PHÁT HIỆN LỖI VÀ LỖ HỔNG PHẦN MỀM

Bài 08. Một số lỗ hổng khác

Tài liệu tham khảo

1. Đặng Vũ Sơn, Vũ Đình Thu, "**Phát hiện lỗi và lỗ hổng phần mềm**", Hv KTMM, 2013
2. Nguyễn Thành Nam, **"Nghệ thuật tận dụng lỗi phần mềm (Chương 4)"**, NXB KHKT, 2009

1. Lỗi tràn số nguyên
2. Lỗi off-by-one
3. Lỗi race condition
4. Lỗi kiểm tra dữ liệu

1. Lỗi tràn số nguyên
2. Lỗi off-by-one
3. Lỗi race condition
4. Lỗi kiểm tra dữ liệu

Kiểu số nguyên

Type	Kích thước	Phạm vi giá trị
char	1 byte	-128 to 127 hoặc 0 to 255
unsigned char	1 byte	0 tới 255
signed char	1 byte	-128 tới 127
int	2 or 4 bytes	-32,768 tới 32,767 hoặc -2,147,483,648 tới 2,147,483,647
unsigned int	2 or 4 bytes	0 tới 65,535 hoặc 0 tới 4,294,967,295
short	2 bytes	-32,768 tới 32,767
unsigned short	2 bytes	0 tới 65,535
long	8 bytes	-9223372036854775808 tới 9223372036854775807
unsigned long	8 bytes	0 tới 18446744073709551615

Lỗi tràn số nguyên

- Tràn số nguyên là việc kết quả của phép tính trên số nguyên vượt quá phạm vi biểu diễn của kiểu dữ liệu nguyên
- Ví dụ

```
unsigned int a = 0xE0000020;
unsigned int b = 0;
a = a + 0x20000020;
b = b - 1;
```

Tràn số nguyên

- Số bù một của A = đảo giá trị các bit của A
- (Số bù hai của A) = (Số bù một của A) +1 = -A

Cho A = 0010 0101
Số bù một = 1101 1010
\+ 1
Số bù hai = 1101 1011

A = 0010 0101
Số bù hai = + 1101 1011
1 0000 0000 = 0
(bỏ qua bit nhớ ra ngoài)

Tràn số nguyên

- Với n = 8 bit, Biểu diễn được các giá trị từ 0 đến 255

```
0000 0000   =   0
0000 0001   =   1
0000 0010   =   2
0000 0011   =   3
    ...
1111 1111   = 255
```

Chú ý:

```
    1111 1111
+   0000 0001
  1 0000 0000
```

Vậy: 255 + 1 = 0 ?

→ do tràn nhớ ra ngoài

Tràn số nguyên

- Với n = 8 bit, Biểu diễn được các giá trị từ -128 đến +127

```
0000 0000   =    0
0000 0001   =   +1
0000 0010   =   +2
0000 0011   =   +3
...
0111 1111   = +127
1000 0000   = - 128
1000 0001   = - 127
...
1111 1110   =   -2
1111 1111   =   -1
```

Chú ý:

+127 + 1 = -128

-128 - 1 = +127

→ do tràn xảy ra

Ví dụ khai thác lỗi tràn số nguyên

```
uint nresp;
.....
nresp=packet_get_int();
if (nrespn>0) {
    response=xmalloc(nresp*sizeof(char*));
    for (i=0;i<nresp;i++)
        response[i]=packet_get_string(NULL);
}
packet_check_eom();
```

Ví dụ khai thác lỗi tràn số nguyên

```
uint nresp;
.....
nresp=packet_get_int();
if (nrespn>0) {
    response=xmalloc(nresp*sizeof(char*));
    for (i=0;i<nresp;i++)
        response[i]=packet_get_string(NULL);
}
packet_c
```

- sizeof(char*) = 4
- Nếu nresp = 0x40000020 kết quả phép nhân là 0x80
- Cấp phát 0x80 bytes
- Nhưng đọc vào nresp chuỗi → tràn!

1. Lỗi tràn số nguyên
2. Lỗi off-by-one
3. Lỗi race condition
4. Lỗi kiểm tra dữ liệu

Off-by-one

- Lỗi dư (tràn) 1 phần tử
- Là một dạng lỗi tràn bộ nhớ, xuất hiện do tính toán sai độ dài một mảng.. Lỗi này xảy ra do việc tính sai vị trí của phần tử cuối cùng hoặc hiểu sai cơ chế hoạt động của chỉ số mảng.

Off-by-one

```
void process_string(char *src)
{
    char dest[16];
    for (i=0; src[i]&&(i<=sizeof(dest)); i++)
        dest[i] =src[i];
    ……….
}
```

Off-by-one

```
void process_string(char *src)
{
    char dest[16];
    for (i=0; src[i]&&(i<=sizeof(dest)); i++)
        dest[i] =src[i];
    ……..
}
```

- Hàm đọc các ký tự từ tham số src của nó và lưu chúng vào trong dest.
- Đoạn mã này có tác dụng ngăn lỗi tràn bộ nhớ nếu src có nhiều hơn 16 ký tự
- nhưng nó có một vấn đề đó là nó có thể ghi một phần tử ra ngoài dest.

Off-by-one. Ví dụ 2

```
int get_user(char* user)
{
    char buf[16];
    if(strlen(user)>sizeof(buf))
        die(“error:user string too long\n”);
    strcpy(buf,user);
}
```

Off-by-one. Ví dụ 2

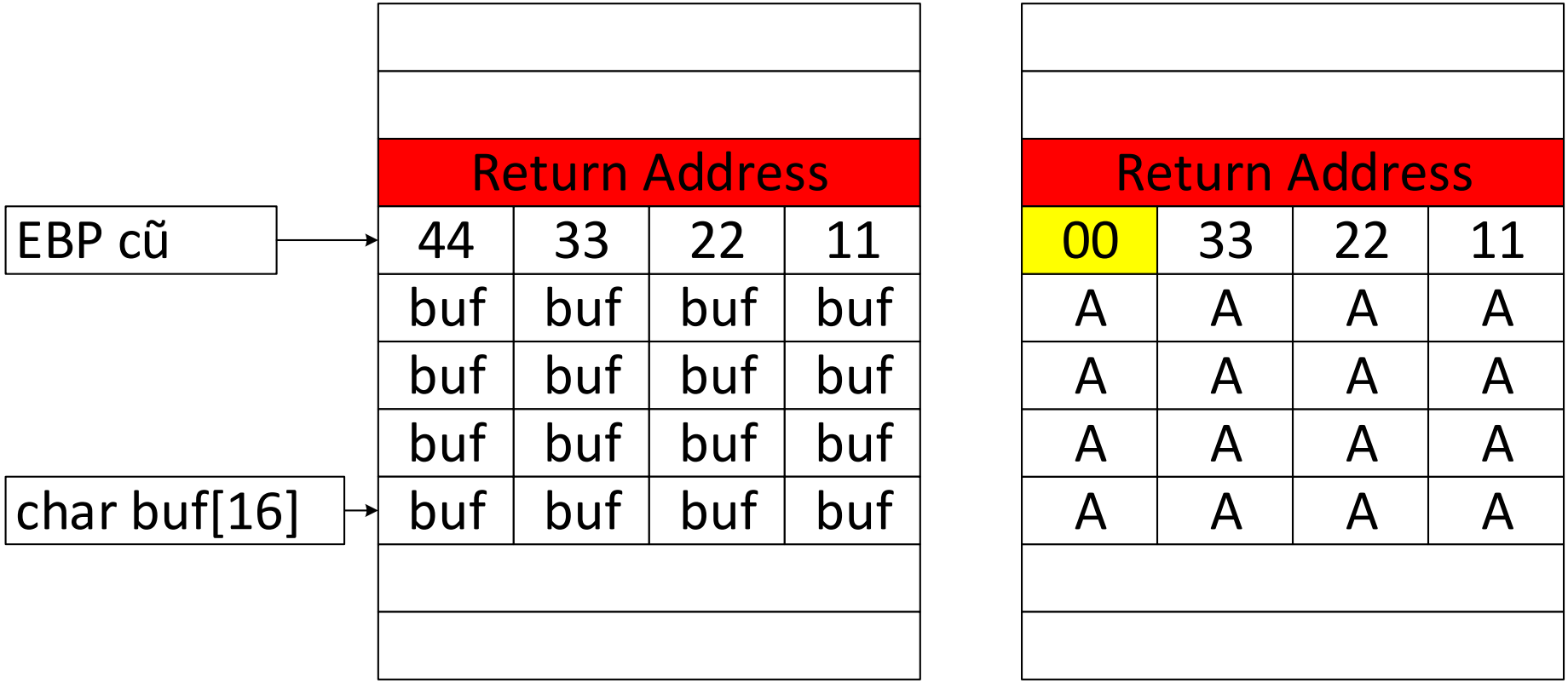
```
int get_user(char* user)
{
    char buf[16];
    if(strlen(user)>sizeof(buf))
        die("erro er string too long\n");
    strcpy(buf
}
```

- Đoạn mã này sử dụng hàm strlen() để kiểm tra liệu có đủ khoảng trống để sao chép username vào buffer hay không.
- nếu chuỗi user được đưa vào có đúng 16 ký tự, thì 1 kí tự thừa bị ghi vào buf.

Off-by-one

- Là lỗi ít gặp và khó khai thác
- Vì chỉ can thiệp trực tiếp được 01 phần tử (1 byte, 4 byte)
- Kỹ thuật khai thác thường gặp: ghi đè 1 byte (và thay đổi) giá trị của thanh ghi cơ sở (EBP) trong stack.

Off-by-one


EBP cũ
char buf[16]
Return Address
44 33 22 11
buf buf buf buf
buf buf buf buf
buf buf buf buf
buf buf buf buf
Return Address
00 33 22 11
A A A A
A A A A
A A A A
A A A A

1. Lỗi tràn số nguyên
2. Lỗi off-by-one
3. Lỗi race condition
4. Lỗi kiểm tra dữ liệu

Khái niệm

- Race Condition = TOCTOU
- TOCTTOU – Time of Check To Time of Use
- Check – Kiểm tra điều kiện
- Use – Thực thi tác vụ (nếu điều kiện được thỏa mãn)
- Lỗi này thường có trong các ứng dụng truy cập tập tin (nhưng không loại trừ các ứng dụng khác)

UNIX File System Security

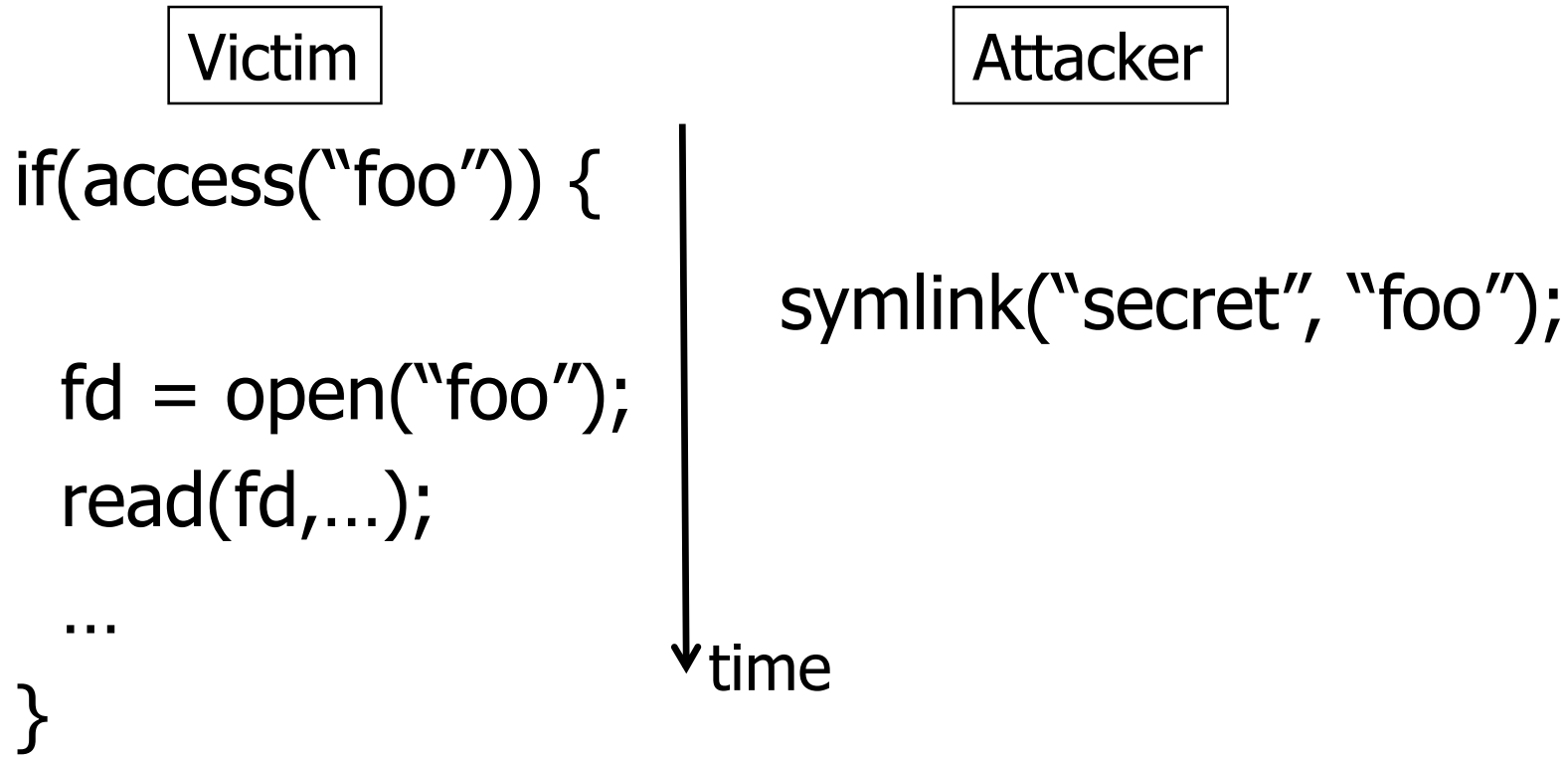
- Access control: người dùng chỉ được phép truy cập file khi có thẩm quyền tương ứng
- SEUID: thuộc tính cho phép chương trình chạy với quyền nào đó khác với quyền của người dùng
  - Một số chương trình (ping, mount...) được setuid root để có được thẩm quyền truy cập hệ thống, dù được chạy bởi user nào
- System call access() cho phép kiểm tra thẩm quyền của người dùng

Kịch bản Race Condition

- Chương trình A: có năng lực lớn, có khả năng truy cập đối tượng H, L
- Người dùng U: không được phép truy cập H, chỉ được truy cập L
- U lợi dụng A để truy cập H
  - U yêu cầu A truy cập L
  - A kiểm tra thẩm quyền → OK → chuẩn bị...
  - U xóa L cũ, tạo L mới là symlink của H
  - ...truy cập L!!!!

TOCTTOU Example – setuid

- **Tấn công**: người dùng bình thường muốn đọc file vượt quá thẩm quyền.



TOCTTOU Example – setuid

- **Tấn công**: người dùng bình thường muốn đọc file vượt quá thẩm quyền ("secret").

Victim	Attacker
if(access("foo")) {	
	symlink("secret", "foo");

Bước 1:

- Attacker tạo symlink "foo" tới một file bất kỳ mà attacker có quyền truy cập
- Attacker yêu cầu một chương trình setuid-root truy cập file "foo"
- Victim kiểm tra quyền truy cập của attacker. Kết quả là OK → chuẩn bị đáp ứng yêu cầu truy cập.

TOCTTOU Example – setuid

- **Tấn công**: người dùng bình thường muốn đọc file vượt quá thẩm quyền ("secret").

Victim	Attacker
if(access("foo")) {	
fd = open("foo");	symlink("secret", "foo");

Bước 2:

- Ở thời điểm sau hàm access() và trước hàm open() của Victim, Attacker thay đổi symlink, trỏ đến file không có quyền truy cập "secret".

TOCTTOU Example – setuid

- **Tấn công**: người dùng bình thường muốn đọc file vượt quá thẩm quyền ("secret").

Victim	Attacker
if(access("foo")) {	
fd = open("foo");	symlink("secret", "foo");
read(fd,…);	

Bước 3:

- Victim thực hiện truy cập file "foo" khi mà nó không còn trỏ đến file ban đầu nữa.
- Hacker đạt mục đích!

Directory Removal Exploit

Recursive removal of a directory tree (GNU file utilities)

Original tree is /tmp/dir1/dir2/dir3

chdir("/tmp/dir1/dir2/dir3")

unlink("*")

chdir("..")

rmdir("dir3")

unlink("*")

chdir("..")

rmdir("dir2")

unlink("*")

rmdir("/tmp/dir1")

Nếu Attacker thực thi câu lệnh **"mv /tmp/dir1/dir2/dir3 /tmp"** ngay tại thời điểm này

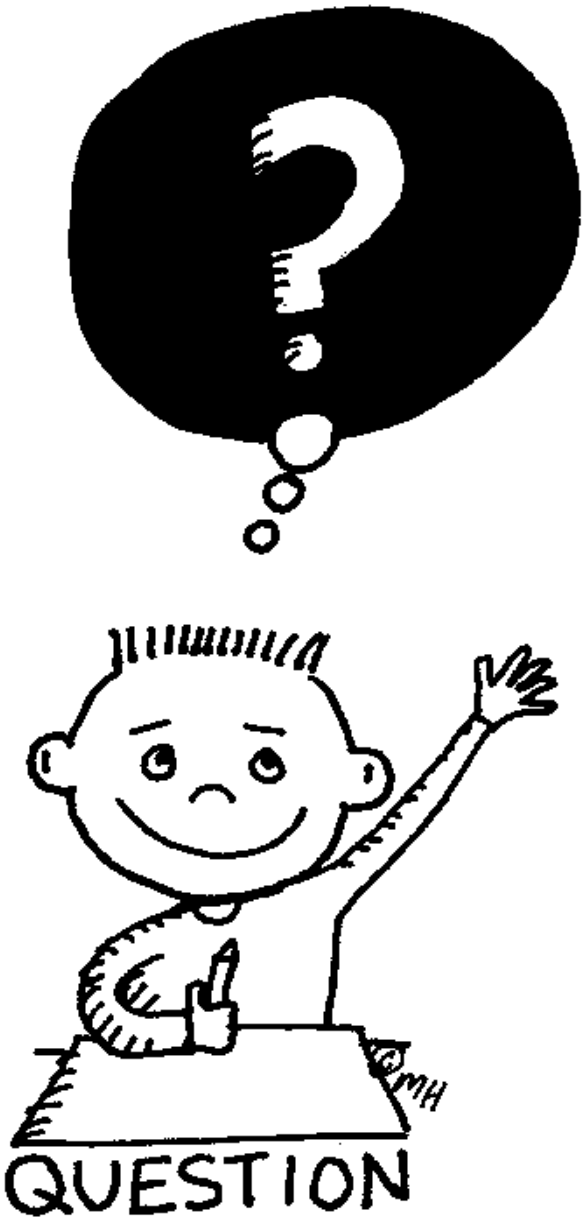
Thì câu lệnh này sẽ chuyển đến /tmp

Câu lệnh này sẽ chuyển đến /

Câu lệnh này xóa toàn bộ /

1. Lỗi tràn số nguyên
2. Lỗi off-by-one
3. Lỗi race condition
4. Lỗi kiểm tra dữ liệu

- OS Command Injection
- SQL Injection
- XSS


QUESTION