Activity Network Security 2

Part 2 :

Q1. What is the attacker's IP address?

ANS : 192.168.64.2

```
^C[10/27/2025 06:34] seed@ubuntu:~$ hostname -I
192.168.64.2 fde9:d146:a7d3:9db8:55da:13bd:117:8d7c fde9:d146:a7d3:9db8:70e8:b6f
f:fe18:aa7f
```

Q2. What command did you use to run the attack?

ANS : sudo netwox 76 -i 192.168.1.107 -p 80

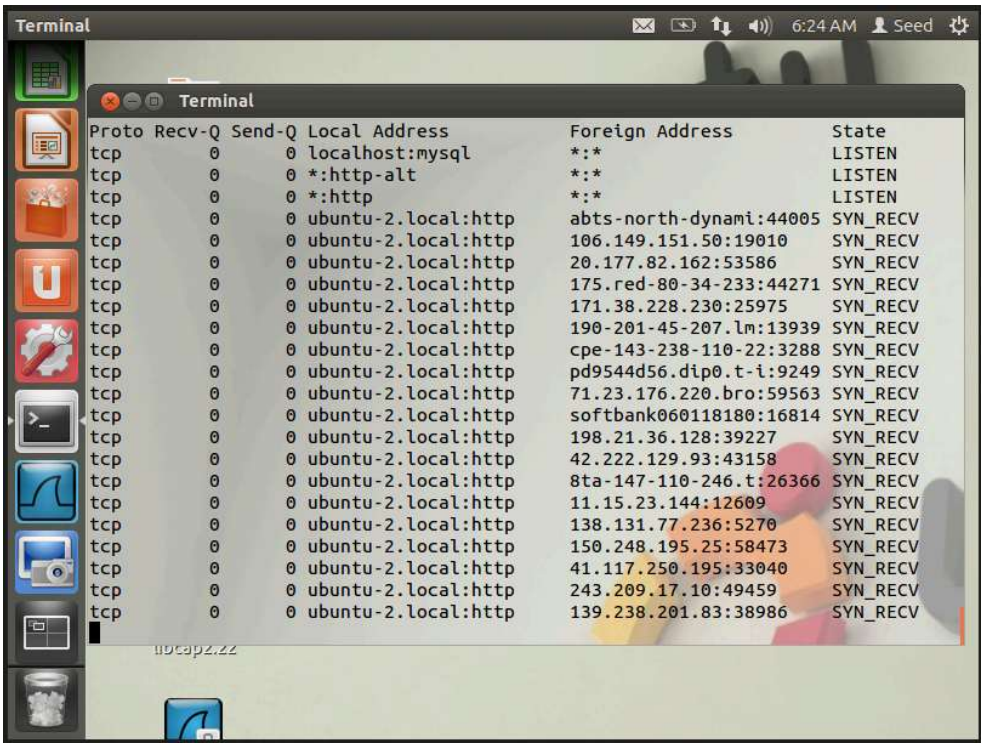
```
[10/27/2025 06:22] seed@ubuntu:~$ sudo netwox 76 -i 192.168.1.107 -p 80
^C[10/27/2025 06:24] seed@ubuntu:~$ sudo netwox 76 -i 192.168.1.107 -p 80
^C[10/27/2025 06:28] seed@ubuntu:~$ sudo netwox 76 -i 192.168.1.107 -p 80
^C[10/27/2025 06:32] seed@ubuntu:~$ sudo netwox 76 -i 192.168.1.107 -p 80
```

Q3. How do you know the attack is successful? Hint: Use the browser on your notebook to access the webpage. What should happen if the attack is successful?

ANS : Webpage is loading for long-time

Q4. "netwox" performs the TCP SYN Flood attack using spoofed IP addresses. Give some examples of the spoofed IP addresses you see on the target machine.

ANS : I use netstat -a instead of netstat -na, so some of them are shown in hostname/service names , but still some of them showing raw spoofed IP addresses



Q5. In the TCP SYN Flood attack, what resource on the server side is exhausted? What is the number of resources available, and how many of those resources get used up in the attack?

ANS : net.ipv4.tcp_max_syn_backlog : queue of pending unconfirmed SYN requests

```
[10/27/2025 06:32] seed@ubuntu:~$ sudo sysctl -w net.ipv4.tcp_max_syn_backlog=64
net.ipv4.tcp_max_syn_backlog = 64
```

I set it to 64, so its very easy to overwhelm

Q6. How do TCP SYN cookies prevent this type of attack?

ANS : prevent by not using backlog/queue and using SYN cookies instead when kernel detect backlog exhaustion or resource pressure, so it not allocating resources until client prove he is him

Here are the mechanism :

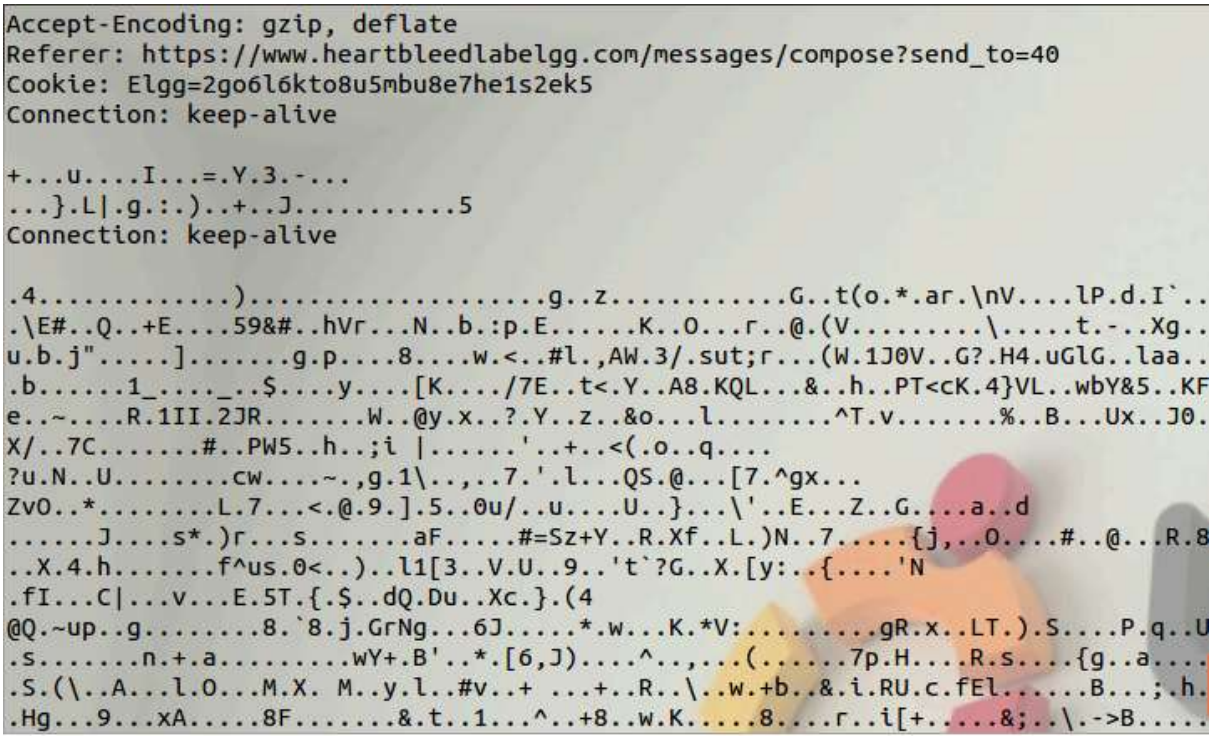
1. Normally, server using backlog.
2. With SYN cookies, server does not store state(queue). Instead, it encodes connection info (client IP, port, timestamp, etc.) inside the SYN-ACK's sequence number.
3. If the client is real, it replies with an ACK that includes this value.
4. Server verifies the cookie ⟶ recreates the connection state only then.

Part 3 :

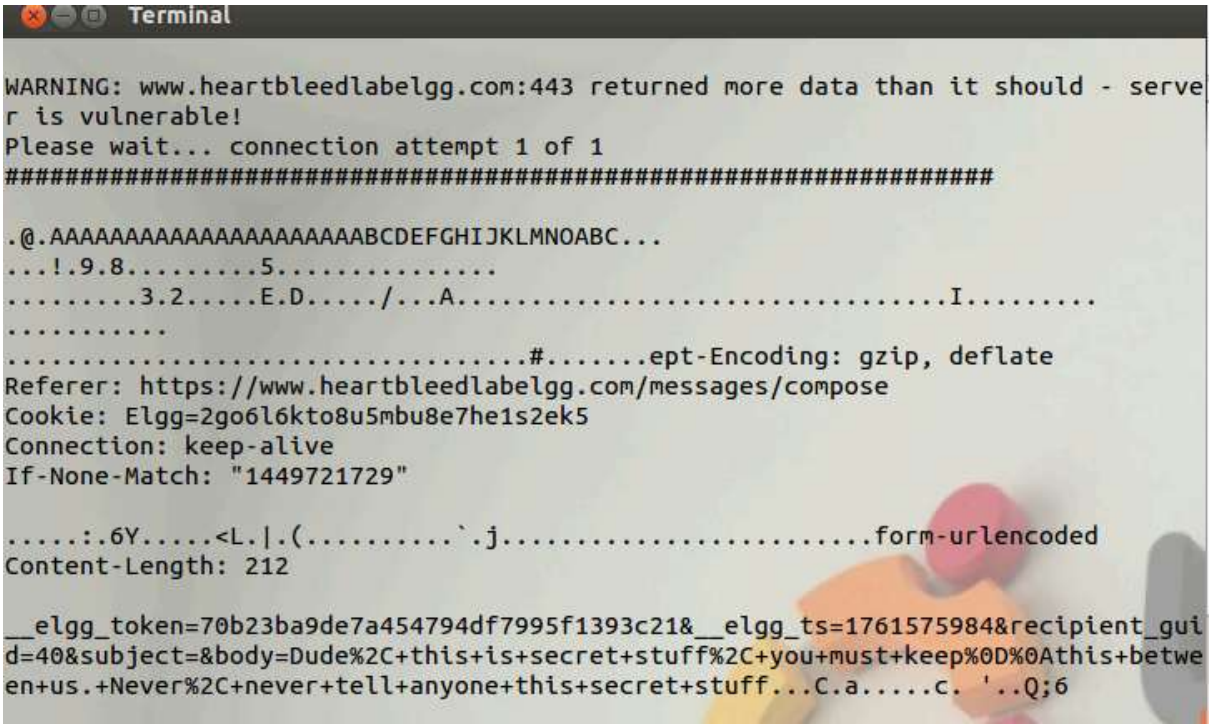
Q7. For each piece of secret that you steal from the Heartbleed attack, you need to show the screenshots as the proof. Upload a pdf of your screenshots.

Username & Password

User's activity : send_to = 40 (bobby)



The exact content of the private message : Dude, this is secret stuff, you must keep this between us. Never, never tell anyone this secret stuff.

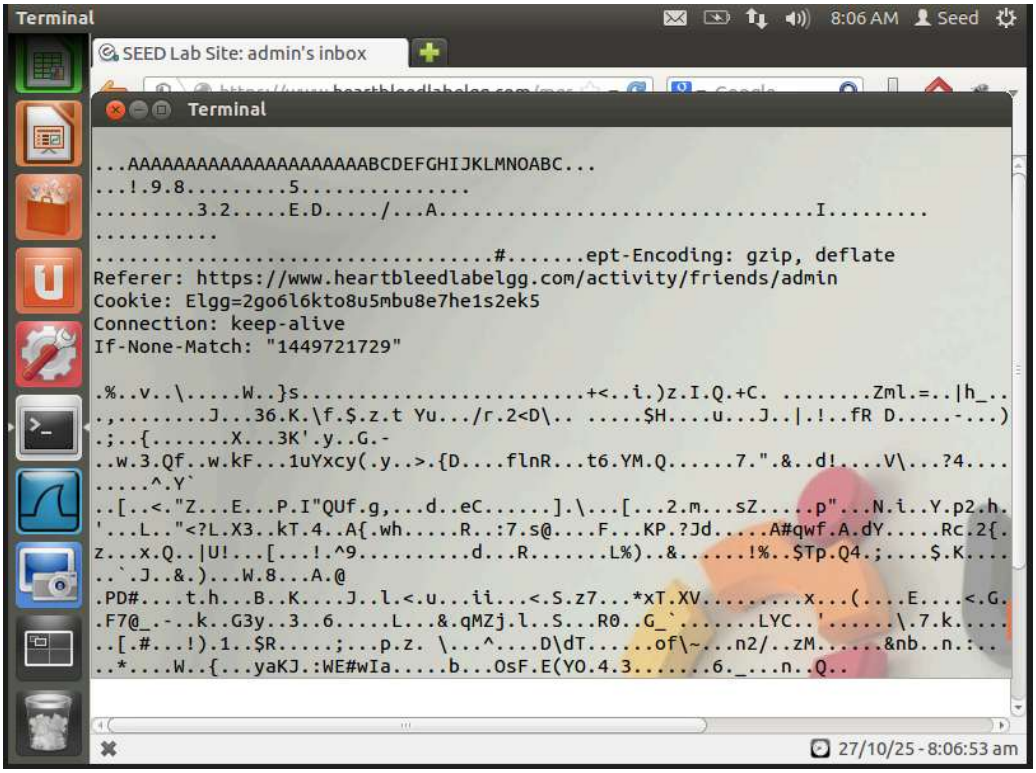


Q8. For the Heartbleed attack, explain how you did the attack, and what your observations are.

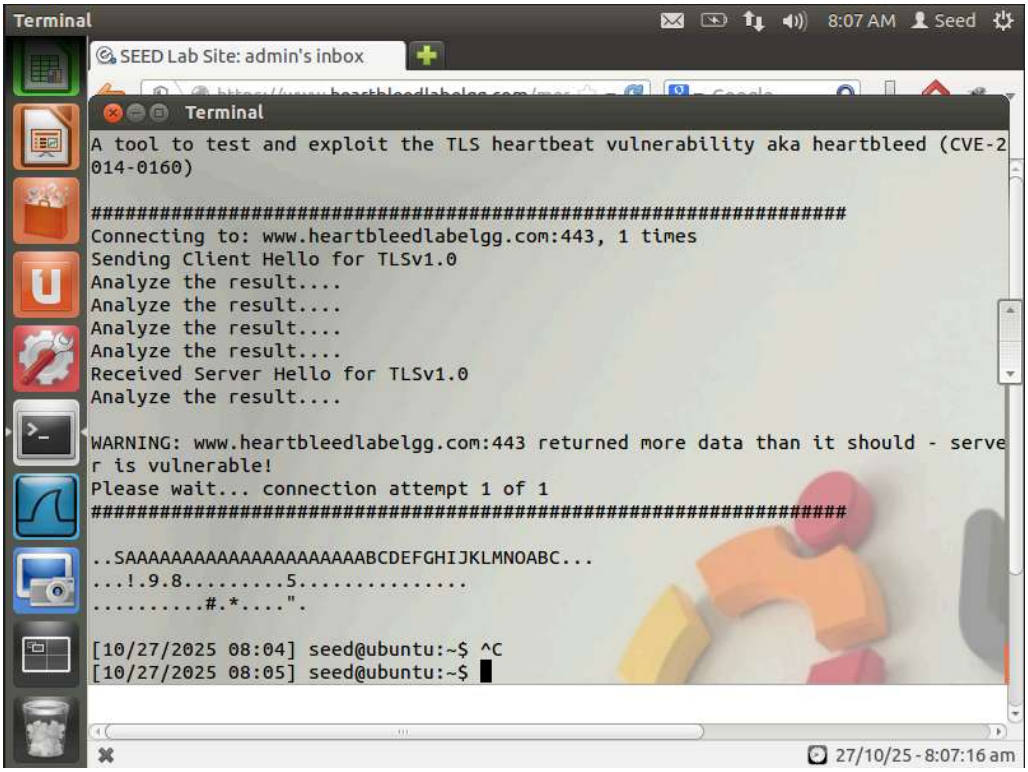
1. First have to chmod +x attack.py to make attack.py executable
2. Run attack.py multiple times
3. Everytime run I think we will get a part of log file, and we have to find out valuable information from these part of log

- Q9: As the length variable decreases, what kind of difference can you observe?

–l 4000



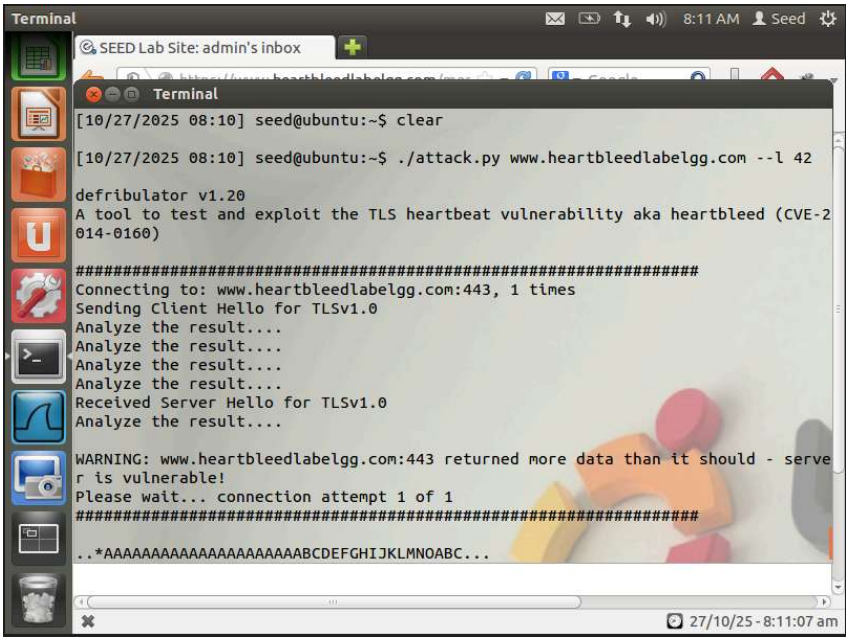
–l 83



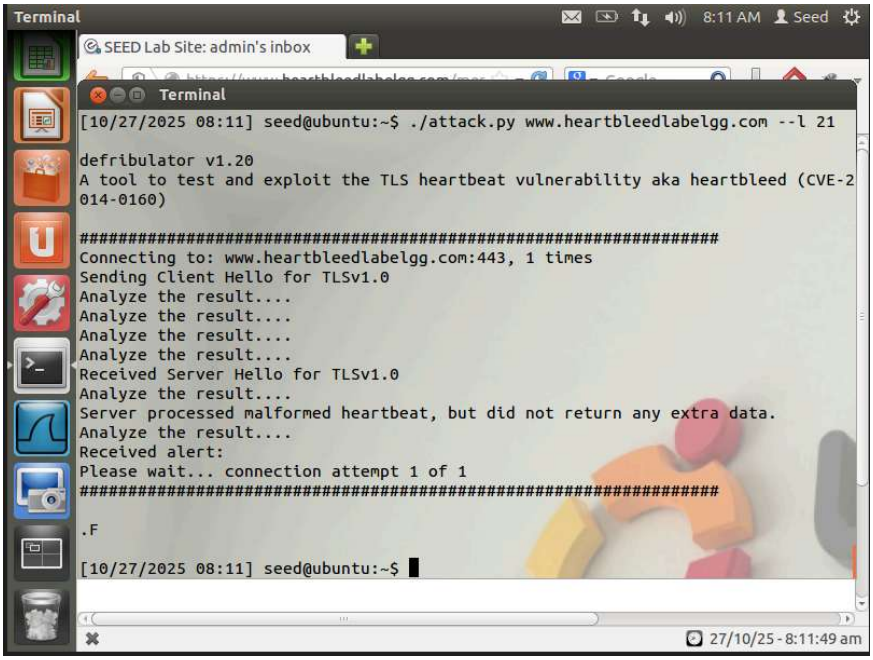
ANS : bigger number -> bigger part of log is display

• Q10: As the length variable decreases, there is a boundary value for the input length variable. At or below that boundary, the Heartbeat query will receive a response packet without attaching any extra data (which means the request is benign). Please find that boundary length. You may need to try many different length values until the web server sends back the reply without extra data. To help you with this, when the number of returned bytes is smaller than the expected length, the program will print "Server processed malformed Heartbeat, but did not return any extra data." What is the boundary length?

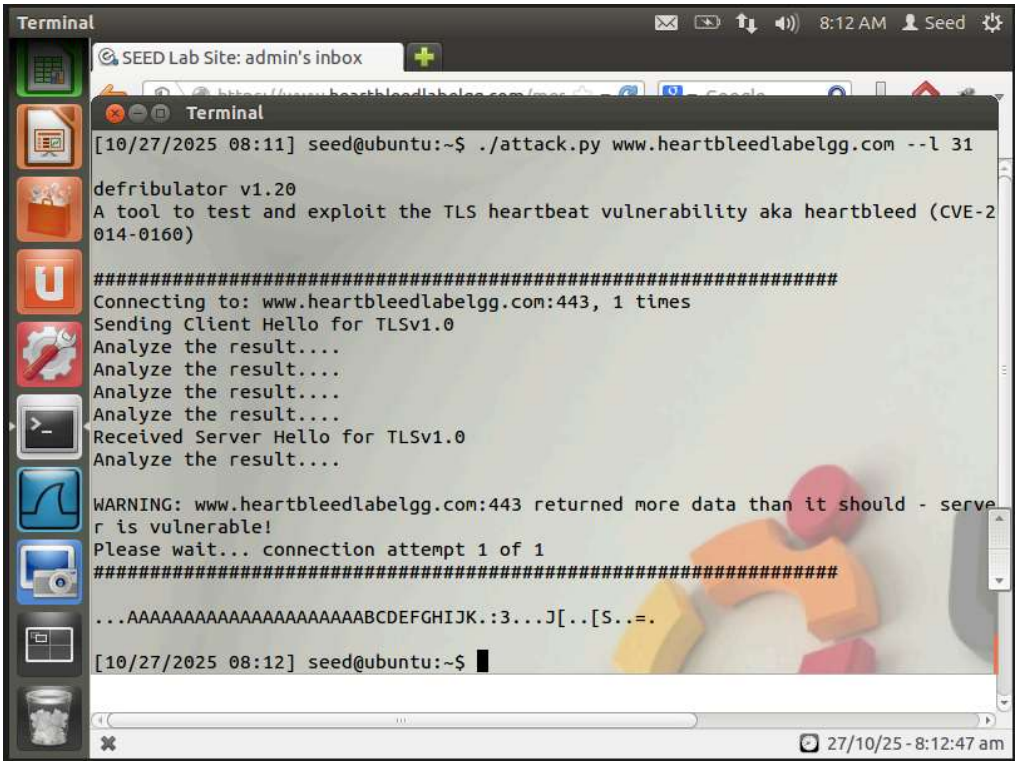
ANS : Since –l 83 still warning so we using binary search start from 1+83 = 42



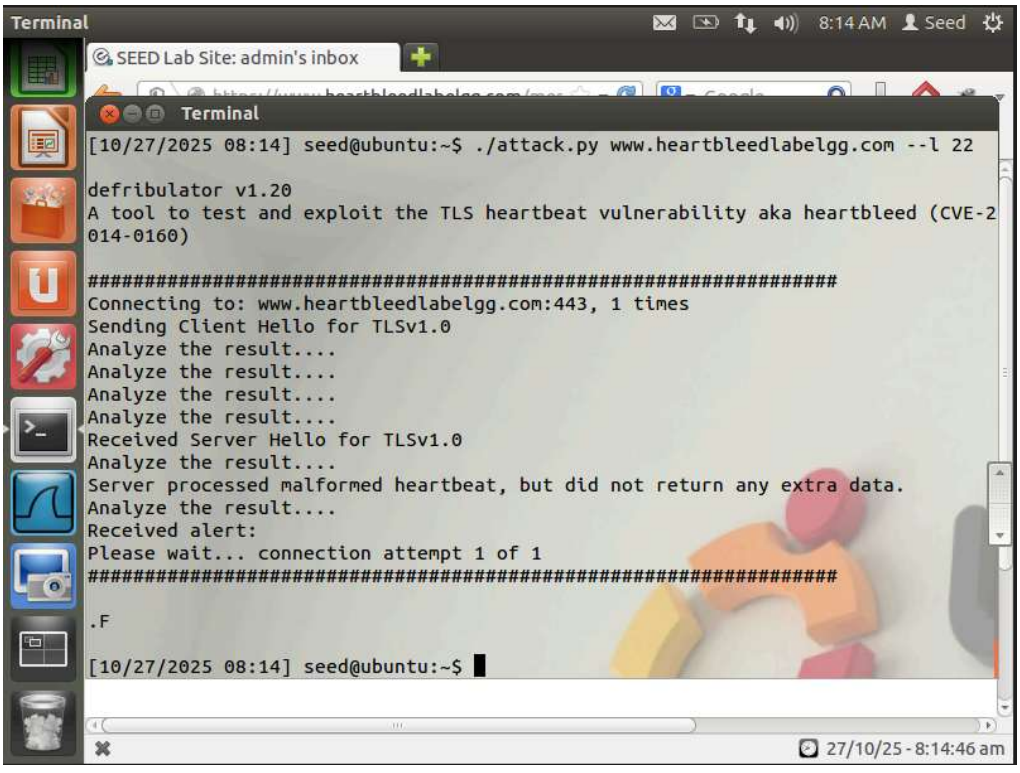
Still warning -> 1 + 41 /2 = 21



No warning so 22 + 41 / 2 = 31



Warning so 22 + 30 /2 = 26 -> Warning so 22+25/2 = 23 -> Warning so try 22



Boundary is 22

Q11. Try your attack again after you have updated the OpenSSL library. Are you successful at stealing data from the server after the upgrade?

ANS : No. After updating OpenSSL, the server no longer leaks extra data in response to malformed Heartbeat requests.

Q12. Please point out the problem from the code and provide a solution to fix the bug (i.e., what modification is needed to fix the bug). You do not need to recompile the code; just describe how you can fix the problem.

ANS : Bug and fix:

- Problem: the Heartbeat handler trusts the client-supplied payload_length and copies payload_length bytes back (using memcpy) /wo verifying that this length is $\leq$ the actual bytes present in the record. That over-reads past the received payload and returns unrelated memory.

- Fix: validate bounds before copying. Example rule: if 1 (type) + 2 (length) + payload_length + padding > record_length, then drop the request and send no reply. Only then allocate/copy payload_length bytes. Optionally zero the reply buffer.

Q13. Comment on the following discussions by Alice, Bob, and Eva regarding the fundamental cause of the Heartbleed vulnerability: Alice thinks the fundamental cause is missing the boundary checking during the buffer copy; Bob thinks the cause is missing the user input validation; Eva thinks that we can just delete the length value from the packet to solve everything. Who do you agree and disagree with, and why?

ANS :

Alice I agree with Alice. That is the precise root cause.

Bob It's right but vague; the missing validation is the length vs actual data size check.

Eva I disagree with Eva, The protocol needs the length because heartbeat data may contain arbitrary bytes; removing it breaks the spec and wouldn't prevent other length-mishandling bugs.