TÌM HIỂU VỀ LỖI TRÀN BỘ ĐỆM (BUFFER OVERFLOW)

KS. Đào Hương Giang

Tổ NCPT An toàn thông tin.

***Tóm tắt**: Tấn công Buffer Overflow có nguyên nhân gần giống với tấn công SQL Injection, khi người dùng hay hacker cung cấp các biên đầu vào hay dữ liệu vượt quá khả năng xử lý của chương trình làm cho hệ thống bị treo dẫn đến từ chối dịch vụ hay có khả năng bị các hacker lợi dụng chèn các chỉ thị trái phép nhằm thực hiện các đoạn mã nguy hiểm từ xa. Vì vậy, trong bài báo này chúng ta sẽ tìm hiểu những khái niệm cơ bản nhất về tấn công tràn bộ đệm, các cách tấn công, cách phát hiện và cách phòng chống để nâng cao kiến thức và kỹ năng phòng chống lại các cuộc tấn công Buffer Overflow.*

1. GIỚI THIỆU.

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).

```
void buffover(int i)
 {
 byte buffer[5];
 strcpy(buffer,"123");
 }

void demo_function(int i)
{
byte buffer[8];
strcpy(buffer,"123456…shellcode");
}
```

Nguyên nhân gây ra lỗi Buffer Overflow của các chương trình, ứng dụng:

- 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.

Tác hại của lỗi Buffer Overflow.

- 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.

2. NỘI DUNG.

2.1. 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.

Ví dụ:

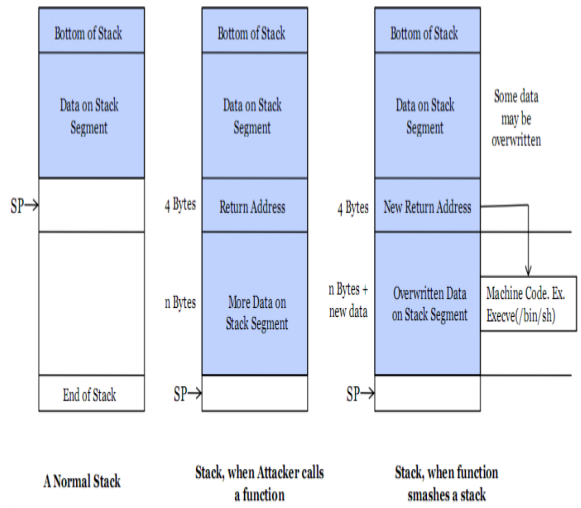
```
#include <string.h>

void f(char* s) {
    char buffer[10];
    strcpy(buffer, s);
}

void main(void) {
    f("01234567890123456789");
}

[root /tmp]# ./stacktest

Segmentation fault
```



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.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

void main(void) {
    char str[100] = scanf("%s");
    printf("%s", str);
}
```

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.

2.2. 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,...

2.3. Cách phát hiện lỗi Buffer Overflow.

Công nghệ biên dịch lý tưởng nhất để phát hiện là dùng chương trình C hoặc C++ để biết được thông tin về kích thước của dữ liệu trong mã nguồn. Một số thông tin có thể xuất phát từ lời khai báo của các biến, mô tả kiểu biến được sử dụng. Các thông tin khác đến từ các lệnh gọi chức năng trong chương trình. Trình biên dịch cần phải hiểu tất cả các thông tin này để tạo ra mã đúng.

Ví dụ:

```
void myfunc(char* somedata)
                {
        char buffer[10];
        strcpy(buffer, somedata);
        }
```

Trong ví dụ trên, trình biên dịch có thể hiểu một đại diện trong gian (IR) có kích thước bộ đệm là 10 và nó biết rằng dữ liệu đến từ một tham số quen thuộc. Tuy nhiên, nếu chỉ nhìn vào các tham số đó sẽ không đủ thông tin để biết làm thế nào các dữ liệu được đại diện từ số bé đến số lớn. Trình biên dịch tối ưu hóa điều này bằng các bước tạo mã code. Đầu tiên, nhìn vào tất cả các mã code trong dòng tiếp theo. Tiếp theo là tìm vào các đoạn mã từ những lệnh gọi thường xuyên. Và nó được gọi là phân tích interprocedural. Tưởng tượng rằng ví dụ trên cũng bao gồm mã này.

```
void yourfunc(char* somedata)
                {
         char yourbuffer[50];
         /* fill yourbuffer */
         myfunc( yourbuffer );
         }
```

Các yourfunc() gọi myfunc() trong bộ đệm địa phương của mình chứa đầu vào dữ liệu. Sử dụng phân tích interprocedural, trình biên dịch xây dựng một đồ thị kết nối yourfunc() với myfunc() bằng cách kết hợp kiến thức của yourfunc() và myfunc(). Và có thể hiểu rằng nó đang cố gắng để sao chép 50 byte dữ liệu vào bộ đệm 10byte dữ liệu. Và như vậy lỗi tràn bộ đệm có thể xảy ra.

Tưởng tượng " yourbuffer" sẽ bị đầy như sau:

```
fread (yourbuffer, 1, 49, file);
```

Một trình biên dịch mà hiểu các thuộc tính của thư viện thời gian tiêu chuẩn sẽ biết rằng bộ đệm bây giờ chứa dữ liệu từ một thời điểm không xác định trong hệ thống tập tin bên ngoài chương trình và có khả năng bị nguy hiểm. Mã myfunc() chính là mã mở cửa cho một cuộc tấn công độc hại và yourbuffer được làm đầy với một chuỗi liên tục.

Sử dụng phân tích mã nguồn để tìm lỗi tràn bộ đệm trong mã thực thi một cách chính xác đòi hỏi phải hiểu biết sâu về code. Và công nghệ của trình biên dịch được thiết kế với mục đích tìm kiếm lỗi tràn bộ đệm. Nó hiệu quả và chính xác hơn các phương pháp khác và cung cấp các cách nhìn sâu sắc về tình trạng bảo mật của từng đoạn mã mà nó phân tích.

2.4. Biện pháp ngăn chặn.

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.
- Polumorphic code: mã đa hình.
- Seft-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.

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 Pythol, Ada, Lisp,...
- Ngoài ra, các môi trường Java hay .Net cũng đòi hỏi kiểm tra biên đối với tất cả các mảng.

Sử dụng các 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 biên có thể làm giảm sự xuất hiện và ảnh hưởng của các hiện tượng tràn bộ đệm. Các thư viện an toàn gồm có: The Better String Library, Arri Buffer API, Vstr.

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 hai phần, một phần dành cho dữ liệu và một phần dành cho các bước trả về hàm, sự phân chia này được dùng trong ngôn ngữ Forth.

Bảo vệ không gian thực thi: Kỹ thuật này ngăn chặn việc thực thi mã tại stack hay heap. Hacker có thể sử dụng tràn bộ đệm để chèn một đoạn mã tùy ý vào bộ nhớ của chương trình, với việc bảo vệ không gian thực thi thì mọi cố gắng chạy đoạn mã đó sẽ gây ra một ngoại lệ. Một số CPU hỗ trợ tính năng có tin bit NX (No eXecute) hoặc bit XD (eXecute Disable). Khi kết hợp với phần mềm các tính năng này có thể được dùng để đánh dấu những trang dữ liệu (chẳng hạn như các trang chứa stack và heap) là đọc được nhưng không thực hiện được.

Ngẫu nhiên hóa sơ đồ không gian địa chỉ (Address space layout randomization ASLR): là một tính năng an ninh máy tính có liên quan đến việc sắp xếp các vùng dữ liệu quan trọng (thường bao gồm nơi chứa mã thực thi và vị trí các thư viện, heap và stack) một cách ngẫu nhiên trong không gian địa chỉ của một tiến trình.

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 kỹ 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 quét gói tin không phải 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.

3. THẢO LUẬN.

Hiện nay, hầu hết các máy tính đều có thể bị tấn công Buffer Overflow, do đó việc tìm hiểu về cuộc tấn công Buffer Overflow là cần thiết cho bất kỳ một người quản trị, hay một người dùng máy tính thông thường.

Như đã nói ở trên, tấn công Buffer Overflow có thể được thực hiện một cách dễ dàng, và kẻ tấn công có thể khai thác dữ liệu bằng những ngôn ngữ lập trình rất phổ biến. Vì vậy, để phòng chống, các doanh nghiệp, tổ chức, cá nhân ngoài việc sử dụng các ngôn ngữ lập trình cao hơn hay những thư viện an toàn,... thì còn phải thường xuyên tổ chức các cuộc rà quét để tìm ra các lỗ hổng, ngăn chặn kịp thời các cuộc tấn công tràn bộ đệm.

4. KẾT LUẬN.

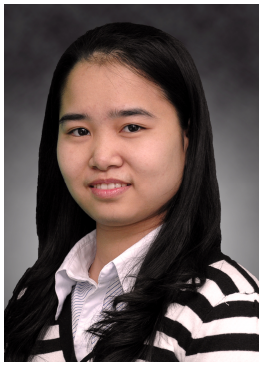
Thực tế chỉ ra rằng, các cuộc tấn công tràn bộ đệm vẫn thường xuyên xảy ra và gây hậu quả nghiêm trọng cho hệ thống vì nó có thể can thiệp trực tiếp vào mã nguồn. Tuy rằng bài báo đã chỉ ra các dạng cơ bản của tấn công Buffer Overflow và các cách phòng chống nhưng với sự phát triển của công nghệ,

các hacker không ngừng phát triển các cách tấn công mới vào hệ thống. Vì vậy, bài báo chỉ có thể giúp người đọc phần nào biết đến cách tấn công Buffer Overflow và mức độ nguy hiểm của các cuộc tấn công bằng phương pháp Buffer Overflow. Bài báo cũng có thể là cơ sở phần nào giúp người đọc dựa vào đó để có thể nghiên cứu sâu hơn về cuộc tấn công này từ đó đưa ra những biện pháp phòng chống hiệu quả và triệt để nhất.

5. TÀI LIỆU THAM KHẢO.

1. *Stack based Buffer Overflow Exploitation Tutorial – Saif IE Sherei.*
2. *Buffer Overflow Vulnerabilities and Attacks – Lecture Notes (Syracuse University).*
3. *Different Techniques to Prevent Buffer Overflow – Kamanashis Biswas.*
4. *Buffer Overflow Attacks – James C.Foster.*
5. *https://www.owasp.org/index.php/Buffer_overflow_attack*

Thông tin tác giả:



Đào Hương Giang

Sinh năm: 1990.

Lý lịch khoa học: Tốt nghiệp đại học ngành Công nghệ Thông tin – Học viện Công nghệ Bưu chính Viễn thông năm 2012.

Lĩnh vực nghiên cứu hiện nay: Giải pháp bảo mật công nghệ an ninh ứng dụng.

Email: giangdh@ptit.edu.vn