

Activity: Network Security Vulnerabilities

Instructor : Kunwadee Sripanidkulchai, Ph.D.

Overview

In this activity, we will work on understanding vulnerabilities that impact network security and how to fix them. The goal is to learn first hand that **designing secure protocols is challenging**. There are 3 parts to this activity: (1) Preparation, (2) DoS (Denial of Service), and (3) Protocol Security (SSL Vulnerabilities).

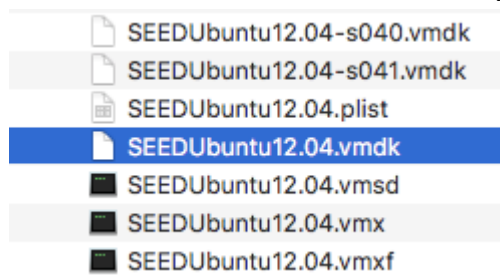
Credits: This set of activities is modified from the SEED project at <http://www.cis.syr.edu/~wedu/seed>.

Part I: Prepare your target VM

Pair up with another student, and have one student be the target and one student be the attacker. Your target is the SeedUbuntu VM that has already been set up ready for use from the SEED project. You should have already downloaded it from the link posted on the course Facebook page.

1. On both attacker and target notebooks, unzip the VM. This is an Ubuntu 12.04 VM with some older versions of software packages that have vulnerabilities. Do not attempt to update this VM. We will call this VM the SeedVM.

2. In VirtualBox, create a new VM. Provide a Name and Select the OS Type (Linux) and Version (32-bit Ubuntu). Create the VM using an existing hard disk and pick the file SEEDUbuntu12.04.vmdk from your unzipped directory to use as the hard disk.



3. Make sure that the VM uses the Bridge network.

4. Boot the VM. To login, you need to first login as an unprivileged user.

User ID: seed, Password: dees

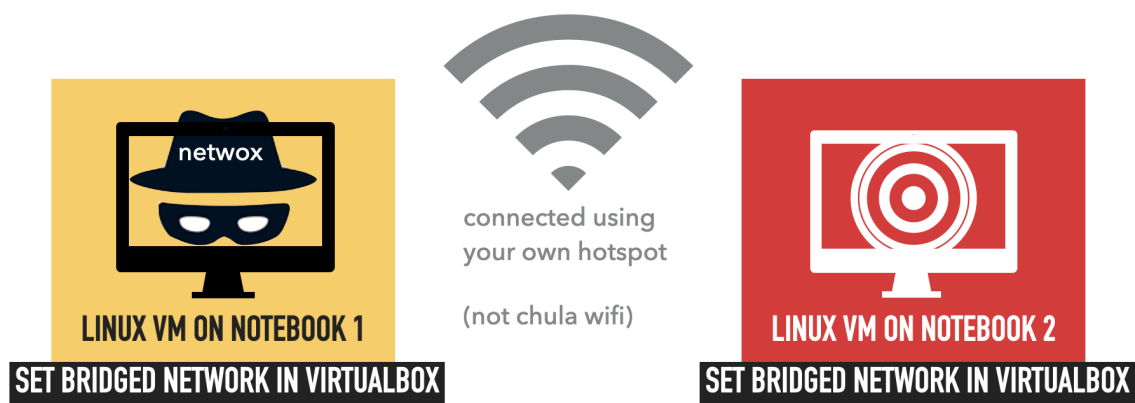
Once in, you can run as root (su or sudo).

User ID: root, Password: seedubuntu

5. Do not agree to any OS/software updates.

Part II. DoS (Denial of Service)

We will attempt to perform DoS on the SeedVM using the TCP SYN Flood Attack. We covered the TCP SYN Flood Attack in lecture. We will set up the attacker and the target. The attacker will be on the SeedVM on one notebook. The target (victim) can be on your friend's SEEDVM but you must set up the network as indicated using bridged networking and connect both attacker and target to a common wifi hotspot (not ChulaWifi).



SYN flood is a form of DoS attack in which attackers send many SYN requests to a victim's TCP port, but the attackers have no intention to finish the 3-way handshake procedure. Attackers either use spoofed IP addresses or do not continue the procedure. Through this attack, attackers can flood the victim's queue that is used for half-opened connections, i.e. connections that have finished SYN, SYN-ACK, but have not yet gotten a final ACK back. When this queue is full, the victim cannot take any more connections.

1. On the target VM, check the size of the queue which is a system-wide setting. In Linux, we can check the setting using the following command. Record the result of this command as you will need it for Q/A.

```
$ sudo sysctl -q net.ipv4.tcp_max_syn_backlog
```
2. We can use the netstat command to check the usage of the queue, i.e., the number of half-opened connections associated with a listening port. The state for such connections is SYN-RECV. If the 3-way handshake is finished, the state of the connections will be ESTABLISHED. Use a browser on your notebook to look at the webpage on the target VM (use the target VM's IP address). Record the result from running the netstat command. What ports do you see open and what connections and states do you see after you get the webpage?

```
$ netstat -a  
$ netstat -na
```

3. In this task, you need to demonstrate the SYN flooding attack. You can run the Netwox tool from the attack VM to conduct the attack. While the attack is going on, run the "netstat -na" command on the target VM, and compare the result with that before the attack. Please also describe how you know whether the attack is successful or not.

The corresponding Netwox tool for this task is numbered 76. To get the help information for how to run this attack, type

```
$ netwox 76 --help
```

4. SYN Cookie Countermeasure: If your attack seems unsuccessful, one thing that you can investigate is whether the SYN cookie mechanism is turned on. SYN cookie is a defense mechanism to counter the SYN flooding attack. The mechanism will kick in if the machine detects that it is under the SYN flooding attack. You can use the sysctl command to turn on/off the SYN cookie mechanism:

```
$ sudo sysctl -a | grep cookie      (Display the SYN cookie flag)  
$ sudo sysctl -w net.ipv4.tcp_syncookies=0 (turn off SYN cookie)  
$ sudo sysctl -w net.ipv4.tcp_syncookies=1 (turn on SYN cookie)
```

Please run your attacks with the SYN cookie mechanism on and off, and compare the results. In your Q/A, please describe why the SYN cookie can effectively protect the machine against the SYN flooding attack. If you are using a browser to test, make sure that you force refresh to make the browser try to access the server instead of serving you content from its cache.

Answer these questions and provide corresponding evidence/screenshots:

Q1. What is the attacker's IP address?

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

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?

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.

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?

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

Part III. SSL Vulnerabilities

The Heartbleed bug (CVE-2014-0160) which we covered in lecture is a severe implementation flaw in the OpenSSL library, which enables attackers to steal data from the memory of the target (victim) server. The contents of the stolen data depend on what is there in the memory of the server. It could potentially contain private keys, TLS session keys, user names, passwords, credit cards, etc. The vulnerability is in the implementation of the Heartbeat protocol, which is used by SSL/TLS to keep the connection alive.

The objective of this lab is for students to understand how **serious** this vulnerability is, how the attack works, and how to fix the problem. The affected OpenSSL version range is from 1.0.1 to 1.0.1f. The version in our Ubuntu VM is 1.0.1. You can check the version using the command in the SeedVM

```
$ openssl version
```

Since it is illegal to attack any real system, we will use the SeedVM as the target for this attack. The attacker will be your friend's notebook. Questions Q7-13 are interleaved with the instructions.

1. Make sure the target SeedVM is configured to use the bridge network in VirtualBox, and the attacker and target are using the same wifi hotspot (not ChulaWifi).
2. Set up a static DNS name mapping for the name www.heartbleedlabelgg.com on the attacker notebook. If you are using a Mac, you can modify the `/private/etc/hosts` file by adding a line like this where the IP address is the IP address of the SeedVM (target). If you are using Linux, then modify the `/etc/hosts` file. If you are using Windows, you need to edit the file `C:\Windows\System32\Drivers\etc\hosts`. Look here for more info for Windows. <https://www.petri.com/easily-edit-hosts-file-windows-10>

```
# network security lab
192.168.56.102 www.heartbleedlabelgg.com
```

3. In the attacker notebook, open up a browser, and create some state (memory) on the SeedVM web server. Go to <https://www.heartbleedlabelgg.com> and use the web app as a normal legitimate user as follows:
 - Login as the site administrator. (User Name:admin; Password:seedelgg)
 - Add Bobby as friend. (Go to More -> Members and click Bobby -> Add Friend)

- Send Bob a private message “Dude, this is secret stuff, you must keep this between us. Never, never tell anyone this secret stuff.”
- 4. After you have done enough interaction as legitimate users, you can launch the attack and see what information you can get out of the victim server. Writing the program to launch the Heartbleed attack from scratch is not easy, because it requires the low-level knowledge of the Heartbeat protocol. Fortunately, other people have already written the attack code. Therefore, we will use the existing code to gain first-hand experience in the Heartbleed attack. The code that we use is called `attack.py`, which was originally written by Jared Stafford. We made some small changes to the code for educational purposes. Download the code, `attack.py`, from google drive onto the attacker notebook and change its permission so the file is executable. Then run the attack code as follows:

```
$ ./attack.py www.heartbleedlabelgg.com
```

You may need to run the attack code multiple times to get useful data. Try and see whether you can get the following information from the target server.

 - Username and password.
 - User’s activity (what the user has done).
 - The exact content of the private message.

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.

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

- 5. Find the cause of Heartbleed: In this task, you will compare the outcome of the benign packet and the malicious packet sent by the attacker code to find out the fundamental cause of the Heartbleed vulnerability.

The Heartbleed attack is based on the Heartbeat request. This request just sends some data to the server, and the server will copy the data to its response packet, so all the data is echoed back. In the normal case, suppose that the request includes 3 bytes of data “ABC”, so the length field has a value 3. The server will place the data in the memory, and copy 3 bytes from the beginning of the data to its response packet. In the attack scenario, the request may contain 3 bytes of data, but the length field may say 1003. When the server constructs its response packet, it copies from the starting of the data (i.e. “ABC”), but it copies 1003 bytes, instead of 3 bytes. These extra 1000 bytes obviously do not come from the request packet; they come from the server’s private memory, and they may contain other user’s information, secret keys, password, etc.

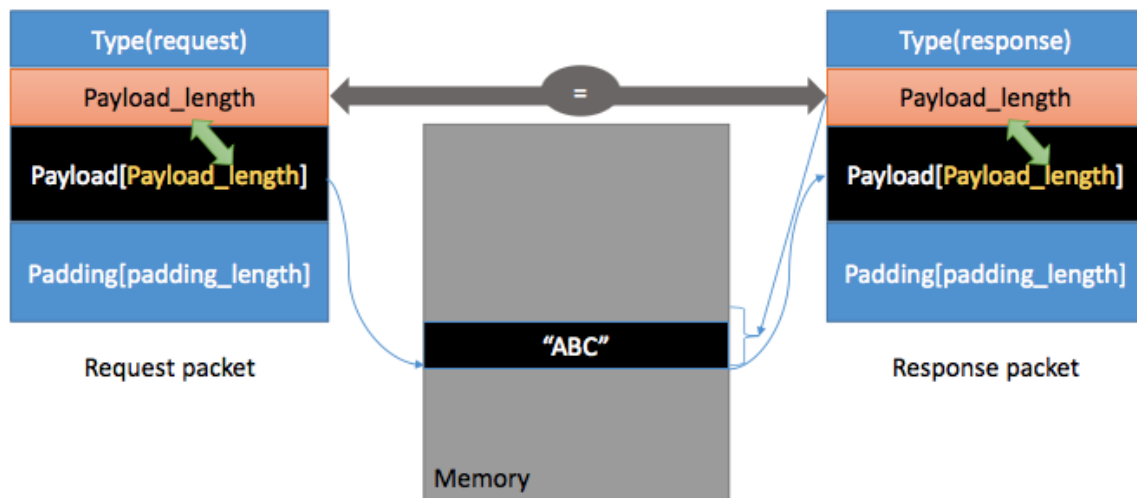


Figure 1: The Benign Heartbeat Communication

In this task, we will play with the length field of the request. First, let's understand how the Heartbeat response packet is built from Figure 1. When the Heartbeat request packet comes, the server will parse the packet to get the payload and the Payload length value (which is highlighted in Figure 1). Here, the payload is only a 3-byte string "ABC" and the Payload length value is exactly 3. The server program will blindly take this length value from the request packet. It then builds the response packet by pointing to the memory storing "ABC" and copy Payload length bytes to the response payload. In this way, the response packet would contain a 3-byte string "ABC".

We can launch the HeartBleed attack like what is shown in Figure 2. We keep the same payload (3 bytes), but set the Payload length field to 1003. The server will again blindly take this Payload length value when building the response packet. This time, the server program will point to the string "ABC" and copy 1003 bytes from the memory to the response packet as a payload. Besides the string "ABC", the extra 1000 bytes are copied into the response packet, which could be anything from the memory, such as secret activity, logging information, password and so on.

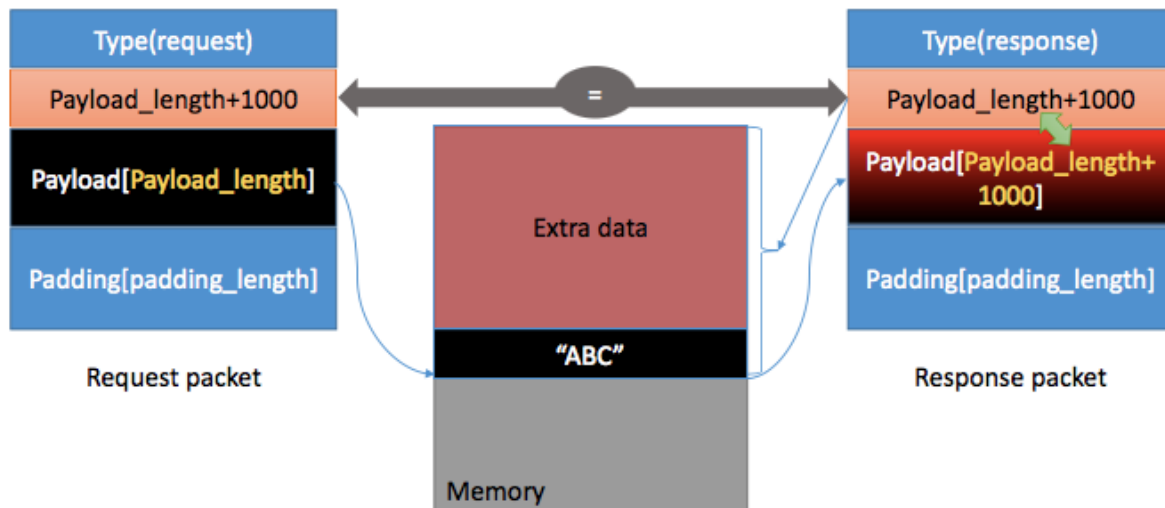


Figure 2: The Heartbleed Attack Communication

Our attack code allows you to play with different Payload length values. By default, the value is set to a quite large one (0x4000), but you can reduce the size using the command option "-l" (letter ell) or "--length" as shown in the following examples:

```
./attack.py www.heartbleedlabelgg.com -l 0x015B
./attack.py www.heartbleedlabelgg.com --length 83
```

Your task is to play with the attack program with different payload length values and answer the following questions in the google sheet:

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

6. Fixing Heartbleed: To fix the Heartbleed vulnerability, the best way is to update the OpenSSL library to the newest version. This can be achieved using the following commands. It should be noted that once it is updated, it is hard to go back to the vulnerable version. Therefore, make sure you have finished the previous tasks before doing the update. You can also take a snapshot of your VM before the update.

```
$ sudo apt-get update  
$ sudo apt-get upgrade
```

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

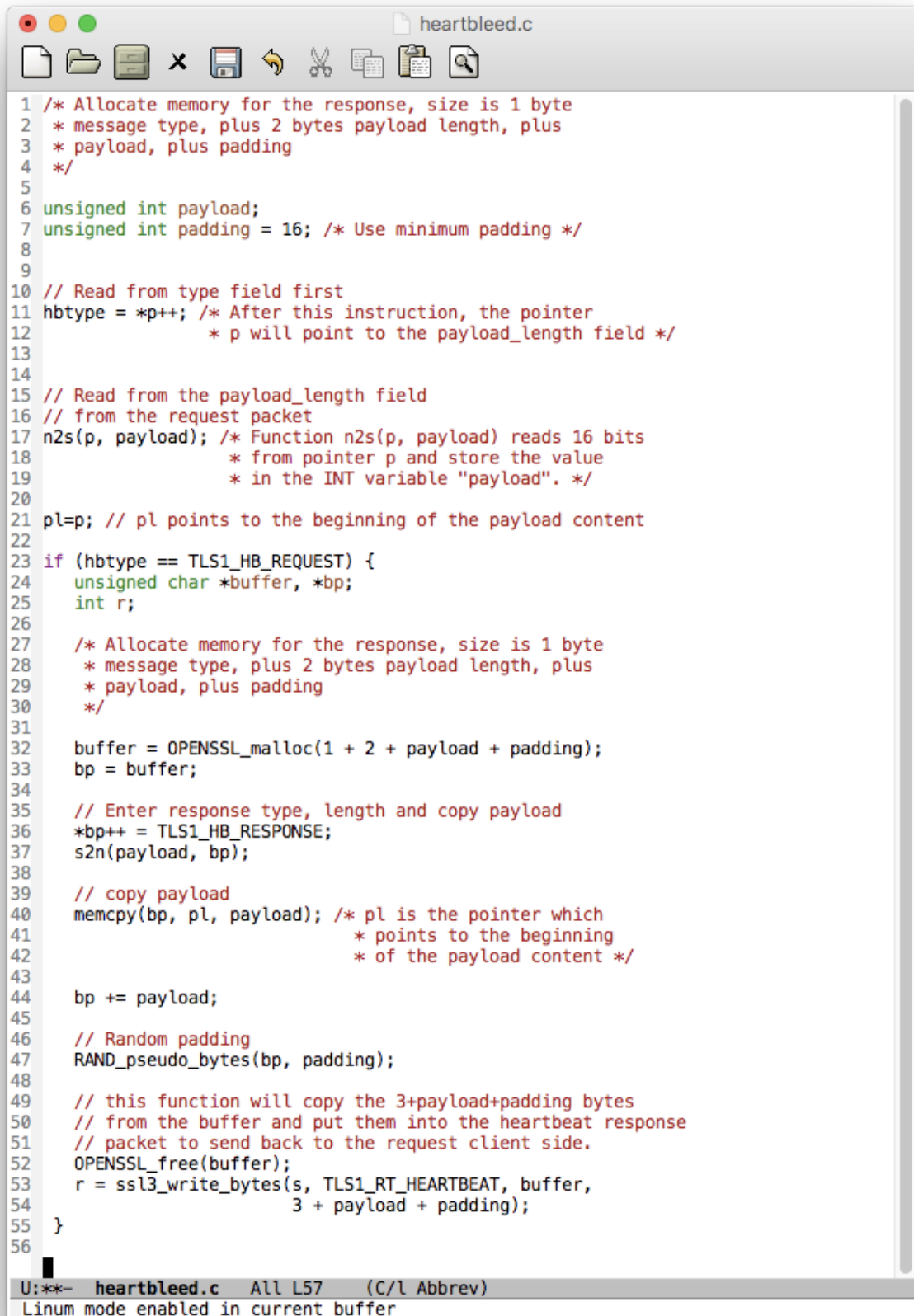
7. Understanding the code behind the Heartbleed problem and the fix: The following C-style structure (not exactly the same as the source code) is the format of the Heartbeat request/response packet.

```
struct {  
    HeartbeatMessageType type; // 1 byte: request or the response  
    uint16 payload_length;      // 2 byte: the length of the payload  
    opaque payload[HeartbeatMessage.payload_length];  
    opaque padding[padding_length];  
} HeartbeatMessage;
```

The first field (1 byte) of the packet is the type information, and the second field (2 bytes) is the payload length, followed by the actual payload and paddings. The size of the payload should be the same as the value in the payload length field, but in the attack scenario, payload length can be set to a different value. The following code snippet shows how the server copies the data from the request packet to the response packet.

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.

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?



```
1 /* Allocate memory for the response, size is 1 byte
2  * message type, plus 2 bytes payload length, plus
3  * payload, plus padding
4  */
5
6 unsigned int payload;
7 unsigned int padding = 16; /* Use minimum padding */
8
9
10 // Read from type field first
11 hbtype = *p++; /* After this instruction, the pointer
12                * p will point to the payload_length field */
13
14
15 // Read from the payload_length field
16 // from the request packet
17 n2s(p, payload); /* Function n2s(p, payload) reads 16 bits
18                  * from pointer p and store the value
19                  * in the INT variable "payload". */
20
21 pl=p; // pl points to the beginning of the payload content
22
23 if (hbtype == TLS1_HB_REQUEST) {
24     unsigned char *buffer, *bp;
25     int r;
26
27     /* Allocate memory for the response, size is 1 byte
28      * message type, plus 2 bytes payload length, plus
29      * payload, plus padding
30      */
31
32     buffer = OPENSSL_malloc(1 + 2 + payload + padding);
33     bp = buffer;
34
35     // Enter response type, length and copy payload
36     *bp++ = TLS1_HB_RESPONSE;
37     s2n(payload, bp);
38
39     // copy payload
40     memcpy(bp, pl, payload); /* pl is the pointer which
41                              * points to the beginning
42                              * of the payload content */
43
44     bp += payload;
45
46     // Random padding
47     RAND_pseudo_bytes(bp, padding);
48
49     // this function will copy the 3+payload+padding bytes
50     // from the buffer and put them into the heartbeat response
51     // packet to send back to the request client side.
52     OPENSSL_free(buffer);
53     r = ssl3_write_bytes(s, TLS1_RT_HEARTBEAT, buffer,
54                        3 + payload + padding);
55 }
56
```

U:*** heartbleed.c All L57 (C/l Abbrev)
Linum mode enabled in current buffer