

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG



BÁO CÁO BÀI THỰC HÀNH

Kiểm thử xâm nhập

Quét mạng và dịch vụ nâng cao

Giảng viên: Đinh Trường Duy

Nhóm lớp: 01

Sinh viên: Hoàng Trung Kiên

Mã sinh viên: B20DCAT098

Hà Nội – 2024

Mục lục

| | |
|------------------------------------|----|
| 1. Mục đích. | 3 |
| 2. Yêu cầu đối với sinh viên. | 3 |
| 3. Thực hành. | 3 |
| 4. Checkwork..... | 10 |

1. Mục đích.

Bài thực hành giới thiệu hàm printf và giúp sinh viên tìm hiểu cách thức mà hàm tham chiếu địa chỉ bộ nhớ để đáp ứng với đặc tả định dạng đã cho của nó. Ngoài ra, bài thực hành còn cung cấp phần giới thiệu về các kỹ thuật được sử dụng trong các bài thực hành printf nâng cao hơn (formatstring và format64).

2. Yêu cầu đối với sinh viên.

- Sinh viên sử dụng được ngôn ngữ lập trình C, Assembly cơ bản
- Sinh viên biết và sử dụng được chương trình gdb

3. Thực hành.

Mở terminal, trong thư mục labtainer-student, bắt đầu bài thực hành bằng lệnh: Labtainer -r printf

```
student@ubuntu:~/labtainer/labtainer-student$ labtainer -r printf
non-network local connections being added to access control list

Please enter your e-mail address: [B20DCAT098]
Started 1 containers, 1 completed initialization. Done.

C Library printf introduction -- Read this first

The lab manual for this lab is at:
  file:///home/student/labtainer/trunk/labs/printf/docs/printf.pdf

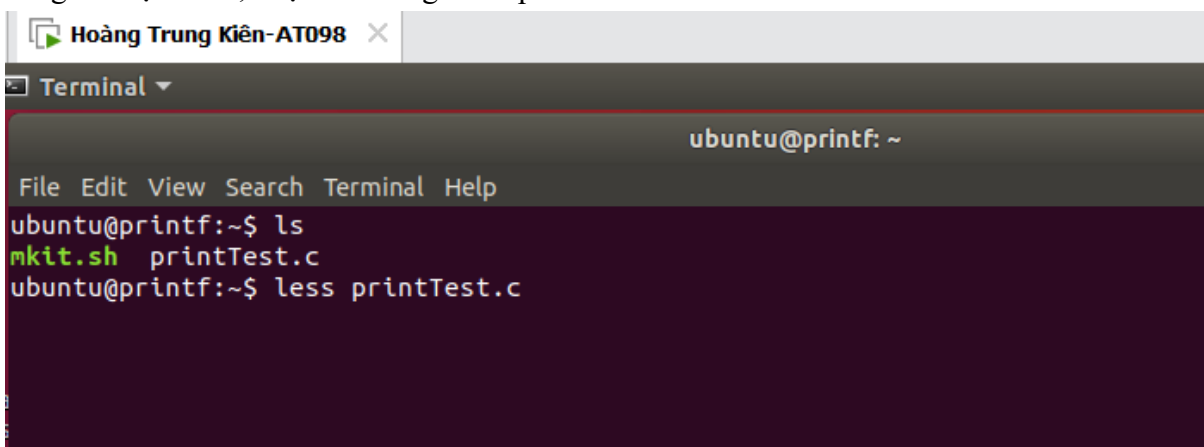
Review the entire lab manual before proceeding with the lab.

Press <enter> to start the lab

student@ubuntu:~/labtainer/labtainer-student$
```

Đánh giá chương trình printTest.c

Khi bắt đầu lab, một cửa sổ terminal sẽ mở ra. Tại đó, xem chương trình printTest.c bằng vi hoặc nano, hoặc chỉ cần gõ less printTest.c.



```
Hoàng Trung Kiên-AT098 x
Terminal v
ubuntu@printf: ~
File Edit View Search Terminal Help
ubuntu@printf:~$ ls
mkit.sh printTest.c
ubuntu@printf:~$ less printTest.c
```

```
ubuntu@printf: ~  
File Edit View Search Terminal Help  
  
#include<stdio.h>  
#include<stdlib.h>  
  
int main(int argc, char *argv[])  
{  
    char user_input[100];  
    int var1 = 13;  
    int var2 = 21;  
    char *str = "my dog has fleas";  
    printf("var1 is: %d \n", var1);  
    printf("var2 is: 0x%x and str is : %s\n", var2, str);  
    printf("Enter a string:\n");  
    scanf("%s", user_input);  
    printf(user_input);  
    printf("\n");  
}  
printfTest.c (END)
```

Quan sát cú pháp của câu lệnh printf đầu tiên. Tham số đầu tiên là một chuỗi định dạng chứa văn bản chính nó muốn hiển thị, và một hoặc nhiều phần định dạng quyết định cách mà các tham số còn lại được hiển thị. Phần định dạng bắt đầu bằng ký hiệu %. Trong câu lệnh printf đầu tiên, phần định dạng là %d, nghĩa là chỉ định printf hiển thị tham số dưới dạng số nguyên. Do đó, giá trị của var1 sẽ được hiển thị dưới dạng số nguyên theo sau chuỗi "var1 is: ". lưu ý: \n là ký tự xuống dòng.

Câu lệnh printf thứ hai minh họa cách chúng ta có thể hiển thị các giá trị của nhiều tham số. Trong trường hợp này, đó là biểu diễn thập lục phân của một số nguyên (sử dụng %x) theo sau bởi một chuỗi (sử dụng phần định dạng %s).

Hàm printf có một tập hợp các phần định dạng rất phong phú, nhưng hầu hết chúng không quan trọng cho bài thực hành này. Điều quan trọng trong bài thực hành này là cách mà printf tham chiếu đến bộ nhớ để tìm các giá trị cần được hiển thị.

Câu lệnh printf thứ ba dễ bị tấn công, như chúng ta sẽ thấy trong bài thực hành này.

Chạy chương trình PrintTest

Tập lệnh mkit.sh sẽ biên dịch chương trình dưới dạng tệp thực thi 32 bit: `./mkit.sh`

```
ubuntu@printf: ~  
File Edit View Search Terminal Help  
ubuntu@printf:~$ ls  
mkit.sh printTest.c  
ubuntu@printf:~$ less printTest.c  
ubuntu@printf:~$ ./mkit.sh  
printTest.c: In function 'main':  
printTest.c:14:12: warning: format not a string literal and no format arguments [-Wformat-security]  
    14 |     printf(user_input);  
       |     ^~~~~~  
ubuntu@printf:~$
```

- Sau đó, ta có thể chạy chương trình: `./printTest`

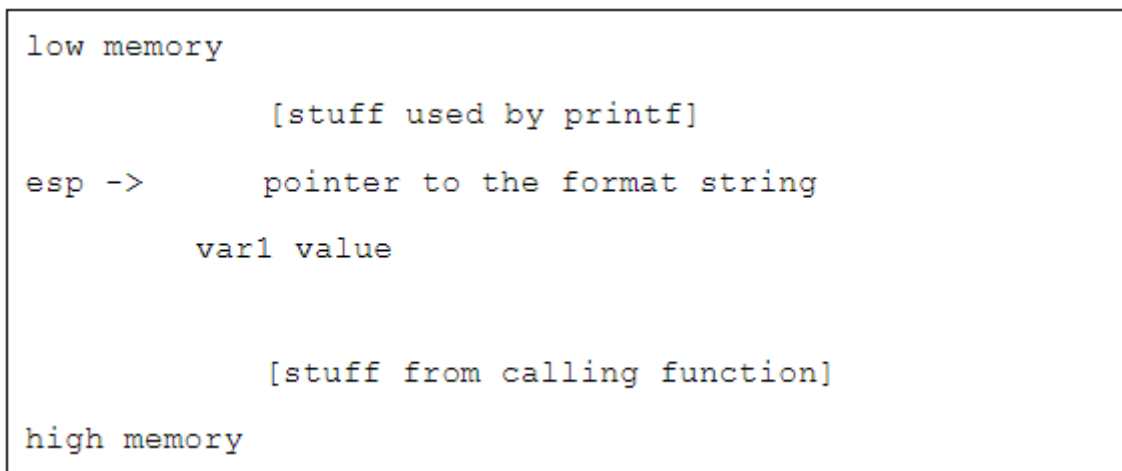
```

ubuntu@printf:~$ ./printTest
var1 is: 13
var2 is: 0x15 and str is : my dog has fleas
Enter a string:
Kien AT098
Kien AT098
Kien
ubuntu@printf:~$ █

```

Quy ước gọi hàm

Khi một chương trình 32-bit x86 chuẩn bị gọi một hàm, các tham số của hàm sẽ được đẩy vào ngăn xếp trước. Hàm sẽ được gọi và tham chiếu đến các tham số của nó từ ngăn xếp. Trong lệnh printf đầu tiên, có hai tham số: chuỗi định dạng (format string) và biến var1. Vì trong 32-bit x86, thanh ghi con trỏ ngăn xếp esp giảm khi ngăn xếp "to ra", sơ đồ hình 1 có bộ nhớ thấp ở đầu của biểu đồ.



Hình: định dạng của ngăn xếp trước khi gọi printf

Trong hình, có thể thấy giá trị của biến var1 được đẩy vào trong ngăn xếp tiếp theo là con trỏ đến chuỗi định dạng.

Điều này là do trong hàm printf, chuỗi định dạng cần được truyền dưới dạng con trỏ đến một chuỗi, trong khi giá trị của biến var1 cần được truyền dưới dạng một tham số riêng biệt. Vì vậy, con trỏ đến chuỗi định dạng được đẩy vào ngăn xếp sau giá trị của biến var1.

Hành vi của hàm printf

Khi hàm printf được gọi, nó sẽ tìm con trỏ tới chuỗi định dạng ở đầu tham số trong ngăn xếp. Sau đó, nó đọc chuỗi định dạng và giải thích các đặc tả chuyển đổi. Trong trường hợp của lệnh printf đầu tiên, nó chỉ nhìn thấy %d, điều này khiến printf xử lý tham số tiếp theo trên ngăn xếp là một số nguyên và hiển thị giá trị của nó cùng với các chuỗi ký tự định dạng khác.

Lần gọi hàm printf thứ hai sẽ có ba tham số. Lần này, hàm printf thấy một đặc tả chuyển đổi %x và nhìn vào tham số tiếp theo, hiện tại là giá trị của biến var2 và hiển thị giá trị đó dưới dạng một giá trị thập lục phân theo định dạng %x. Sau đó, nó thấy %s và xử lý tham số tiếp theo như là một con trỏ đến một chuỗi, sau đó hiển thị chuỗi đó.

Quan sát các quy ước gọi hàm của chương trình GDB

Chạy chương trình trong gdb: gdb printTest

```
ubuntu@printf:~$ gdb printTest
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
  <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from printTest...
(gdb)
```

Liệt kê chương trình bằng lệnh: list

```
(gdb) list
1      #include<stdio.h>
2      #include<stdlib.h>
3
4      int main(int argc, char *argv[])
5      {
6          char user_input[100];
7          int var1 = 13;
8          int var2 = 21;
9          char *str = "my dog has fleas";
10         printf("var1 is: %d \n", var1);
(gdb) list
11         printf("var2 is: 0x%x and str is : %s\n", var2, str);
12         printf("Enter a string:\n");
13         scanf("%s", user_input);
14         printf(user_input);
15         printf("\n");
16     }
(gdb) █
```

đặt một breakpoint tại dòng chứa lệnh printf đầu tiên và chạy chương trình

break <số dòng>

run

```
(gdb) break 10
Breakpoint 1 at 0x129e: file printTest.c, line 10.
(gdb) run
Starting program: /home/ubuntu/printTest

Breakpoint 1, main (argc=1, argv=0xffffd654) at printTest.c:10
10         printf("var1 is: %d \n", var1);
(gdb) █

Please enter your e-mail address: [B20DCAT098]
Started 1 containers, 1 completed initialization. Done.
```

Chương trình sẽ ngắt ngay trước lệnh gọi printf. Tuy nhiên, như vậy là chưa đủ gần đến mục đích của bài thực hành, vì vậy cần xem mã hợp ngữ của các lệnh máy để có thể thực thi đến trước khi lệnh gọi hàm thực sự được thực hiện. Sử dụng lệnh GDB sau để hiển thị mã hợp ngữ của lệnh hiện tại: display/i \$pc

```
(gdb) display/i $pc
1: x/i $pc
=> 0x5655629e <main+81>:      sub    $0x8,%esp
(gdb) █

Please enter your e-mail address: [B20DCAT098]
Started 1 containers, 1 completed initialization. Done.
```

Sau đó, sử dụng lệnh nexti để thực thi lệnh tiếp theo. Nhấn liên tục phím Return (Enter) để tiếp tục các bước cho đến khi chạm đến lệnh gọi printf@plt

```
(gdb) nexti
0x565562a1      10      printf("var1 is: %d \n", var1);
1: x/i $pc
=> 0x565562a1 <main+84>:      pushl   -0x7c(%ebp)
(gdb)
0x565562a4      10      printf("var1 is: %d \n", var1);
1: x/i $pc
=> 0x565562a4 <main+87>:      lea     -0x1faf(%ebx),%eax
(gdb)
0x565562aa      10      printf("var1 is: %d \n", var1);
1: x/i $pc
=> 0x565562aa <main+93>:      push    %eax
(gdb)
0x565562ab      10      printf("var1 is: %d \n", var1);
1: x/i $pc
=> 0x565562ab <main+94>:      call    0x565560b0 <printf@plt>
(gdb)

Please enter your e-mail address: [B20DCAT098]
Started 1 containers, 1 completed initialization. Done.
```

Bây giờ chương trình mới thực sự sắp gọi printf. Để xem hai mươi từ trên ngăn xếp dưới dạng giá trị thập lục phân ta dùng: x/20xw \$esp

```
(gdb) x/20xw $esp
0xfffffd500: 0x56557019 0x0000000d 0x000000c2 0x5655626b
0xfffffd510: 0xfffffd54a 0xf7ffc89c 0xf7ffc8a0 0xfffffd654
0xfffffd520: 0xf7ffd000 0xf7ffc8a0 0xfffffd54a 0x0000000d
0xfffffd530: 0x00000015 0x56557008 0x00000001 0xf7ffc7e0
0xfffffd540: 0x00000000 0x00000000 0x00005034 0x4bb4f300
(gdb) █

Please enter your e-mail address: [B20DCAT098]
Started 1 containers, 1 completed initialization. Done.
```

Thanh ghi esp đang trở tới đỉnh ngăn xếp, chứa tham số đầu tiên của printf, nghĩa là, con trỏ tới chuỗi định dạng. Xác nhận điều đó bằng cách kiểm tra bộ nhớ tại địa chỉ đó (nghĩa là từ được hiển thị đầu tiên) dưới dạng một chuỗi: x/s <địa chỉ>

```
(gdb) x/20xw $esp
0xfffffd500: 0x56557019 0x0000000d 0x000000c2 0x5655626b
0xfffffd510: 0xfffffd54a 0xf7ffc89c 0xf7ffc8a0 0xfffffd654
0xfffffd520: 0xf7ffd000 0xf7ffc8a0 0xfffffd54a 0x0000000d
0xfffffd530: 0x00000015 0x56557008 0x00000001 0xf7ffc7e0
0xfffffd540: 0x00000000 0x00000000 0x00005034 0x4bb4f300
(gdb) x/s 0x56557019
0x56557019: "var1 is: %d \n"
(gdb) █

Please enter your e-mail address: [B20DCAT098]
Started 1 containers, 1 completed initialization. Done.
```

=> chuỗi định dạng. “Word” ở tham số tiếp theo trên ngăn xếp là của giá trị var1 là 13 (hex 0x0d).

Nhìn vào nội dung của các địa chỉ tiếp theo. Sinh viên thấy một số giá trị của địa chỉ, nhưng xa hơn một chút, sinh viên sẽ thấy giá trị của 2 biến var1 và var2 trong các “words” liền kề. Bộ nhớ đó là nơi chương trình printTest đã lưu trữ hai giá trị đó. Trước đó sinh viên đã quan sát thấy một bản sao giá trị của var1 ở gần đầu ngăn xếp. Các giá trị tại các địa chỉ cao hơn là các giá trị ban đầu của các biến đó.

Bước tiếp theo của là đánh lừa printf để hiển thị những giá trị đó từ vị trí gốc của chúng. đặt một breakpoint tại printf thứ hai, thực thi từng lệnh hợp ngữ cho đến lệnh call printf đó và xem xét ngăn xếp để xác định ba tham số của printf.

Quit và gõ lệnh gdb printTest

```
(gdb) list
1      #include<stdio.h>
2      #include<stdlib.h>
3
4      int main(int argc, char *argv[])
5      {
6          char user_input[100];
7          int var1 = 13;
8          int var2 = 21;
9          char *str = "my dog has fleas";
10         printf("var1 is: %d \n", var1);
(gdb) list
11         printf("var2 is: 0x%x and str is : %s\n", var2, str);
12         printf("Enter a string:\n ");
13         scanf("%s", user_input);
14         printf(user_input);
15         printf("\n");
16     }
(gdb) break 11
Breakpoint 1 at 0x12b3: file printTest.c, line 11.
(gdb) run
Starting program: /home/ubuntu/printTest
var1 is: 13

Breakpoint 1, main (argc=1, argv=0xffffd654) at printTest.c:11
11         printf("var2 is: 0x%x and str is : %s\n", var2, str);
(gdb) display/i $pc
1: x/i $pc
=> 0x565562b3 <main+102>:      sub    $0x4,%esp
(gdb) nexti
0x565562b6      11      printf("var2 is: 0x%x and str is : %s\n", var2, str);
1: x/i $pc
=> 0x565562b6 <main+105>:      pushl  -0x74(%ebp)
(gdb)
0x565562b9      11      printf("var2 is: 0x%x and str is : %s\n", var2, str);
1: x/i $pc
=> 0x565562b9 <main+108>:      pushl  -0x78(%ebp)
(gdb)
0x565562bc      11      printf("var2 is: 0x%x and str is : %s\n", var2, str);
1: x/i $pc
=> 0x565562bc <main+111>:      lea    -0x1fa0(%ebx),%eax
(gdb)
0x565562c2      11      printf("var2 is: 0x%x and str is : %s\n", var2, str);
1: x/i $pc
=> 0x565562c2 <main+117>:      push  %eax
(gdb)
0x565562c3      11      printf("var2 is: 0x%x and str is : %s\n", var2, str);
1: x/i $pc
=> 0x565562c3 <main+118>:      call   0x565560b0 <printf@plt>
(gdb)
(gdb) x/20xw $esp
0xfffffd500: 0x56557028 0x00000015 0x56557008 0x5655626b
0xfffffd510: 0xfffffd54a 0xf7ffc89c 0xf7ffc8a0 0xfffffd654
0xfffffd520: 0xf7ffd000 0xf7ffc8a0 0xfffffd54a 0x0000000d
0xfffffd530: 0x00000015 0x56557008 0x00000001 0xf7ffc7e0
0xfffffd540: 0x00000000 0x00000000 0x00005034 0xbf8b4300
(gdb) x/s 0x56557028
0x56557028: "var2 is: 0x%x and str is : %s\n"
(gdb)
Please enter your e-mail address: [B20DCAT098]
Started 1 containers, 1 completed initialization. Done.
```

Người dùng nhập vào định dạng của chuỗi

Xem lại mã nguồn của chương trình testPrint.c và tìm dòng code sau:

```
printf(user_input);
```


Quit và gõ lệnh gdb printTest

```
(gdb) list
1      #include<stdio.h>
2      #include<stdlib.h>
3
4      int main(int argc, char *argv[])
5      {
6          char user_input[100];
7          int var1 = 13;
8          int var2 = 21;
9          char *str = "my dog has fleas";
10         printf("var1 is: %d \n", var1);
(gdb) list
11         printf("var2 is: 0x%x and str is : %s\n", var2, str);
12         printf("Enter a string:\n");
13         scanf("%s", user_input);
14         printf(user_input);
15         printf("\n");
16     }
(gdb) █

Please enter your e-mail address: [B20DCAT098]
Started 1 containers, 1 completed initialization. Done.
```

Chạy chương trình (không có gdb) và cung cấp chuỗi trên làm đầu vào. Các giá trị được hiển thị đến từ đâu? Chạy lại chương trình trong gdb, lần này đặt dấu ngắt ở số dòng của lệnh gọi printf để bị tấn công và sử dụng run để khởi động chương trình.

```
(gdb) break 14
Breakpoint 1 at 0x12f3: file printTest.c, line 14.
(gdb) r
Starting program: /home/ubuntu/printTest
var1 is: 13
var2 is: 0x15 and str is : my dog has fleas
Enter a string:
KienAT098
KienAT098

Breakpoint 1, main (argc=1, argv=0xffffd654) at printTest.c:14
14         printf(user_input);
(gdb)

Please enter your e-mail address: [B20DCAT098]
Started 1 containers, 1 completed initialization. Done.
```

Trước khi chương trình đạt đến điểm dừng, nó sẽ bắt sinh viên nhập chuỗi. Dán chuỗi trên và sau đó chương trình sẽ ngắt tại lệnh gọi (gần như) tới printf. Sử dụng: display/i \$pc

nexti

<return>

```
(gdb) display/i $pc
1: x/i $pc
=> 0x565562f3 <main+166>:      sub    $0xc,%esp
(gdb) nexti
0x565562f6      14      printf(user_input);
1: x/i $pc
=> 0x565562f6 <main+169>:      lea    -0x70(%ebp),%eax
(gdb)
0x565562f9      14      printf(user_input);
1: x/i $pc
=> 0x565562f9 <main+172>:      push   %eax
(gdb)
0x565562fa      14      printf(user_input);
1: x/i $pc
=> 0x565562fa <main+173>:      call   0x565560b0 <printf@plt>
(gdb) █

Please enter your e-mail address: [B20DCAT098]
Started 1 containers, 1 completed initialization. Done.
```

Tới bước gọi hàm `printf@plt` cần hiển thị nội dung của ngăn xếp. `x/20xw $esp`

```
=> 0x565562fa <main+173>:      call   0x565560b0 <printf@plt>
(gdb) x/20xw $esp
0xfffffd500:    0xfffffd538    0xfffffd538    0x56557008    0x5655626b
0xfffffd510:    0xfffffd54a    0xf7ffc89c    0xf7ffc8a0    0xffffd654
0xfffffd520:    0xf7ffd000    0xf7ffc8a0    0xffffd54a    0x0000000d
0xfffffd530:    0x00000015    0x56557008    0x6e65694b    0x39305441
0xfffffd540:    0x00000038    0x00000000    0x00005034    0x7743c900
(gdb) █

Please enter your e-mail address: [B20DCAT098]
Started 1 containers, 1 completed initialization. Done.
```

Tìm tham số đầu tiên (và duy nhất) cho câu lệnh `printf` và xác nhận đó là địa chỉ của chuỗi định dạng do người dùng cung cấp:

`x/s <địa chỉ>`

```
(gdb) x/20xw $esp
0xfffffd500:    0xfffffd538    0xfffffd538    0x56557008    0x5655626b
0xfffffd510:    0xfffffd54a    0xf7ffc89c    0xf7ffc8a0    0xffffd654
0xfffffd520:    0xf7ffd000    0xf7ffc8a0    0xffffd54a    0x0000000d
0xfffffd530:    0x00000015    0x56557008    0x6e65694b    0x39305441
0xfffffd540:    0x00000038    0x00000000    0x00005034    0x7743c900
(gdb) x/s 0xfffffd538
0xfffffd538:    "KienAT098"
(gdb) █

Please enter your e-mail address: [B20DCAT098]
Started 1 containers, 1 completed initialization. Done.
```

4. Checkwork.

```
student@ubuntu:~/labtainer/labtainer-student$ checkwork
Results stored in directory: /home/student/labtainer_xfer/printf
Labname printf

Student          |      gdb_commands |
=====|=====|
B20DCAT098      |          226      |
What is automatically assessed for this lab:
```