E1%BB%87m)