HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
Posts and Telecommunications Institute of Technology

KIỂM THỬ XÂM NHẬP

KHOA AN TOÀN THÔNG TIN
TS. ĐINH TRƯỜNG DUY

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
Posts and Telecommunications Institute of Technology

KIỂM THỬ XÂM NHẬP

Ôn tập kiến thức

KHOA AN TOÀN THÔNG TIN
TS. ĐINH TRƯỜNG DUY

Biên soạn từ bài giảng: Nguyễn Ngọc Điệp, Bài giảng Kiểm thử xâm nhập, Học viện Công nghệ Bưu chính Viễn thông, 2021.

Mục lục

1. Kiến trúc máy tính và hệ điều hành
2. Linux
3. C
4. x86 Assembly

1. Kiến trúc máy tính và hệ điều hành

- Tham khảo slide bài giảng của giảng viên Nguyễn Thị Ngọc Vinh.

2. Linux

- Là hệ điều hành mã nguồn mở
- Linux được phát triển trên nền tảng UNIX và sử dụng kernel (nhân hệ thống) Linux.
- Được viết bằng ngôn ngữ C và hợp ngữ (assembly)
- Sử dụng ELF (Executable and Linkable Format) files

http://overthewire.org/wargames/bandit/

Một số lệnh Linux cơ bản thường dùng

- ls
  - liệt kê nội dung thư mục
- cd [path]
  - Thay đổi thư mục
  - ".." đi đến thư mục trước
- pwd
  - In ra thư mục làm việc
- man [command]
  - Hướng dẫn sử dụng lệnh
- apropos [whatever]
  - Đưa ra hướng dẫn về sử dụng các lệnh nào cho các trường hợp cụ thể khác nhau.

Một số lệnh Linux cơ bản thường dùng

- cat [file]
  - In thông tin của tệp ra màn hình
- less [file]
  - Giống "cat" nhưng dành cho tài liệu dài
- mv [file1] [file2]
  - di chuyển file1 sang file2, xóa file1 và ghi đè lên file2 nếu nó tồn tại
- cp [file1] [file2]
  - Sao chép file1 sang file2, ghi đè lên file2 nếu nó đã tồn tại
- rm [file]
  - Xóa file
- nano / vim / emacs
  - Trình soạn thảo văn bản dòng lệnh

Một số lệnh Linux cơ bản thường dùng

- cat [file]
  - In thông tin của tệp ra màn hình
- less [file]
  - Giống "cat" nhưng dành cho tài liệu dài
- mv [file1] [file2]
  - di chuyển file1 sang file2, xóa file1 và ghi đè lên file2 nếu nó tồn tại
- cp [file1] [file2]
  - Sao chép file1 sang file2, ghi đè lên file2 nếu nó đã tồn tại
- rm [file]
  - Xóa file
- nano / vim / emacs
  - Trình soạn thảo văn bản dòng lệnh

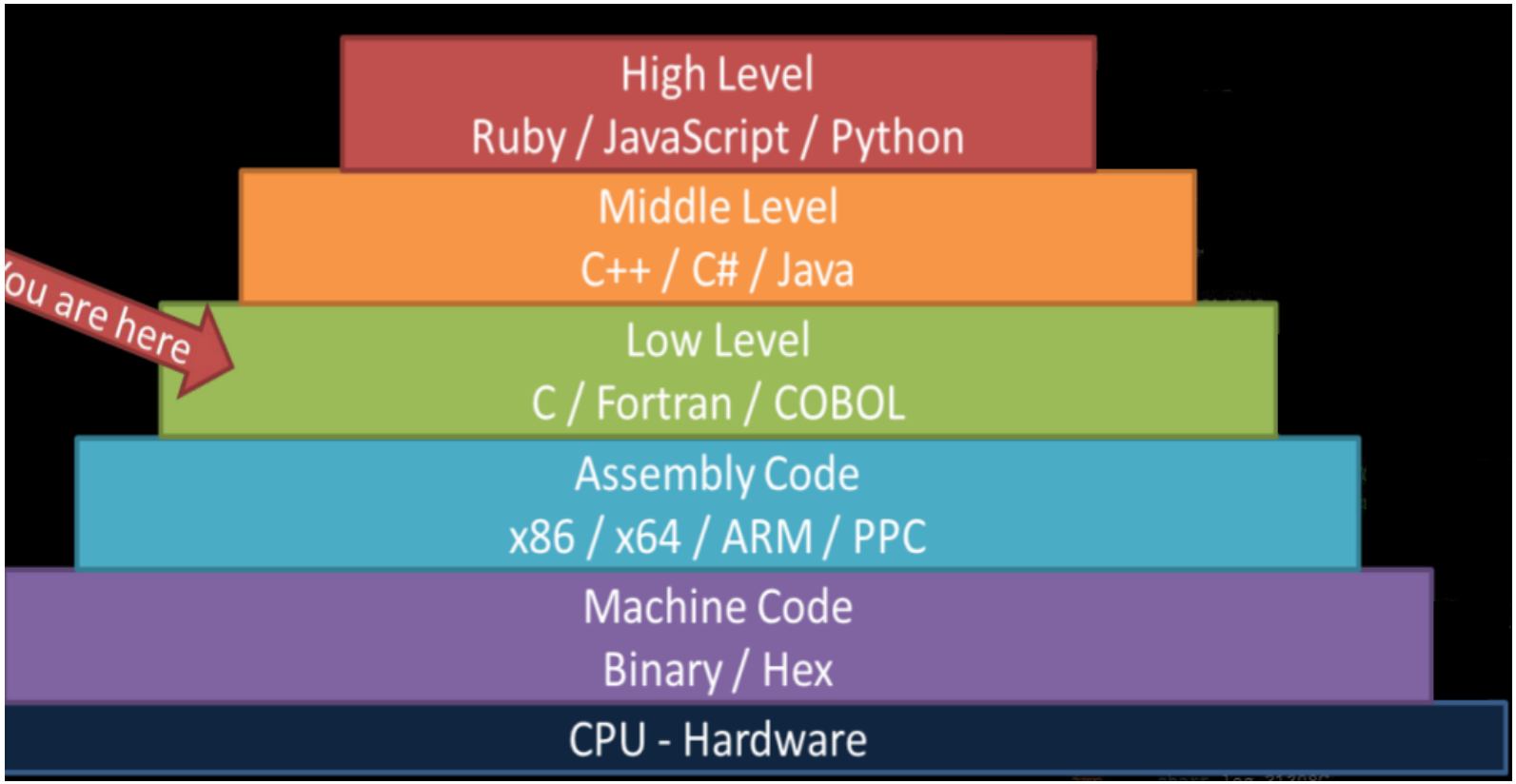
Một số lệnh Linux cơ bản thường dùng

- Dấu "|" (pipes)
  - nối các lệnh (command) với nhau để tạo thành một chuỗi các lệnh thực hiện liên tiếp và truyền dữ liệu từ lệnh này đến lệnh khác một cách liên tục
  - VD: grep "hello" file.txt | cat

Ngôn ngữ C

- Được thiết kế vào năm 1969-1972 để viết hệ điều hành UNIX
- Ngôn ngữ lập trình hướng thủ tục cho các hệ thống
- Ngôn ngữ được biên dịch rất nhanh
- Kiểm soát chặt chẽ tốt bộ nhớ và máy tính
- So với các ngôn ngữ hiện đại, C được coi là ngôn ngữ 'cấp thấp'

Ngôn ngữ C


High Level
Ruby / JavaScript / Python
Middle Level
C++ / C# / Java
ou are here
Low Level
C / Fortran / COBOL
Assembly Code
x86 / x64 / ARM / PPC
Machine Code
Binary / Hex
CPU - Hardware

Hello World! - C Source

```
#include <stdio.h>

int main(int argc, char * argv[])

{

    printf("Hello  World!\n");

    return 0;

}
```

Hello World! - Compiling/Running

```
$ gcc helloworld.c -o helloworld

$ ./helloworld

Hello World!
```

Basic Memory Manipulation

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

int main(int argc, char * argv[])
{
    int i = 0;
    char * message = "hello world";
    char * buffer = (char *) malloc(7);

    if (buffer == NULL)
        return 1;

    strncpy(buffer, message, 5);
    buffer[5] = '\n';
    buffer[6] = '\0';

    for (i = 0; i < 10; i++)
        printf("%s", buffer);

    free(buffer);
    return 0;
}
```

Ví dụ 1

```
$ gcc basic.c -o basic -std=gnu99
$ ./basic
hello
hello
hello
hello
hello
...
```

Ví dụ 2

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

int main(int argc, char *argv[]) {
    char buffer[10] = {0};

    printf("What's your name?\n");
    read(STDIN_FILENO, buffer, 10);
    printf("Hello %s\n", buffer);
    return 0;
}
```

Ví dụ 2

```
$ gcc name.c -o name
$ ./name
What's your name?
ALEX 1234 ABCD
Hello ALEX 1234 ??
```

Ví dụ 3

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

int main(int argc, char *argv[]) {
    char buffer[10] = {0};

    printf("What's your name?\n");
    read(STDIN_FILENO, buffer, 100);
    printf("Hello %s\n", buffer);
    return 0;
}
```

Ví dụ 3

```
$ gcc name2.c -o name2
$ ./name2
What's your name?
ALEX 1234 ABCD EFGH IJKL
Hello ALEX 1234 ABCD EFGH IJKL
????????????
```

x86 Assembly

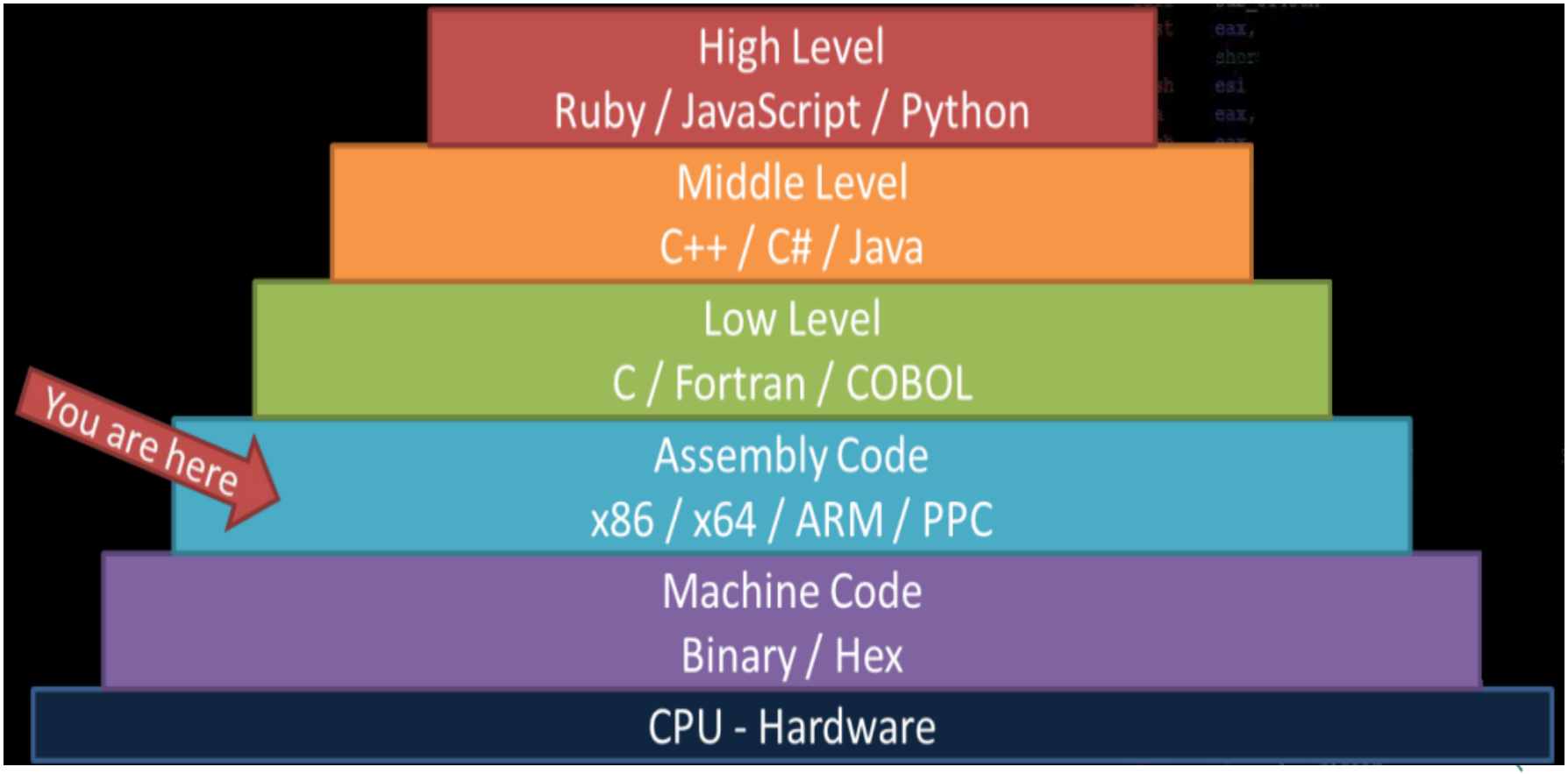
- Một bộ ngôn ngữ lập trình hợp ngữ (assembly language) được giới thiệu bởi Intel từ năm 1978:
  - 1978 - 16bit
  - 1985 - 32bit
  - 2001 - 64bit (Itanium)
  - 2003 - 64bit (AMD64)
- cho phép lập trình viên có thể tương tác trực tiếp với phần cứng và cung cấp cho họ mức độ kiểm soát cao hơn so với các ngôn ngữ lập trình khác
- Các chương trình được viết bằng x86 Assembly có thể được biên dịch thành các file thực thi (executable files) hoặc các thư viện động (dynamic libraries) để được chạy trên hệ thống x86.

```
mov     esi, 1
jmp     loc_8056CB8

                                ; CODE XRE
cmp     [esp+4BCh+var_494], 45h ;
jz      loc_8056CA0
mov     eax, [esp+4BCh+var_478]
mov     [esp+4BCh+var_470], 1
mov     esi, [eax+18h]
jmp     loc_8056DD0

                                ; CODE XRE
cmp     [esp+4BCh+var_494], 45h ;
jz      loc_8056CA0
mov     esi, [esp+4BCh+var_478]
mov     ebx, 92492493h
mov     eax, [esi+18h]
lea     ecx, [eax+6]
mov     [esp+4BCh+var_48C], eax
mov     eax, ecx
imul    ebx
lea     eax, [edx+ecx]
mov     edx, ecx
sar     edx, 1Fh
sar     eax, 2
```

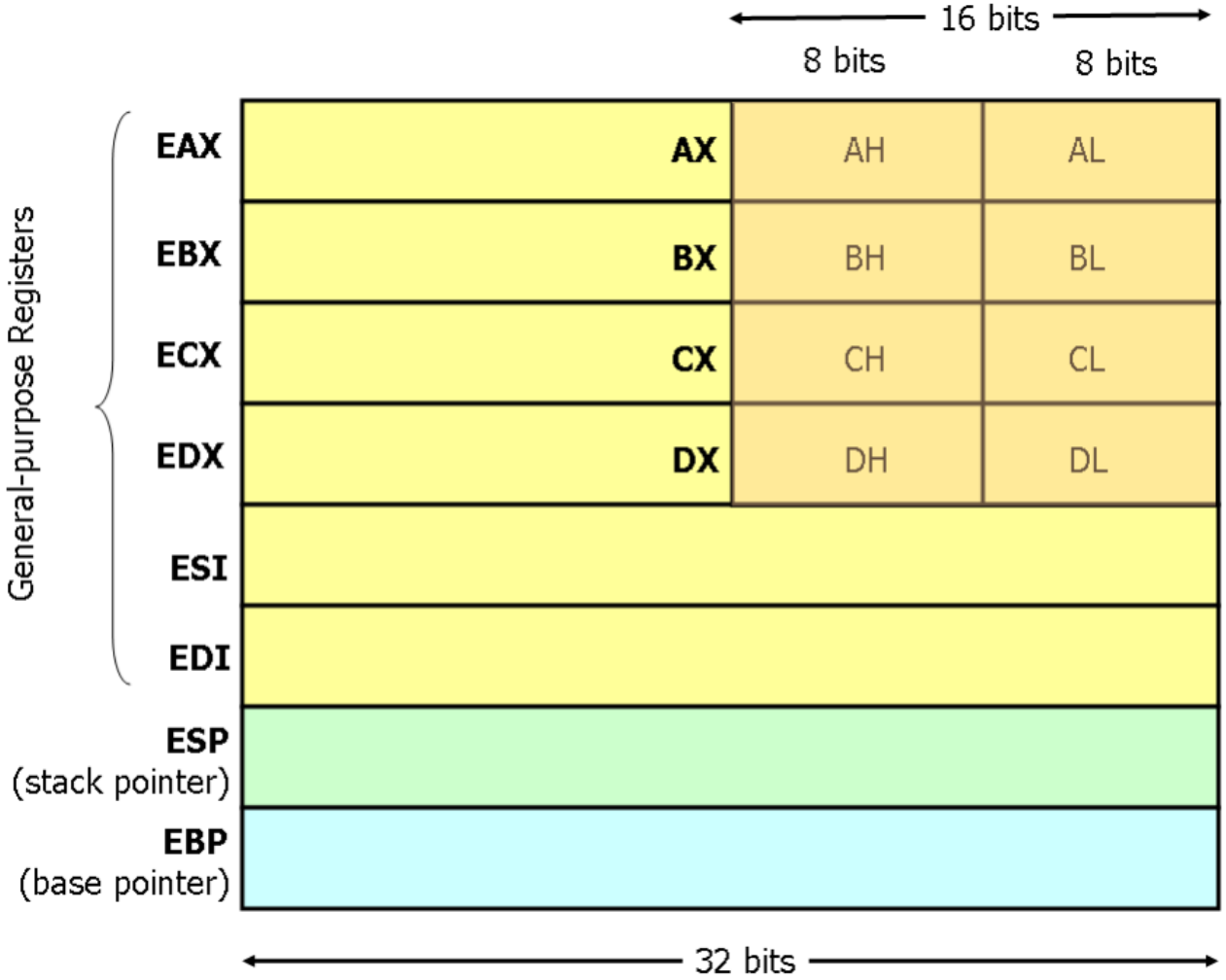
x86 Assembly



x86 Assembly Syntax

- Tất cả các hợp ngữ đều được tạo thành từ các tập lệnh
- Các hướng dẫn nói chung là các phép tính số học đơn giản lấy các thanh ghi hoặc các giá trị không đổi làm đối số
- Còn được gọi là Operands, OpCode, Op(s), mnemonics
- Intel syntax: operand destination, source
  - `mov eax, 5`
- AT&T syntax: operand source, destination
  - `mov $5, eax`

x86 Register Diagram



Các thanh ghi quan trọng

- EAX, EBX, ECX, EDX – thanh ghi đa năng
- ESP – Con trỏ ở đầu của khung ngăn xếp (bộ nhớ thấp hơn)
- EBP – con trỏ cơ sở nằm ở đáy của khung ngăn xếp (bộ nhớ cao hơn)
- EIP - Con trỏ lệnh, con trỏ tới lệnh tiếp theo sẽ được CPU thực hiện
- EFLAGS - lưu trữ các bit cờ
  - ZF = zero flag;
  - IF = Interrupt enable flag;
  - SF = sign flag

Di chuyển dữ liệu

- mov ebx, eax
  - Chuyển dữ liệu từ eax sang ebx
- mov eax, 0xDEADBEEF
  - Chuyển giá trị 0xDEADBEEF vào eax
- mov edx, DWORD PTR [0x41424344]
  - Chuyển giá trị 4 byte ở địa chỉ 0x41424344 vào edx
- mov ecx, DWORD PTR [edx]
  - Chuyển giá trị 4 byte tại địa chỉ trong edx, sang ecx
- mov eax, DWORD PTR [ecx+esi*8]
  - Chuyển giá trị tại địa chỉ trong ecx+esi*8 sang ecx

Các phép tính toán học

- `sub edx, 0x11`
  - `edx = edx - 0x11;`
- `add eax, ebx`
  - `eax = eax + ebx;`
- `inc edx`
  - `edx++;`
- `dec ebx`
  - `ebx--;`
- `xor eax, eax`
  - `xdx = eax^eax;`
- `or eax, 0x1337`
  - `edx = eax | 0x1337;`

Một số bước nhảy có điều kiện

- jz $LOC
  - Nhảy tới $LOC nếu ZF = 1
- jnz $LOC
  - Nhảy tới $LOC nếu ZF = 0
- jg $LOC
  - Nhảy tới $LOC nếu kết quả so sánh là đích lớn hơn nguồn

Thao tác với ngăn xếp

- `push ebx`
  - Trừ 4 từ con trỏ ngăn xếp để di chuyển nó tới bộ nhớ thấp hơn (không,) và sao chép giá trị trong EBX trên đầu ngăn xếp

```
sub esp, 4
mov DWORD PTR [esp], ebx
```

- `pop ebx`
  - Sao chép giá trị ra khỏi đầu ngăn xếp và vào EBX, thêm 4 vào con trỏ ngăn xếp để di chuyển nó tới bộ nhớ cao hơn (0xFFFFFFFF)

```
mov ebx, DWORD PTR [esp]
add esp, 4
```

Calling / Returning

- `call some_function`
  - Gọi mã tại some_function. Chúng ta cần đẩy địa chỉ trả về vào ngăn xếp, sau đó rẽ nhánh tới some_function

  push eip

  `mov eip, some_function ; not actually valid`
- `ret`
  - Được sử dụng để trở về từ một lời gọi hàm. Bật đỉnh của ngăn xếp thành eip

  `pop eip ; not actually valid`
- `nop`
  - "no operation" – không làm gì cả

Basic x86

```
0x08048624: "YOLOSWAG\0"
    mov ebx, 0x08048624
    mov eax, 0
LOOPY:
    mov cl, BYTE PTR [ebx]
    cmp cl, 0
    jz  end
    inc eax
    inc ebx
    jmp LOOPY
end:
    ret
```

Basic x86

```
0x08048624: "YOLOSWAG\0" ; 9 bytes of string data
   mov ebx, 0x08048624   ; char *ebx = "YOLOSWAG\0";
   mov eax, 0            ; set eax to 0
LOOPY:                   ; label, top of loop
   mov cl, BYTE PTR [ebx]; char cl =*ebx
   cmp cl, 0             ; is cl0? (eg "\0")
   jz end                ; if cl was 0, go to end
   inc eax               ; eax++; (counter for length)
   inc ebx               ; ebx++; ([ebx] = "y", "0" … "\0")
   jmp LOOPY             ; go to LOOPY
end:                     ; label, end of loop/function
   ret                   ; return (len of str in eax)
```

Human Decompiler - x86 →C

```
0x08048624: "YOLOSWAG\0"
    mov ebx, 0x08048624
    mov eax, 0
LOOPY:
    mov cl, BYTE PTR [ebx]
    cmp cl, 0
    jz end
    inc eax
    inc ebx
    jmp LOOPY
end:
    ret
```

```
…
char *word = "YOLOSWAG";
int len=0;

while (*word !=0)
{
        len++;
        word++;
}


return len;
```

Tài liệu tham khảo

- Jon Erickson. Hacking: The Art of Exploitation

  Chapter 0x200: Programming - C programming and GDB

- Bruce Dang, et al. Practical reverse engineering: x86, x64, ARM, Windows Kernel, reversing tools, and obfuscation

- Trang web luyện tập linux

  http://overthewire.org/wargames/bandit/