An toàn Phần mềm
Lỗi phần mềm

Trần Đức Khánh
Bộ môn HTTT – Viện CNTT&TT
ĐH BKHN

Lỗi phần mềm

- Một số lỗi phần mềm thường gặp
- Các biện pháp an toàn
  - Kiểm thử (Testing)
  - Kiểm định hình thức (Formal Verification)
  - Lập trình an toàn (Secure Coding)

Lỗi phần mềm

- Một số lỗi phần mềm thường gặp
- Các biện pháp an toàn
  - Kiểm thử (Testing)
  - Kiểm định hình thức (Formal Verification)
  - Lập trình an toàn (Secure Coding)

Lỗi phần mềm

- Lập trình viên thường mắc lỗi
  - không cố ý
  - không độc hại
  - nhưng đôi khi gây hậu quả nghiêm trọng

Một số lỗi phần mềm thường gặp

- Tràn bộ đệm (Buffer Overflow)
  - Array Index Out of Bound
- Không đầy đủ (Incomplete Mediation)
  - Implicit Cast, Integer Overflow
- Đồng bộ (Synchronization)
  - File stat()/open()

Lỗi tràn bộ đệm: ví dụ 1

```
1. char buf[80];
2. void vulnerable() {
3.     gets(buf);
4. }
```

- gets() đọc các bytes từ stdin và ghi vào buf
- Điều gì xảy ra nếu đầu vào có hơn 80 byte
  - gets() sẽ ghi đè lên bộ nhớ vượt ra ngoài phần bộ nhớ của buf
  - Đây là một lỗi phần mềm

Lỗi tràn bộ đệm: ví dụ 2

1. char buf[80];
2. int authenticated = 0;
3. void vulnerable() {
4. gets(buf);
5. }

- Thủ tục login sẽ gán biến authenticated khác 0 nếu người dùng có mật khẩu
- Điều gì xảy ra nếu đầu vào có hơn 80 byte
  - authenticated ở ngay sau buf
  - Kẻ tấn công nhập 81 bytes ghi bytes thứ 81 khác 0 ngay vào vùng bộ nhớ của authenticated

Lỗi tràn bộ đệm: ví dụ 3

```
1. char buf[80];
2. int (*fnptr)();
3. void vulnerable() {
4.     gets(buf);
5. }
```

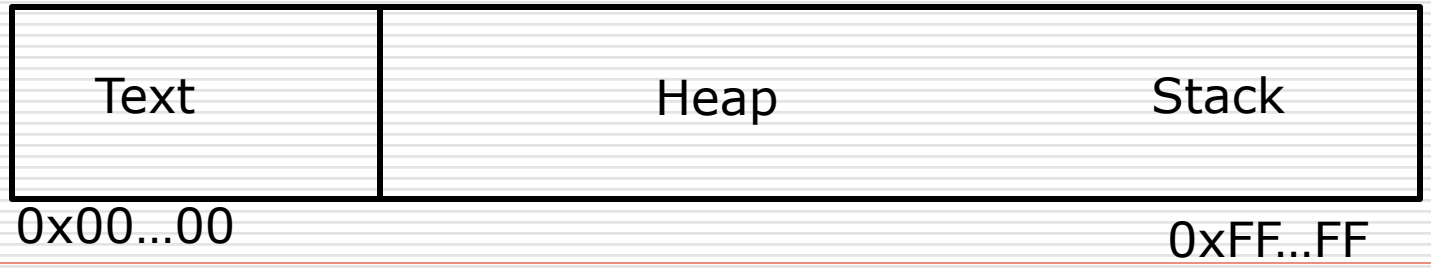
- Điều gì xảy ra nếu đầu vào có hơn 80 byte
  - Ghi địa chỉ bất kỳ vào int (*fnptr)()
  - int (*fnptr)() trỏ đến mã của hàm khác
  - Kẻ tấn công nhập mã độc kèm theo sau là địa chỉ để ghi đè lên (*fnptr)()

Khai thác lỗi tràn bộ đệm

- Đoạt quyền kiểm soát máy chủ
- Phát tán sâu
  - Sâu Morris phát tán thông qua khai thác lỗi tràn bộ đệm (ghi đè lên authenticated flag trong in.fingerd)
- Tiêm mã độc

Quản lý bộ nhớ chương trình C

- Text
  - Mã thực thi
- Heap
  - Kích thước tăng/giảm khi các đối tượng được cấp phát/huỷ
- Stack
  - Kích thươc tăng giảm khi hàm được gọi/trả về
  - Gọi hàm sẽ push stack frame lên stack



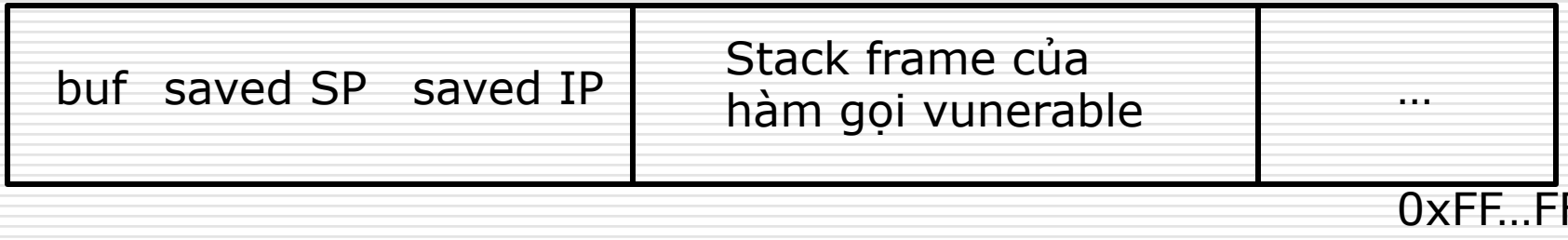
Thực thi chương trình C

- Thanh ghi con trỏ lệnh (IP) trỏ về lệnh kế tiếp
- Hàm gọi chuẩn bị tham số trên stack
- Gọi hàm
  - Lưu IP hiện tại lên stack (địa chỉ trở về)
  - Nhảy đến địa chỉ bắt đầu text của hàm được gọi
- Chương trình dịch thêm vào phần cuối mỗi hàm
  - Lưu con trỏ stack (SP) hiện tại lên stack
  - Cấp phát stack frame cho biến cục bộ (thay đổi SP)
- Hàm trở về
  - Tìm lại SP cũ và IP cũ
  - Tiếp tục thực thi lệnh trỏ bởi IP

Thực thi chương trình C: ví dụ

```
1. void vulnerable() {
2.       char buf[80];
3.       gets(buf);
4.  }
```

- Khi vunerable() được gọi stack frame sẽ được push lên stack
- Nếu buf quá lớn, saved SP và saved IP sẽ bị ghi đè



Lỗi không đầy đủ

```
1.  char buf[80];
2.  void vulnerable() {
3.      int len = read_int_from_network();
4.      char *p = read_string_from_network();
5.      if (len > sizeof buf) {
6.          error("length too large, nice try!");
7.          return;
8.      }
9.      memcpy(buf, p, len);
10. }
11. void *memcpy(void *dest, const void *src, size_t n);
12. typedef unsigned int size_t;
```

- Điều gì sẽ xảy ra nếu lên là một số âm
  - copy một đoạn bộ nhớ khổng lồ

Lỗi đồng bộ

```
1.  int openfile(char *path) {
2.      struct stat s;
3.      if (stat(path, &s) < 0)
4.          return -1;
5.      if (!S_ISRREG(s.st_mode)) {
6.          error("only allowed to regular files; nice try!");
7.          return -1;
8.      }
9.      return open(path, O_RDONLY);
10. }
```

- Điều gì sẽ xảy ra?
  - trạng thái hệ thống thay đổi giữa stat() và open()

Lỗi phần mềm

- Một số lỗi phần mềm thường gặp
- Các biện pháp an toàn
  - Kiểm thử (Testing)
  - Kiểm định hình thức (Formal Verification)
  - Lập trình an toàn (Secure Coding)

Lỗi phần mềm

- Một số lỗi phần mềm thường gặp
- Các biện pháp an toàn
  - Kiểm thử (Testing)
  - Kiểm định hình thức (Formal Verification)
  - Lập trình an toàn (Secure Coding)

Kiểm thử

- Mục đích của kiểm thử là tìm ra lỗi của hệ thống
  - Nếu không tìm ra lỗi, chúng ta hi vọng rằng hệ thống là an toàn

Quy trình kiểm thử

1. Đơn vị (Unit Testing)
2. Tích hợp (Integration Testing)
3. Chức năng (Function Testing)
4. Hiệu năng (Performance Testing)
5. Công nhận (Acceptance Testing)
6. Cài đặt (Installation Testing)

Một số loại hình kiểm thử đặc biệt

- Hồi quy (Regression Testing)
  - Nếu hệ thống có thay đổi, chỉnh sửa
- Xoắn (Fuzz Testing)
  - Các trường hợp đặc biệt, dễ bị khai thác và tấn công

Các tiếp cận trong kiểm thử

- Hộp đen (Black-box)
  - Không có thông tin về cấu trúc bên trong của phần mềm
  - Dùng cho tất cả các mức của quy trình kiểm thử
- Hộp trắng (White-box)
  - Biết cấu trúc bên trong của phần mềm
  - Thường dùng cho kiểm thử đơn vị
- Hộp xám (Grey-box)
  - Hỗn hợp
    - Đen: kiểm thử
    - Trắng: thiết kế ca kiểm thử

Lỗi phần mềm

- Một số lỗi phần mềm thường gặp
- Các biện pháp an toàn
  - Kiểm thử (Testing)
  - Kiểm định hình thức (Formal Verification)
  - Lập trình an toàn (Secure Coding)

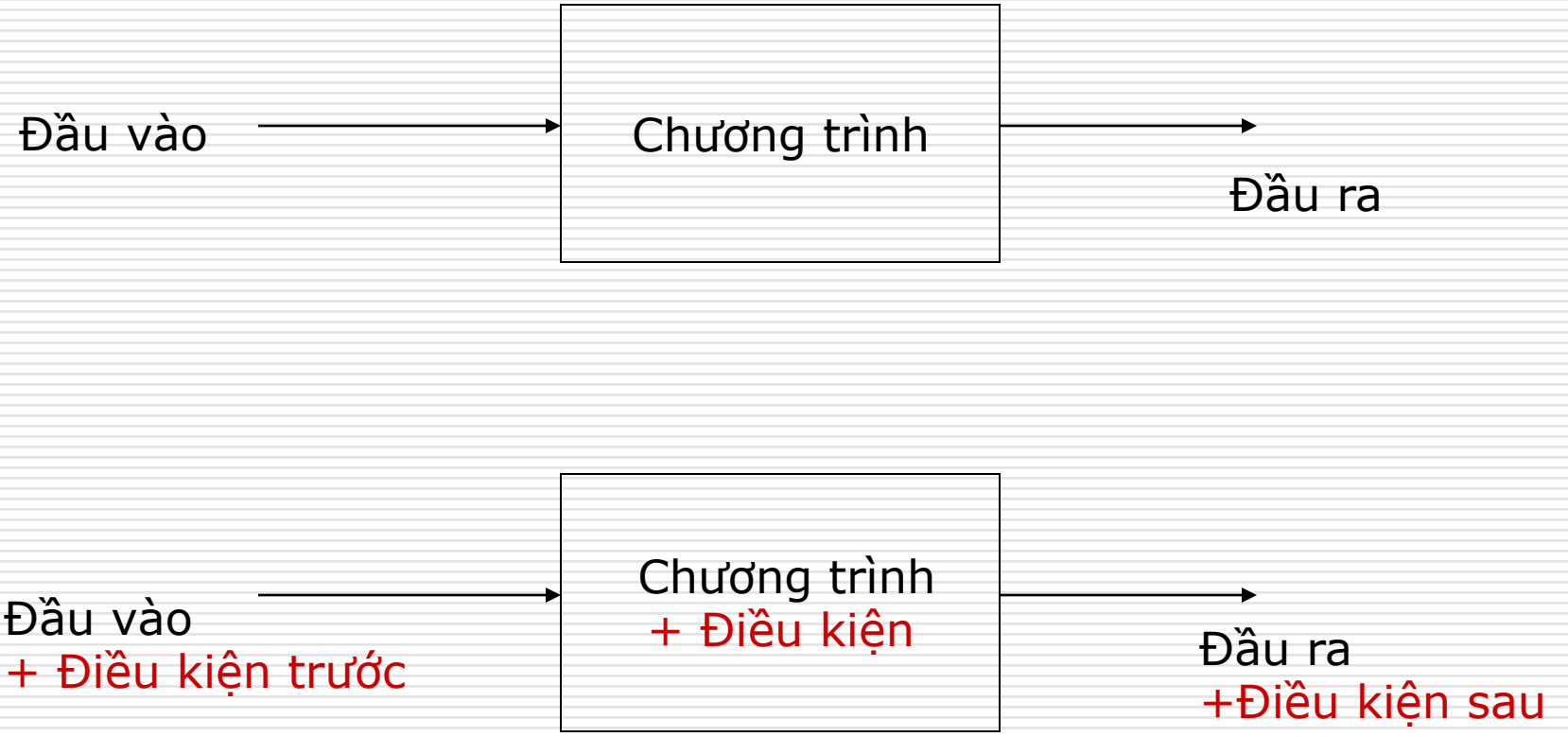
Kiểm định hình thức

- Mục đích của kiểm định hình thức là chứng minh hệ thống an toàn

Các tiếp cận trong kiểm định hình thức

- Kiểm định mô hình (Model checking)
  - Phần mềm được đặc tả bằng một mô hình
  - Quá trình kiểm định thực hiện bằng cách duyệt tất cả các trạng thái thông qua tất cả các chuyển tiếp
- Suy diễn logic (Logical Inference)
  - Đầu vào của phần mềm bị ràng buộc bằng một biểu thức logic
  - Tương tự với đầu ra
  - Bản thân phần mềm cũng bị ràng buộc bằng một biểu thức logic

Kiểm định hình thức sử dụng suy diễn logic



Điều kiện trước (Precondition)

```
1. /* Requires: n >= 1 */

2. int fact(int n) {
3.     int t;
4.     if (n == 1)
5.         return 1;
6.     t = fact(n-1);
7.     t *= n;
8.     return t;
9. }
```

Điều kiện sau (Postcondition)

```
1. /* Ensures: returnvalue >= 0 */

2. int fact(int n) {
3.     int t;
4.     if (n == 1)
5.         return 1;
6.     t = fact(n-1);
7.     t *= n;
8.     return t;
9. }
```

Điều kiện trong chương trình

```
1.  int fact(int n) {
2.      int t;
3.      if (n == 1)
4.          return 1;
5.              /* n>=2 */
6.      t = fact(n-1);
7.              /* t>=0 */
8.      t *= n;
9.              /* t>=0 */
10.     return t;
11. }
```

Điều kiện

```
1.  /* Requires: n >= 1; Ensures: returnvalue >= 0 */

2.  int fact(int n) {
3.      int t;
4.      if (n == 1)
5.          return 1;
6.              /* n>=2 */
7.      t = fact(n-1);
8.              /* t>=0 */
9.      t *= n;
10.             /* t>=0 */
11.     return t;
12. }
```

Lỗi phần mềm

- Một số lỗi phần mềm thường gặp
- Các biện pháp an toàn
  - Kiểm thử (Testing)
  - Kiểm định hình thức (Formal Verification)
  - Lập trình an toàn (Secure Coding)

Lập trình an toàn (Secure Coding)

- Nguyên tắc
  - Mô đun (Modularity)
  - Đóng gói (Encapsulation)
  - Giấu thông tin (Information Hiding)

Mô đun

- Thiết kế các hợp phần
  - Một mục tiêu/nhiệm vụ
  - Nhỏ
  - Đơn giản
  - Độc lập

Đóng gói

- Giấu thông tin về cách thức cài đặt các hợp phần
  - Ví dụ: lớp ảo C++, giao diện Java
- Giảm thiểu chia xẻ giữa các hợp phần
  - Ví dụ: các thư viện
- Các hợp phần tương tác thông qua các giao diện
  - Ví dụ: tương tác giữa các đối tượng thông qua các phương thức

Giấu thông tin

- Một hợp phần như một hộp đen nhìn từ phía ngoài
  - Ví dụ: một lớp C++, Java
- Các phần tử bên ngoài không thể thay đổi sữa chữa thông tin một cách ác ý và trái phép
  - Ví dụ: các thuộc tính private, protected

Lập trình an toàn (Secure Coding)

- Một số quy tắc thực hành
  - Sử dụng một chuẩn lập trình
  - Lập trình phòng thủ
    - Kiểm tra dữ liệu đầu vào/đầu ra
    - Sử dụng đặc quyền thấp nhất có thể
  - Thiết kế theo chính sách an toàn
  - Sử dụng các công cụ đảm bảo chất lượng
    - Kiểm thử
    - Kiểm định
    - Duyệt lại mã