**TRÀN BỘ NHỚ ĐỆM / BUFFER OVERFLOW**

1. **Tràn bộ nhớ đệm là gì?**

**1.1 Khái niệm**

Trong lĩnh vực an ninh máy tính và lập trình, lỗi Buffer overflow hay tiếng Việt gọi là lỗi tràn bộ nhớ đệm/lỗi tràn bộ đệm là khi mà bộ nhớ bị ghi đè nhiều lần trên ngăn xếp. Lỗi này thường xuyên xảy ra do người dùng gửi một lượng lớn [dữ liệu](https://bkhost.vn/blog/data-du-lieu/) tới server ứng dụng, điều này làm cho dữ liệu bị bắt phải đè lên các vị trí bộ nhớ liền kề đó. Đây là một lỗi lập trình có thể gây ra một ngoại lệ truy nhập bộ nhớ máy tính và chương trình bị kết thúc, hoặc khi người dùng cố tình phá hoại, có thể lợi dụng lỗi này để phá vỡ an ninh hệ thống.

**1.2 Nguyên nhân**

* Không thực hiện đầy đủ, hoặc không kiểm tra biên.
* Các ngôn ngữ lập trình như C, bản thân nó đã luôn có những tiềm ẩn các lỗ hổng mà hacker dễ dàng có thể tấn công vào. [Trong ngôn ngữ lập trình](https://bkhost.vn/blog/cac-ngon-ngu-lap-trinh-pho-bien-nhat-hien-nay/) C còn tồn tại các hàm không kiểm tra những buffer được cấp phát trên stack có kích thước lớn hơn dữ liệu được copy và bộ đệm hay không.

**1.3 Các cách khai thác tràn bộ nhớ đệm**

* **Data Execution Prevention**

Phương pháp đơn giản nhất để chặn khả năng khai thác lỗ hổng phát sinh do tràn bộ nhớ đệm mà các lập trình viên thường sử dụng là luôn đảm bảo mã lập trình được bảo mật. Thực ra đây không phải là một tiến trình được tự động hóa vì nó yêu cầu tiêu tốn nhiều thời gian và công sức cho việc kiểm tra lại mã để đảm bảo rằng tính toàn vẹn của mã chương trình được duy trì, do đó số lượng dòng lệnh tỉ lệ thuận với thời gian và công sức cần phải bỏ ra. Xuất phát từ yêu cầu đó Microsoft đã phát triển một tính năng có tên Data Execution Prevention (DEP). Có hai cách phổ biến để fix lỗi này đó là:

* **Hardware-based DEP**

DEP được cho là bảo mật nhất khi sử dụng Hardware-based DEP. Trong trường hợp này vi xử lý sẽ đánh dấu mọi vị trí nhớ là “không thể thực thi” nếu vị trí này không chứa mã thực thi. Mục đích của việc này là DEP sẽ chặn mọi mã chạy trong những vùng không thể thực thi.

* **Software-based DEP**

Khi Hardware-based DEP không tồn tại thì Software-based DEP phải được sử dụng. Loại DEP này được tích hợp trong hệ điều hành Windows. Software-based DEP vận hành bằng cách dò tìm thời điểm mà những ngoại lệ được các chương trình đưa vào và đảm bảo rằng những ngoại lệ này là một phần hợp lệ của chương trình này trước khi cho phép chúng xử lý.

1. **Tràn bộ đệm ngăn xếp - Stack Overflows**

Tràn bộ đệm ngăn xếp (Stack Overflow) là một loại lỗi phổ biến trong lập trình, đặc biệt là trong ngôn ngữ lập trình sử dụng ngăn xếp (stack) như C, C++, Java, và các ngôn ngữ khác.

Ngăn xếp là một cấu trúc dữ liệu theo nguyên tắc "Last In, First Out" (LIFO), có nghĩa là phần tử được thêm vào cuối (push) sẽ được lấy ra trước (pop). Khi một hàm được gọi, thông tin về hàm đó sẽ được đặt trên ngăn xếp. Khi hàm hoàn thành, thông tin của hàm đó sẽ bị loại bỏ khỏi ngăn xếp.

Tràn bộ đệm ngăn xếp xảy ra khi quá nhiều dữ liệu được đặt trên ngăn xếp mà không được loại bỏ. Điều này thường xảy ra khi một hàm gọi chính nó một cách đệ quy mà không có điều kiện dừng hoặc khi một chương trình sử dụng một lượng lớn bộ nhớ stack do việc khai báo các biến cục bộ quá lớn.

Khi xảy ra tràn bộ đệm ngăn xếp, chương trình thường sẽ bị crash hoặc gặp phải các vấn đề khác như không thể thực hiện các lời gọi hàm tiếp theo một cách đúng đắn. Để khắc phục, cần phải xem xét lại cách sử dụng đệ quy hoặc tối ưu hóa việc sử dụng bộ nhớ stack trong chương trình.

1. **Phòng thủ chống tràn bộ đệm**

Để phòng thủ chống tràn bộ đệm (Stack Overflow), ta có thể thực hiện một số biện pháp sau:

* Tối ưu hóa đệ quy: Nếu sử dụng đệ quy, hãy chắc chắn rằng bạn có một điều kiện dừng hoặc giới hạn số lần đệ quy để tránh tràn bộ đệm.
* Sử dụng vòng lặp thay vì đệ quy: Trong một số trường hợp, bạn có thể thay thế các hàm đệ quy bằng vòng lặp để tránh tràn bộ đệm.
* Kiểm soát cỡ ngăn xếp: Trong một số ngôn ngữ lập trình, bạn có thể tăng kích thước ngăn xếp được phép, nhưng điều này chỉ làm chậm quá trình và không giải quyết vấn đề gốc.
* Kiểm tra và giảm thiểu việc sử dụng ngăn xếp: Đảm bảo rằng bạn không sử dụng ngăn xếp một cách quá đà. Tránh khai báo các biến cục bộ quá lớn và tránh sử dụng đệ quy một cách không cần thiết.
* Sử dụng các công cụ kiểm tra mã lỗi: Các công cụ như trình biên dịch hoặc bộ kiểm tra mã có thể giúp phát hiện và báo cáo về các điểm tiềm ẩn có thể gây ra tràn bộ đệm.
* Kiểm tra và sửa lỗi logic: Một số tràn bộ đệm xảy ra do lỗi logic trong mã của bạn. Hãy kiểm tra mã của bạn để đảm bảo nó hoạt động như mong đợi.
* Sử dụng ngôn ngữ lập trình có quản lý bộ nhớ tự động: Một số ngôn ngữ lập trình như Python hoặc JavaScript có quản lý bộ nhớ tự động, giảm thiểu nguy cơ tràn bộ đệm.
* Kiểm tra và tối ưu hóa mã: Tối ưu hóa mã của bạn để giảm thiểu việc sử dụng bộ nhớ và tăng hiệu suất, có thể giúp giảm thiểu nguy cơ tràn bộ đệm.

1. **Các hình thức tấn công tràn bộ đệm khác**

Ngoài tràn bộ đệm thông thường, còn có một số hình thức tấn công tràn bộ đệm khác mà kẻ tấn công có thể sử dụng để xâm nhập vào hệ thống hoặc gây ra những vấn đề bảo mật. Dưới đây là một số hình thức phổ biến:

* Shellcode Injection: Đây là một loại tấn công khi kẻ tấn công tiêm vào một chương trình hoặc hệ thống một đoạn mã shellcode qua việc tràn bộ đệm. Khi thành công, kẻ tấn công có thể kiểm soát chương trình hoặc hệ thống và thực thi các lệnh bên trong shellcode.
* Return-to-libc (Return Oriented Programming - ROP): Thay vì thực thi mã shellcode, tấn công ROP sử dụng các "gadget" (các đoạn mã đã có sẵn trong thư viện hệ thống) để thực hiện các hành động khác nhau mà không cần chạy shellcode. Điều này làm tăng khó khăn trong việc phát hiện và ngăn chặn tấn công.
* Heap Overflow: Tương tự như tràn bộ đệm, heap overflow xảy ra khi dữ liệu được ghi vào vùng nhớ heap một cách quá mức, gây ra lỗi và có thể dẫn đến việc kiểm soát chương trình hoặc thực hiện các hành động không mong muốn.
* Format String Vulnerabilities: Loại tấn công này xảy ra khi một chuỗi định dạng không an toàn được sử dụng trong hàm như printf hoặc sprintf. Kẻ tấn công có thể sử dụng lỗ hổng này để đọc hoặc ghi vào vùng nhớ một cách không mong muốn.
* Integer Overflow: Trong trường hợp này, kẻ tấn công cố gắng ghi dữ liệu vào một biến có kiểu dữ liệu nguyên nhưng vượt quá giới hạn kiểu dữ liệu đó, gây ra tràn bộ đệm và có thể gây ra lỗi hoặc thực hiện các hành động không mong muốn.
* Các hình thức tấn công này đều là những nguy cơ lớn về bảo mật trong ứng dụng và hệ thống máy tính. Để phòng tránh, cần thực hiện kiểm tra bảo mật kỹ lưỡng, sử dụng các phương pháp lập trình an toàn và kiểm tra mã lỗi một cách định kỳ.

1. **Xác định tràn bộ đệm là gì và liệt kê các hậu quả có thể xảy ra.**

Khi một chương trình bị tràn bộ đệm, có thể xảy ra nhiều hậu quả khác nhau, phụ thuộc vào cách thức và mục đích của kẻ tấn công cũng như loại lỗ hổng bảo mật mà họ tận dụng. Dưới đây là một số hậu quả phổ biến của việc bị tràn bộ đệm:

* Làm suy yếu tính ổn định của chương trình: Tràn bộ đệm có thể làm cho chương trình bị crash hoặc hoạt động không đúng đắn, gây ra sự không ổn định trong hệ thống.
* Thực thi mã độc hại: Kẻ tấn công có thể chèn mã độc hại vào vùng nhớ tràn bộ đệm, sau đó khi chương trình tiếp tục thực thi, mã độc hại này được thực thi, cho phép kẻ tấn công kiểm soát máy tính hoặc thực hiện các hành động không mong muốn.
* Tiết lộ thông tin nhạy cảm: Dữ liệu nhạy cảm như mật khẩu, thông tin tài khoản hoặc các thông tin cá nhân khác có thể bị tiết lộ nếu chúng được lưu trong vùng nhớ bị tràn bộ đệm và sau đó được đọc bởi kẻ tấn công.
* Bypass các cơ chế bảo mật: Tràn bộ đệm có thể được sử dụng để bypass các cơ chế bảo mật như phân quyền, kiểm soát truy cập hoặc kiểm tra tính hợp lệ của dữ liệu.
* Tấn công từ xa (Remote Code Execution - RCE): Trong một số trường hợp, khi một ứng dụng bị tràn bộ đệm, kẻ tấn công có thể thực thi mã từ xa trên máy chủ mà ứng dụng đang chạy, cung cấp quyền truy cập không hợp pháp vào hệ thống.
* Quyền kiểm soát hệ thống: Nếu một kẻ tấn công có thể thực thi mã độc hại thành công, họ có thể có quyền kiểm soát hoặc quản lý hệ thống máy tính một cách không hợp pháp.

1. **Mô tả chi tiết cách hoạt động của tràn bộ đệm ngăn xếp.**

Tràn bộ đệm ngăn xếp là một lỗi phổ biến trong lập trình, đặc biệt là trong các ngôn ngữ sử dụng cấu trúc dữ liệu ngăn xếp như C và C++. Để hiểu cách hoạt động của tràn bộ đệm ngăn xếp, trước tiên chúng ta cần hiểu về cấu trúc dữ liệu ngăn xếp.

Ngăn xếp là một cấu trúc dữ liệu theo nguyên tắc "Last In, First Out" (LIFO), nghĩa là phần tử được thêm vào cuối (push) sẽ được lấy ra trước (pop). Trong môi trường lập trình, ngăn xếp thường được sử dụng để lưu trữ các biến cục bộ và các giá trị trả về của các hàm.

Khi một chương trình thực thi, một vùng nhớ cụ thể được cấp phát cho ngăn xếp. Các biến cục bộ và giá trị trả về của các hàm được lưu trữ trong vùng nhớ này. Khi một hàm được gọi, các tham số và các biến cục bộ của hàm đó được đặt vào ngăn xếp. Khi hàm hoàn thành, các giá trị này được loại bỏ khỏi ngăn xếp để giải phóng bộ nhớ.

Tràn bộ đệm ngăn xếp xảy ra khi một chương trình cố gắng ghi dữ liệu vào vùng nhớ ngăn xếp nhiều hơn so với kích thước đã được cấp phát cho nó. Điều này có thể xảy ra khi một hàm ghi quá nhiều dữ liệu vào ngăn xếp hoặc khi một chuỗi kí tự quá dài được sao chép vào một mảng cục bộ mà không có kiểm tra về độ dài.

Khi ngăn xếp bị tràn, các giá trị cực kỳ quan trọng như địa chỉ trở về (return address) của các hàm, các biến cục bộ và các thông tin quan trọng khác có thể bị ghi đè. Điều này có thể dẫn đến việc lỗi hoặc lỗi không xác định xảy ra khi chương trình cố gắng thực thi các hành động với dữ liệu không hợp lệ hoặc không mong muốn.

Kẻ tấn công có thể tận dụng lỗ hổng này bằng cách chèn các dữ liệu độc hại vào vùng nhớ ngăn xếp để kiểm soát luồng thực thi của chương trình hoặc thực hiện các hành động độc hại khác như thực thi mã độc hại.

1. **Xác định shellcode và mô tả việc sử dụng nó trong một cuộc tấn công tràn bộ đệm.**

Shellcode là một loạt các mã máy nhỏ được thiết kế để thực hiện một loạt các chức năng độc hại hoặc tấn công trong một môi trường máy tính cụ thể. Thường thì shellcode được viết bằng ngôn ngữ lập trình mã máy hoặc mã Assembly để chạy trên một nền tảng cụ thể như Windows, Linux, hoặc một loại kiến trúc nhất định.

Shellcode thường được sử dụng trong các cuộc tấn công tràn bộ đệm, nơi một kẻ tấn công cố gắng thiết lập và thực thi mã độc hại trên máy chủ hoặc hệ thống mục tiêu thông qua lỗ hổng tràn bộ đệm. Quá trình sử dụng shellcode trong một cuộc tấn công tràn bộ đệm thường diễn ra như sau:

* Tìm lỗ hổng: Kẻ tấn công tìm kiếm các ứng dụng hoặc dịch vụ mà họ có thể tận dụng để thực hiện cuộc tấn công tràn bộ đệm. Các ứng dụng thường dễ bị tấn công bao gồm các ứng dụng web, máy chủ email, trình duyệt, và các dịch vụ mạng khác.
* Chuẩn bị shellcode: Kẻ tấn công tạo ra hoặc sử dụng shellcode đã có sẵn để thực hiện các hành động độc hại mong muốn, như mở một cổng kết nối từ xa, tạo quyền quản trị, hoặc thực hiện các lệnh shell.
* Tạo payload: Kẻ tấn công tích hợp shellcode vào một payload (tải trọng) để gửi đến ứng dụng hoặc dịch vụ mục tiêu thông qua một gói tin mạng hoặc dữ liệu đầu vào. Thông thường, payload này được thiết kế để ghi đè lên vùng nhớ của ứng dụng và thực thi shellcode.
* Gửi và thực thi payload: Khi kẻ tấn công gửi payload chứa shellcode đến ứng dụng mục tiêu thông qua một kênh truyền nào đó, ví dụ như một yêu cầu HTTP hoặc một gói tin TCP, lỗ hổng tràn bộ đệm được kích hoạt và shellcode được thực thi trên máy chủ hoặc hệ thống mục tiêu.
* Thực hiện hành động độc hại: Sau khi shellcode được thực thi thành công, các hành động độc hại như lấy thông tin nhạy cảm, tạo quyền truy cập không hợp pháp, hoặc thực thi lệnh từ xa có thể được thực hiện bởi kẻ tấn công.

1. **Liệt kê các biện pháp phòng thủ khác nhau chống lại các cuộc tấn công tràn bộ đệm.**

Có nhiều biện pháp phòng thủ khác nhau mà tổ chức và nhà phát triển có thể triển khai để ngăn chặn các cuộc tấn công tràn bộ đệm. Dưới đây là một số biện pháp phòng thủ phổ biến:

* Kiểm tra và xác minh dữ liệu đầu vào: Điều này bao gồm việc sử dụng kiểm tra dữ liệu đầu vào hợp lệ để đảm bảo rằng dữ liệu được nhập vào không vượt quá kích thước được xác định và không chứa các ký tự đặc biệt có thể làm hỏng vùng nhớ.
* Sử dụng các ngôn ngữ lập trình an toàn: Chọn các ngôn ngữ lập trình có tính an toàn cao như Rust, Go, hoặc Java, có cơ chế bảo vệ tự nhiên để ngăn chặn các lỗi tràn bộ đệm.
* Sử dụng thư viện và frameworks an toàn: Sử dụng các thư viện và frameworks được phát triển chặt chẽ và được kiểm tra kỹ lưỡng để giảm thiểu rủi ro tràn bộ đệm.
* Kiểm tra mã lỗi và pentesting định kỳ: Thực hiện kiểm tra mã lỗi và kiểm tra an ninh định kỳ để phát hiện và khắc phục lỗ hổng tràn bộ đệm trước khi chúng được khai thác bởi kẻ tấn công.
* Sử dụng kỹ thuật DEP (Data Execution Prevention): Kỹ thuật DEP ngăn chặn việc thực thi mã từ các vùng nhớ chỉ đọc được, giảm thiểu khả năng thực hiện cuộc tấn công tràn bộ đệm.
* Sử dụng kỹ thuật ASLR (Address Space Layout Randomization): ASLR làm thay đổi vị trí của các vùng nhớ quan trọng trong không gian địa chỉ, làm khó khăn hơn cho kẻ tấn công để tìm ra vùng nhớ cụ thể để thực hiện cuộc tấn công.
* Sử dụng kỹ thuật canary: Canary là một giá trị được thêm vào trước return address trong ngăn xếp, giúp phát hiện các cuộc tấn công tràn bộ đệm bằng cách kiểm tra giá trị canary trước khi trở về từ một hàm.
* Tăng cường quản lý bộ nhớ: Sử dụng các công nghệ như StackGuard, ProPolice, hoặc Microsoft Buffer Security Check để ngăn chặn các cuộc tấn công tràn bộ đệm.
* Giảm thiểu quyền truy cập: Giảm thiểu quyền truy cập của các ứng dụng và dịch vụ để hạn chế tác động của các cuộc tấn công tràn bộ đệm nếu chúng được khai thác thành công.
* Tích hợp giải pháp bảo mật: Sử dụng các giải pháp bảo mật như tường lửa, IDS/IPS để phát hiện và ngăn chặn các cuộc tấn công tràn bộ đệm từ mạng bên ngoài.

1. **Liệt kê một loạt các kiểu tấn công tràn bộ đệm khác.**

**Dưới đây là một loạt các kiểu tấn công tràn bộ đệm khác nhau:**

* Heap Overflow: Tương tự như tràn bộ đệm ngăn xếp, heap overflow xảy ra khi dữ liệu được ghi vào vùng nhớ heap một cách quá mức, gây ra lỗi và có thể dẫn đến việc kiểm soát chương trình hoặc thực hiện các hành động không mong muốn.
* Format String Vulnerabilities: Loại tấn công này xảy ra khi một chuỗi định dạng không an toàn được sử dụng trong hàm như printf hoặc sprintf. Kẻ tấn công có thể sử dụng lỗ hổng này để đọc hoặc ghi vào vùng nhớ một cách không mong muốn.
* Integer Overflow: Trong trường hợp này, kẻ tấn công cố gắng ghi dữ liệu vào một biến có kiểu dữ liệu nguyên nhưng vượt quá giới hạn kiểu dữ liệu đó, gây ra tràn bộ đệm và có thể gây ra lỗi hoặc thực hiện các hành động không mong muốn.
* Return-to-libc (Return Oriented Programming - ROP): Thay vì thực thi mã shellcode, tấn công ROP sử dụng các "gadget" (các đoạn mã đã có sẵn trong thư viện hệ thống) để thực hiện các hành động khác nhau mà không cần chạy shellcode. Điều này làm tăng khó khăn trong việc phát hiện và ngăn chặn tấn công.
* Stack-based Buffer Overflow: Đây là loại tràn bộ đệm phổ biến nhất, xảy ra khi dữ liệu nhập vào vượt quá kích thước được chỉ định của một mảng hoặc một vùng nhớ cục bộ trên ngăn xếp. Khi điều này xảy ra, thông tin quan trọng như địa chỉ trở về (return address) có thể bị ghi đè, cho phép kẻ tấn công kiểm soát luồng thực thi của chương trình.
* Integer Underflow: Đây là loại tấn công tràn bộ đệm ngược, xảy ra khi một giá trị nguyên được giảm đến giá trị nhỏ hơn giới hạn dưới của kiểu dữ liệu, gây ra tràn bộ đệm và có thể dẫn đến lỗi hoặc hành động không mong muốn.
* Stack Pivot Attack: Loại tấn công này tận dụng một lỗ hổng trong quản lý bộ nhớ stack để thay đổi địa chỉ trở về (return address) của một hàm, dẫn đến việc thực thi các đoạn mã độc hại hoặc thực hiện các hành động không mong muốn.
* NOP Sled Attack: Tấn công này tận dụng một chuỗi các "no-operation" (NOP) instructions được đặt trước shellcode để đảm bảo rằng mã độc hại sẽ được thực thi ngay khi điều kiện tràn bộ đệm xảy ra.