1. **0-sr-1**

In this case, we use the concept of sequential read to get the stack cookie to pass the examination of it. After that, we can use ROP to trigger the execv function.

**sr.py**

**Text

Description automatically generated**

1. **1-ar-2**

In the case, the concept of arbitrary read was used. We can read 8 bytes from the address of printf\_got to get the address of libc\_printf. After that, the precise address of system can be calculated by the offset between printf and system. Finally, use the ROP to trigger the libc\_system.

**ar.py**

**Text

Description automatically generated**

1. **2-aw-1**

In this case, the concept of arbitrary write was used. We can exploit the function program provides to write the address of the function which we want to run into the address of printf\_got. Finally, the program will execute the please\_execute\_me when it calls the printf function.

**aw.py**

**Text

Description automatically generated**

1. **3-aw-2**

In this case, the concept of arbitrary read and write were used. First, we got the value of libc\_put by read\_function then use it to calculate the precise address of libc\_system. After that, we write the address of libc\_system into the address of printf\_got to trigger the system function then run the Writing file (set PATH).

**aw.py**

**Text

Description automatically generated**

1. **4-fs-read-1-32**

In this case, we used the concept of format string. We can use “%p” to get the value of the stack. Thus, we type like “%p %p %p %p %p %p %p” to get the several values then compare them to the answer. We can find that the answer will be at 6th value.

**\*\*\*\*\*\*\*\*program\*\*\*\*\*\*\*\***

**$ ./fs-read-1-32**

Please type your name first:

%p %p %p %p %p %p %p %p %p %p %p

Hello 0xffac3b6c 0x3f 0x804870a 0xf7f437eb (nil) **0x6d6155cc** 0x252070250x70252070 0x20702520 0x25207025 0x70252070

Can you guess the random?

0x6d6155cc

Great!

1. **5-fs-read-1-64**

In this case, we used the concept of format string. We can use “%p” to get the value of the stack. Thus, we type like “%p %p %p %p %p %p %p” to get the several values then compare them to the answer. We can find that the answer will be at 7th value.

**\*\*\*\*\*\*\*\*program\*\*\*\*\*\*\*\***

**$ ./fs-read-1-64**

Please type your name first:

%p %p %p %p %p %p %p %p %p %p %p %p

Hello 0x400b21 0x7f913d15b780 0x6 0x7f913d367700 0x6 (nil) **0xf0662cc100000000** 0x7025207025207025 0x2520702520702520 0x2070252070252070 0x7025207025207025 0xa702520

Can you guess the random?

0xf0662cc1

Great!

1. **6-fs-read-2-32**

In this case, we used the concept of format string. However, we cannot use the previous method to get the value because of the number limitation. Thus, we are going to use like %position$p to directly read the value at the designated position. By using gdb, we know that the random was putted at ebp-0x10 and the first address of the printed value. Thus, we can calculate the precise gap we need to cross then read the random number. **((0xff915758 - 0xff915504)/4 +1 →150)**

**\*\*\*\*\*\*\*\*program\*\*\*\*\*\*\*\***

**$ ./fs-read-2-32**

Please type your name first:

%150$p

Hello **0xdba34866**

Can you guess the random?

0xdba34866

Great!

1. **7-fs-read-2-64**

In this case, we used the concept of format string. However, we cannot use the previous method to get the value because of the number limitation. Thus, we are going to use like %position$p to directly read the value at the designated position. By using gdb, we know that the random was putted at ebp-0x1c and the first address of the printed value. Thus, we can calculate the precise gap we need to cross then read the random number. But we have to add 5 more number because it exists more 5 values before the top of stack. (**(0x7ffc3c54fd14 - 0x7ffc3c54fad0) / 8 + 1 + 5 →** **78)**

**\*\*\*\*\*\*\*\*program\*\*\*\*\*\*\*\***

**$ ./fs-read-2-64**

Please type your name first:

%78$p

Hello **0xd482a519**00000007

Can you guess the random?

0xd482a519

Great!

1. **8-fs-arbt-read-32**

In this case, the random number was stored as global variable and its address is fixed. Thus, we can use format string to read the value from the fixed address then extract the number we need.

**fs.py**

**Text

Description automatically generated**

1. **9-fs-arbt-read-64**
2. In this case, the random number was stored as global variable and its address is fixed. Thus, we can use format string to read the value from the fixed address then extract the number we need. But the address we want to read must be placed after the instruction because the program will stop if it meets the null in 64 bits system.

**fs.py**

**Text

Description automatically generated**

1. **a-fs-arbt-write-32**

In this case, we try to write the correct value into global\_random. First, we got the address of variable, global\_random. Separate it to two parts, then write the 0xfaceb00c into it.

It will be like:

"\x0c\xb0\x00\x00" (10 11 12 13)

then

"\xce\xfa\x00\x00" (12 13 14 15)

→"\x0c\xb0\xce\xfa" (finally)

**fs.py**

**Text

Description automatically generated**

1. **b-fs-arbt-write-64**

In this case, we try to write the correct value into global\_random. First, we got the address of variable, global\_random. Separate it to two parts, then write the 0xfaceb00c into it. In addition, the instruction must be placed before the address of global random because the program will stop when it meets the null.

**fs.py**

**Text

Description automatically generated**

1. **c-fs-code-exec-32**

In this case, we use printf as AR to leak the GOT of printf in the beginning. Then, get the address of system in libc. Finally, use printf as AW to overwrite GOT[‘printf’] = libc\_system.

**Text

Description automatically generatedText

Description automatically generatedfs.py**

1. **d-fs-code-exec-64**

In this case, we use printf as AR to leak the GOT of printf in the beginning. Then, get the address of system in libc. Finally, use printf as AW to overwrite GOT[‘printf’] = libc\_system. Furthermore, we instruction must placed before the address we want to read and write.

**Text

Description automatically generatedfs.py**

**Text

Description automatically generated**

1. **e-fs-code-exec-pie-64**

First, use 1st printf as sequential read primitive. Then, use 2nd printf as arbitrary read primitive, leak libc\_puts. Finally, use 3rd printf as arbitrary write primitive, overwrite PUTS got.

**fs.py**

**Text

Description automatically generated**

**Text

Description automatically generated**

****