

Integer Overflow

Nhóm 01: Acceleration

$$C = \frac{B^3 + C^2 + A}{3BA}$$

$$\frac{10+17}{3.45}$$

$$\left(\frac{C-B}{3-D} \right) = \left(\frac{A}{3B} \right) = \frac{3C(2)^4}{X+Y+C}$$

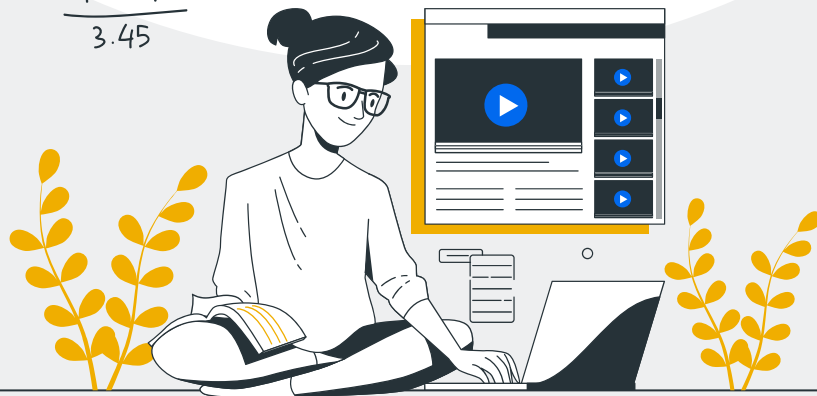


Table of contents

01

Tổng quan

02

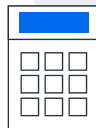
Kiến thức nền tảng

03

Khai thác và biện pháp khắc phục

04

Demo



$$\frac{3c(2)^4}{x+y+c}$$

$$\frac{\sqrt{2.8}}{3+2^+}$$



01

Tổng quan

Tổng quan

$$\frac{10+17}{3.45}$$

○

Theo **Wikipedia**: Trong lập trình máy tính, Integer Overflow 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.

◇

○

$$\frac{4+6+(2\sqrt{3})}{\sqrt{276}}$$

Tổng quan

$$\frac{10+17}{3.45}$$

○

⇒ Integer Overflow hiểu đơn giản là lỗi hỏng 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*

$$\frac{4+6+(2\sqrt{3})}{\sqrt{276}}$$



$$\frac{4+6+(2\sqrt{3})}{\sqrt{276}}$$

Kiến thức nền tảng

Tại sao lại xảy ra lỗi
Integer Overflow?



$$\frac{\sqrt{2.8}}{3+2^+}$$


$$\frac{10+17}{3.45}$$

A donut chart with a yellow background and a white center. A yellow segment represents 75% of the total, while the remaining 25% is white. The chart is set against a light gray background.

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

How computer stored an integer number?

Sign-Magnitude



1111 1111	(-127)
1111 1110	(-126)
.....	...
.....	...
1000 0001	(-1)
1000 0000	(-0)
0111 1111	(+127)
0111 1110	(+126)
.....	...
.....	...
0000 0001	(+1)
0000 0000	(+0)

Increasing
binary
codes

1's complement

0111 1111	(+127)
0111 1110	(+126)
.....	...
.....	...
0000 0001	(+1)
0000 0000	(+0)

1111 1111	(-0)
1111 1110	(-1)
.....	...
.....	...
1000 0001	(-126)
1000 0000	(-127)

wrap
around

Negative number

2's complement

0111 1111	(+127)
0111 1110	(+126)
.....	...
.....	...
0000 0001	(+1)
0000 0000	(+0)

1111 1111	(-1)
1111 1110	(-2)
.....	...
.....	...
1000 0001	(-127)
1000 0000	(-128)

+1



Các kiểu dữ liệu trong C/C++

$$\frac{10+17}{3.45}$$

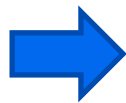
Data type	Size(bytes)	Range	Format string
Char	1	128 to 127	%c
Unsigned char	1	0 to 255	%c
Short or int	2	-32,768 to 32,767	%i or %d
Unsigned int	2	0 to 65535	%u
Long	4	-2147483648 to 2147483647	%ld
Unsigned long	4	0 to 4294967295	%lu
Float	4	3.4 e-38 to 3.4 e+38	%f or %g
Double	8	1.7 e-308 to 1.7 e+308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

Một số ví dụ về integer overflow

test.cpp

C: > Document > LapTrinhAnToanVaKhairThacLoHongPhanMem > Semina

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main() {
6      int a = 4278190080;
7      cout << "a = " << a << '\n';
8      unsigned int b = 4299360564;
9      cout << "b = " << b << '\n';
10     cout << "a - b = " << a - b << '\n';
11     int c = 1;
12     unsigned int d = 4294967295;
13     cout << "c - d = " << c - d << '\n';
14     return 0;
15 }
16
```



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

$$\frac{4+6+(2\sqrt{3})}{\sqrt{276}}$$

Sai số trong lưu trữ số thực

How Float is stored in Computer Memory

float num = 10.75; 1010.11

1. significant bit * 2^{exponent}

$$\begin{aligned} 1.01011 * 2^3 & \quad \text{bias} = 2^{n-1} - 1 \\ & = 2^{8-1} - 1 \\ & = 127 \end{aligned}$$

Normalized exponent = 127 + 3 = 130 -> 10000010

$$10.75 = 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0 + 1 * 2^{-1} + 1 * 2^{-2}$$

Binary Equivalent:

0 1 0 0 0 0 0 1 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

```
#include <stdio.h>
int main()
{
    int num1 = 10.75;
    printf("%f", num1);
    return 0;
}
```

Output: 10.75

0 0

32 bits



Memory:

0 1 0 0 0 0 0 1 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Sai số trong lưu trữ số thực

<https://www.h-schmidt.net/FloatConverter/IEEE754.html>

Tools & Thoughts

IEEE-754 Floating Point Converter

Translations: [de](#)

This page allows you to convert between the

IEEE 754 floating point).

Value:
Encoded as:
Binary:

Sign

+

0

1

2

3

4

5

6

7

8

9

A

B

C

D

```
>>> 0.1 + 0.1 + 0.1
0.30000000000000004
>>> (0.1 + 0.1 + 0.1 == 0.3)
False
>>> _
```

Error due to conversion:

1.490116119384765625E-9

Binary Representation

00111101110011001100110011001101

Hexadecimal Representation

0x3dcccccd

☒ ☐ ☐ ☒ ☒ ☐ ☒

+1

-1

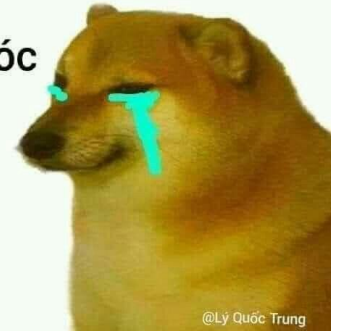
$$\frac{4+6+(2\sqrt{3})}{\sqrt{276}}$$

$$\frac{A}{3B}$$



Khai thác lỗ
hổng này như
thế nào?

Khmóc



$$\frac{5 \pm \sqrt{3-4}}{2}$$

@Ly Quốc Trung

Khai thác lỗ hổng này như thế nào?

Integer
Overflow

```
graph TD; A(Integer Overflow) --> B(Làm sai lệch các phép tính nhằm thay đổi luồng chạy chương trình); A --> C(Lợi dụng để ghi đè lên các vị trí bên ngoài mảng);
```

Làm sai lệch các phép tính
nhằm thay đổi luồng chạy
chương trình

Lợi dụng để ghi đè lên các vị
trí bên ngoài mảng

$$\frac{10+17}{3.45}$$



Làm sao để khắc phục



Sử dụng python



Sử dụng một kiểu dữ liệu đủ lớn



Kiểm tra phép toán có kiểu dữ liệu signed và unsigned



Chuẩn bị sẵn cho một khoảng sai số cho phép khi tính toán số thực

$$\frac{3 \sin 4/8}{\sqrt{3 \cdot 2 \cdot 4 + 2}}$$



$$\frac{\sqrt{2.8}}{3+2^+}$$



Demo

$$\frac{\sqrt{z}}{(\frac{1}{z})^2}$$

Demo1: <https://github.com/Cobra-de1/NT521.M11.ANTN/tree/main/Seminar/demo1>

Demo2: <https://github.com/Cobra-de1/NT521.M11.ANTN/tree/main/Seminar/demo2>



Demo2

What is GOT and PLT?

C test.c

home > kali > Desktop > C test.c

```
1  #include <stdio.h>
2
3  int main() {
4      char name[20];
5      printf("Input your name: ");
6      scanf("%19s", name);
7      printf("Hello %s", name);
8      return 0;
9  }
10 |
```

Dynamic linking



pwndbg> info functions
All defined functions:

Non-debugging symbols:

0x0000000000401000	_init
0x0000000000401030	printf@plt
0x0000000000401040	__isoc99_scanf@plt
0x0000000000401050	_start
0x0000000000401080	_dl_relocate_static_pie
0x0000000000401090	deregister_tm_clones
0x00000000004010c0	register_tm_clones
0x0000000000401100	__do_global_ctors_aux
0x0000000000401130	frame_dummy
0x0000000000401132	main
0x0000000000401190	__libc_csu_init
0x00000000004011f0	__libc_csu_fini
0x00000000004011f4	_fini



Demo2

What is GOT and PLT?

```
pwndbg> disassemble 0x0000000000401030
```

```
Dump of assembler code for function printf@plt:
```

```
0x0000000000401030 <+0>:    jmp     QWORD PTR [rip+0x2fe2]        # 0x404018 <printf@got.plt>
0x0000000000401036 <+6>:    push   0x0
0x000000000040103b <+11>:   jmp     0x401020
```

```
End of assembler dump.
```

```
pwndbg> disassemble 0x0000000000401040
```

```
Dump of assembler code for function __isoc99_scanf@plt:
```

```
0x0000000000401040 <+0>:    jmp     QWORD PTR [rip+0x2fda]        # 0x404020 <__isoc99_scanf@got.plt>
0x0000000000401046 <+6>:    push   0x1
0x000000000040104b <+11>:   jmp     0x401020
```

```
End of assembler dump.
```



Demo2

What is GOT and PLT?

```
pwndbg> x/xg 0x404018
0x404018 <printf@got.plt>: 0x00007ffff7e42cf0
pwndbg> x/xg 0x404020
0x404020 <__isoc99_scanf@got.plt>: 0x00007ffff7e440e0
pwndbg> x printf
0x7ffff7e42cf0 <__printf>: 0x49000000d8ec8148
pwndbg> x scanf
0x7ffff7e431b0 <__scanf>: 0x49000000d8ec8148
```

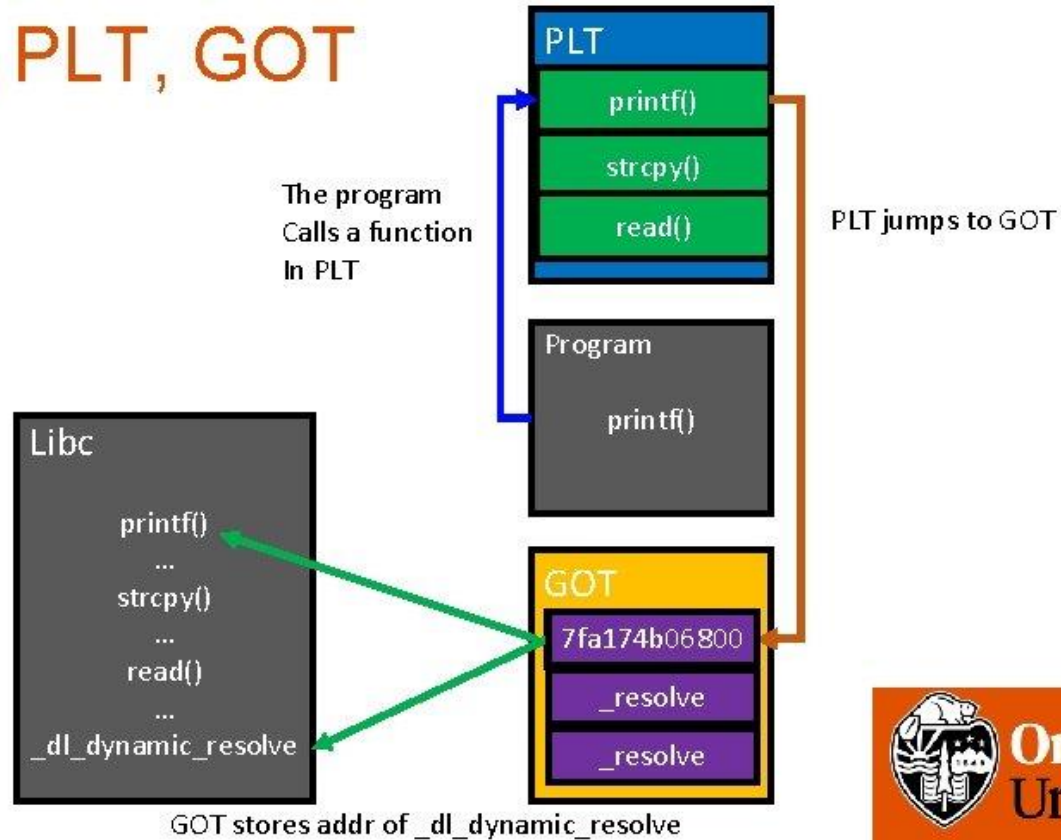


0x7ffff7dec000	0x7ffff7e11000	r--p	25000	0	/usr/lib/x86_64-linux-gnu/libc-2.31.so
0x7ffff7e11000	0x7ffff7f5c000	r-xp	14b000	25000	/usr/lib/x86_64-linux-gnu/libc-2.31.so
0x7ffff7f5c000	0x7ffff7fa6000	r--p	4a000	170000	/usr/lib/x86_64-linux-gnu/libc-2.31.so
0x7ffff7fa6000	0x7ffff7fa7000	---p	1000	1ba000	/usr/lib/x86_64-linux-gnu/libc-2.31.so
0x7ffff7fa7000	0x7ffff7faa000	r--p	3000	1ba000	/usr/lib/x86_64-linux-gnu/libc-2.31.so
0x7ffff7faa000	0x7ffff7fad000	rw-p	3000	1bd000	/usr/lib/x86_64-linux-gnu/libc-2.31.so

Demo2

What is GOT and PLT?

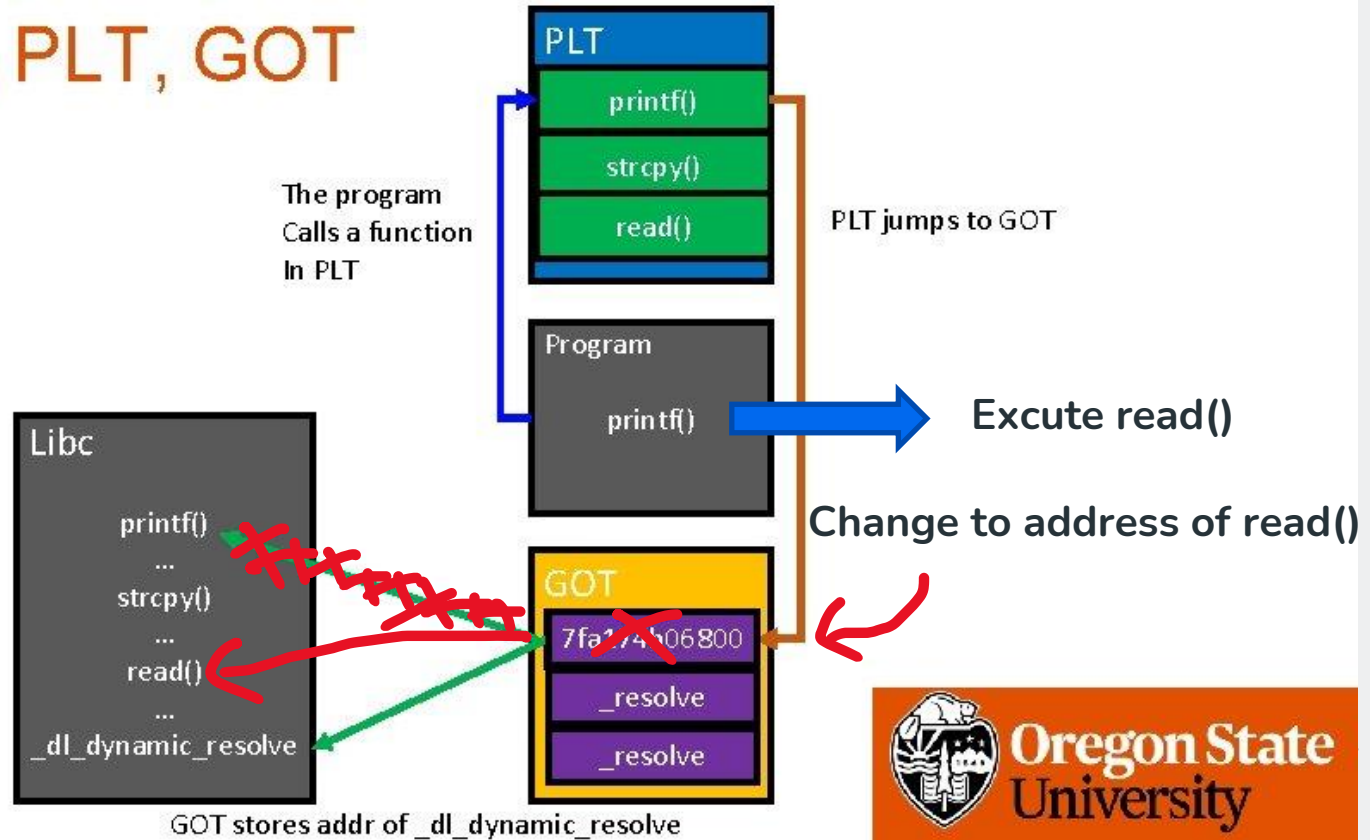
ELF, PLT, GOT



Demo2

Attack overwrite GOT

ELF, PLT, GOT



$$\frac{C - B}{3 - D}$$



$$\frac{\sqrt{z}}{(\frac{1}{z})^2}$$

Acceleration



Thanks

