**I. Stack:**

# Giới thiệu về bộ nhớ Stack

- Bộ nhớ máy tính gồm nhiều phần như RAM, ROM, cache,... Trong đó, **stack** là một vùng nhớ trong **RAM** được sử dụng để lưu trữ các giá trị tạm thời trong quá trình thực thi chương trình, đặc biệt là khi thực hiện các lệnh gọi hàm. **Stack** đóng vai trò quan trọng trong việc quản lý bộ nhớ ngắn hạn cho các hàm và biến cục bộ.

**- Stack** được quản lý bằng một thanh ghi đặc biệt gọi là **ESP (Extended Stack Pointer)** hay **con trỏ Stack**, có nhiệm vụ luôn trỏ đến vị trí hiện tại của đỉnh Stack. Thanh ghi ESP được sử dụng để thêm hoặc lấy dữ liệu khỏi stack thông qua các thao tác như PUSH và POP.

- Khi một chương trình thực hiện nhiều hàm lồng nhau, cần có một cơ chế để ghi nhớ địa chỉ trả về, các tham số và biến cục bộ của mỗi hàm. Stack giúp chương trình quản lý những thông tin này một cách hiệu quả, lưu trữ chúng trong các khung stack (stack frame) tương ứng với từng hàm.

# Hoạt động của Stack

**- Stack** hoạt động dựa trên nguyên tắc **LIFO** (Last In, First Out - Vào sau ra trước), nghĩa là dữ liệu mới nhất được thêm vào sẽ là dữ liệu đầu tiên được lấy ra. Các thao tác cơ bản bao gồm:

**+ PUSH**: Đẩy (thêm) dữ liệu vào Stack. Mỗi lần thực hiện thao tác này, thanh ghi ESP sẽ giảm xuống (vì địa chỉ bộ nhớ của stack giảm dần).

**+ POP**: Lấy (rút) dữ liệu từ Stack ra. Thanh ghi ESP tăng lên khi lấy dữ liệu.

**-** Khi một hàm được gọi, hệ điều hành tạo một khung Stack (**stack frame**) cho hàm đó, chứa các thông tin sau:

**+ Địa chỉ trả về**: Địa chỉ bộ nhớ nơi chương trình cần quay lại sau khi thực thi xong hàm.

**+ Các tham số của hàm**: Nếu hàm có tham số, các tham số này sẽ được đưa vào Stack để sử dụng trong hàm.

**+ Biến cục bộ**: Các biến được khai báo bên trong hàm sẽ được cấp phát bộ nhớ trên Stack.

- Sau khi hàm kết thúc, khung stack này sẽ bị xóa khỏi Stack, và con trỏ ESP được cập nhật để quay lại vị trí trước khi gọi hàm.

**- Tràn Stack (Stack Overflow):** Khi số lượng lệnh gọi hàm hoặc biến cục bộ quá nhiều, kích thước của Stack có thể vượt quá giới hạn bộ nhớ cho phép. Điều này dẫn đến **tràn Stack** (stack overflow), gây ra lỗi nghiêm trọng cho chương trình. Thông thường, lỗi tràn stack xảy ra khi:

+ Có quá nhiều lệnh gọi đệ quy mà không có điều kiện dừng.

+ Có quá nhiều biến cục bộ hoặc các biến lớn được cấp phát trong hàm.

# Quản lý Buffer trên Stack

**- Buffer** là một vùng nhớ tạm thời được sử dụng để lưu trữ dữ liệu trong quá trình xử lý. Trong các chương trình C/C++, buffer thường được cấp phát trên Stack. Một điểm yếu của buffer trong C/C++ là kích thước của nó cố định và không có cơ chế kiểm tra kích thước dữ liệu đầu vào.

**+ Buffer trong Stack**: Khi một buffer được khai báo trong một hàm, nó sẽ được cấp phát bộ nhớ trên Stack. Ví dụ: char buffer[10]; sẽ cấp phát một vùng nhớ có kích thước 10 byte trên Stack.

**- Nguy cơ tràn buffer:** Nếu dữ liệu đầu vào lớn hơn kích thước buffer, nó sẽ ghi đè lên các vùng nhớ liền kề trên Stack. Điều này có thể bao gồm địa chỉ trả về hoặc các biến cục bộ khác, dẫn đến lỗi chương trình, thậm chí có thể bị khai thác bởi kẻ tấn công.

**II. Buffer Overflow:**

**1. Khái niệm :**

Lỗi tràn bộ đệm (Buffer Overflow) 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. Kết quả là dữ liệu có thể đè lên các bộ nhớ liền kề. Dữ liệu bị ghi đè có thể bao gồm các bộ nhớ đệm khác, các biến và dữ liệu điều khiển luồng chảy của cả chương trình (program flow control).

**2. Nguyên Nhân**

* Phương thức kiểm tra bên (boundary) không được thực hiện đầy đủ hoặc là được bỏ qua.
* Các ngôn ngữ lập trình như là ngôn ngữ C, bản thân nó đã tiền ẩn các lỗi mà hacker có thể khai thác.
* Các phương thức strcat(), strcpy(), sprintf(), bcopy(), gets(), canf() trong ngôn ngữ C có thể được khai thác vì các hàm này không kiểm tra những buffer được cấp phát trên stack có kích thước lớn hơn dữ liệu được copy vào buffer hay không.

**3. 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.

**4. Các kiểu lỗi Buffer Overflow thường gặp.**

* Stack overflow: sẽ xuất hiện khi buffer tràn trong stack space và là hình thức tấn công phổ biến nhất của lỗi tràn bộ đệm. Mục đích: - Ghi đè một biến địa phương nằm gần bộ nhớ đệm trong stack để thay đổi hành vi của chương trình nhằm phục vụ ý đồ của hacker. - Ghi đè địa chỉ trả về trong khung stack. Khi hàm trả về thực thi sẽ được tiếp tục tại địa chỉ mà hacker đã chỉ rõ, thường là tại một bộ đệm chứa dữ liệu vào của người dùng.
* Format String: Tràn bộ đệm chuỗi định dạng (thường được gọi là “lỗ hổng định dạng chuỗi”) là lỗi tràn bộ đệm ở mức chuyên môn cao, tác hại tương tự như các cuộc tấn công tràn bộ đệm khác. Về cơ bản, lỗ hổng định dạng chuỗi tận dụng lợi thế của các kiểu dữ liệu hỗn hợp và kiểm soát thông tin trong chức năng nhất định, chẳng hạn như C/C++ printf
* Chương trình đơn giản này có đầu vào từ người dùng và hiển thị lại trên màn hình. Chuỗi %s có nghĩa là các tham số khác. str sẽ được hiển thị như là một chuỗi. Ví dụ trên không dễ bị tấn công, nhưng nếu thao đổi dòng cuối cùng thành printf(str); thì nó có thể dễ dàng bị khai thác.

**5. Các kiểu khai thác lỗi Buffer Overflow**

**Khai thác lỗi tràn bộ đệm 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.

**Khai thác lỗi tràn bộ đệm trên heap**

Một hiện tường tràn bộ đệm xảy ra trong khu vực dữ liệu heap được gọi là hiện tượng tràn heap và có thể khai thác được bằng các kỹ thuật khác với các lỗi tràn stack.

* Bộ nhớ heap được cấp phát động bởi các ứng dụng tại thời gian chạy và thường chứa dữ liệu của chương trình.
* Việc khai thác được thực hiện bằng cách phá dữ liệu này theo các cách đặc biệt để làm cho ứng dụng ghi đè lên các cấu trúc dữ liệu nội bộ chẳng hạn 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,...

**6. CÁCH PHÁT HIỆN LỖI Buffer Overflow.**

Công nghệ biên dịch tối ưu để phát hiện lỗi tràn bộ đệm trong chương trình C hoặc C++ dựa trên việc phân tích kích thước dữ liệu trong mã nguồn. Trình biên dịch phải hiểu thông tin từ khai báo biến và các lệnh gọi hàm để tạo ra mã đúng. Phương pháp \*\*phân tích liên thủ tục (interprocedural analysis)\*\* giúp trình biên dịch xác định các lỗ hổng như cố gắng sao chép dữ liệu quá lớn vào bộ đệm nhỏ. Bằng cách xây dựng đồ thị kết nối giữa các hàm, trình biên dịch có thể phát hiện lỗi bảo mật tiềm ẩn, chẳng hạn khi dữ liệu ngoại lai được đưa vào bộ đệm không đủ kích thước. Công nghệ này giúp phát hiện lỗi tràn bộ đệm một cách chính xác và hiệu quả hơn các phương pháp khác, cung cấp cái nhìn sâu sắc về bảo mật mã nguồn.

**7. Biện pháp ngăn chặn**

**- Kiểm tra kích thước buffer:** Sử dụng các hàm an toàn hơn như

+ strncpy() thay cho strcpy() – Giới hạn kích thước chuỗi được sao chép.

+ fgets() thay cho gets() – Giới hạn kích thước chuỗi nhập từ bàn phím.

**- Sử dụng Heap thay vì Stack:** Với những dữ liệu lớn hoặc biến có kích thước động, lập trình viên nên sử dụng **Heap** thay vì **Stack**. Heap có kích thước linh hoạt hơn và ít gặp lỗi tràn bộ đệm.

**-** Kỹ thuật ASLR (Address Space Layout Randomization) làm cho địa chỉ của các vùng bộ nhớ, bao gồm stack, heap và các thư viện hệ thống, bị ngẫu nhiên hóa mỗi khi chương trình chạy. Điều này khiến cho việc dự đoán địa chỉ trở nên khó khăn hơn với kẻ tấn công.

- Kỹ thuật DEP (Data Execution Prevention) ngăn chặn việc thực thi mã trong các vùng bộ nhớ không đáng tin cậy như stack và heap. Nếu kẻ tấn công chèn mã độc vào buffer, chương trình sẽ không thể thực thi mã đó.

# 6. Demo đồ đó keo

- Bài tập CTF: [link](https://inseclab.uit.edu.vn/k14-chia-se-ve-cuoc-thi-ctf-dau-tien-ky-5-nhung-bai-buffer-overflow-kinh-dien/)  [link](https://nhattruong.blog/2019/03/14/note-lo-hong-buffer-overflow/)

- Nguồn có thể tham khảo để viết report chi tiết: [Bài tập lớn - Tìm hiểu về Buffer overflow Attack - Mục lục Giới - Studocu](https://www.studocu.com/vn/document/hoc-vien-cong-nghe-buu-chinh-vien-thong/an-toan-va-bao-mat-he-thong-thong-tin/bai-tap-lon-tim-hieu-ve-buffer-overflow-attack/88921067)

Demo code:

|  |
| --- |
| #include <iostream>  #include <cstring>  void vulnerableFunction() {  char buffer[10]; // Bộ đệm nhỏ với kích thước 10 byte  char overflowData[20] = "This\_is\_a\_long\_data";  // Cố tình sao chép chuỗi dữ liệu lớn hơn vào buffer nhỏ  std::cout << "Buffer address: " << static\_cast<void\*>(buffer) << std::endl;  std::cout << "Overflowing buffer with: " << overflowData << std::endl;    strcpy(buffer, overflowData); // Không kiểm tra kích thước, gây tràn bộ đệm  std::cout << "Buffer after overflow: " << buffer << std::endl;  }  int main() {  vulnerableFunction(); // Gọi hàm dễ bị tấn công  return 0;  } |

**Giải thích:**

1. **buffer[10]**: Là một mảng có kích thước cố định 10 byte.
2. **overflowData[20]**: Là dữ liệu có kích thước lớn hơn buffer, với 19 ký tự cộng thêm ký tự kết thúc chuỗi \0.
3. **strcpy(buffer, overflowData)**: Lệnh này sao chép chuỗi lớn vào buffer, dẫn đến **tràn bộ đệm** vì chuỗi vượt quá kích thước 10 byte.
4. **Kết quả**: Sẽ xảy ra **ghi đè lên bộ nhớ** sau buffer, dẫn đến các vấn đề không mong muốn, có thể là lỗi chương trình hoặc thay đổi dữ liệu không kiểm soát.