

Lab 4 – Heartbleed

Goal: To fully understand the weakness of the implementation of the Heartbeat protocol.

Instructions: Please refer to attached lab instructions with this document.

Deliverable: A lab report, an electronic submission to **Canvas**, is expected to **explain all the commands** that you use, and **include the screen shots** when you achieve the major milestones in the lab, such as when you get the credentials of the victim website. A demo may be requested when necessary.

Requirement: The report will all be evaluated based on the following grading criteria.

Correctness	25%
Completeness	25%
Clarity	25%
Quality of English writing	25%

Heartbleed Attack Lab

Copyright © 2016 Wenliang Du, Syracuse University.

The development of this document was partially funded by the National Science Foundation under Award No. 1303306 and 1318814. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License. A human-readable summary of (and not a substitute for) the license is the following: You are free to copy and redistribute the material in any medium or format. You must give appropriate credit. If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original. You may not use the material for commercial purposes.

1 Overview

The Heartbleed bug (CVE-2014-0160) is a severe implementation flaw in the OpenSSL library, which enables attackers to steal data from the memory of the 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.

2 Lab Environment

In this lab, we need to set up two VMs: one called attacker machine and the other called victim server. We use the pre-built SEEDUbuntu12.04 VM. The VMs need to use the NAT-Network adapter for the network setting. This can be done by going to the VM settings, picking Network, and clicking the Adaptor tag to switch the adapter to NAT-Network. Make sure both VMs are on the same NAT-Network.

The website used in this attack can be any HTTPS website that uses SSL/TLS. However, since it is illegal to attack a real website, we have set up a website in our VM, and conduct the attack on our own VM. We use an open-source social network application called ELGG, and host it in the following URL: <https://www.heartbleedlabelgg.com>.

We need to modify the `/etc/hosts` file on the attacker machine to map the server name to the IP address of the server VM. Search the following line in `/etc/hosts`, and replace the IP address 127.0.0.1 with the actual IP address of the server VM that hosts the ELGG application.

```
127.0.0.1 www.heartbleedlabelgg.com
```

3 Lab Tasks

Before working on the lab tasks, you need to understand how the heartbeat protocol works. The heartbeat protocol consists of two message types: HeartbeatRequest packet and HeartbeatResponse packet. Client sends a HeartbeatRequest packet to the server. When the server receives it, it sends back a copy of the received message in the HeartbeatResponse packet. The goal is to keep the connection alive. The protocol is illustrated in Figure 1.

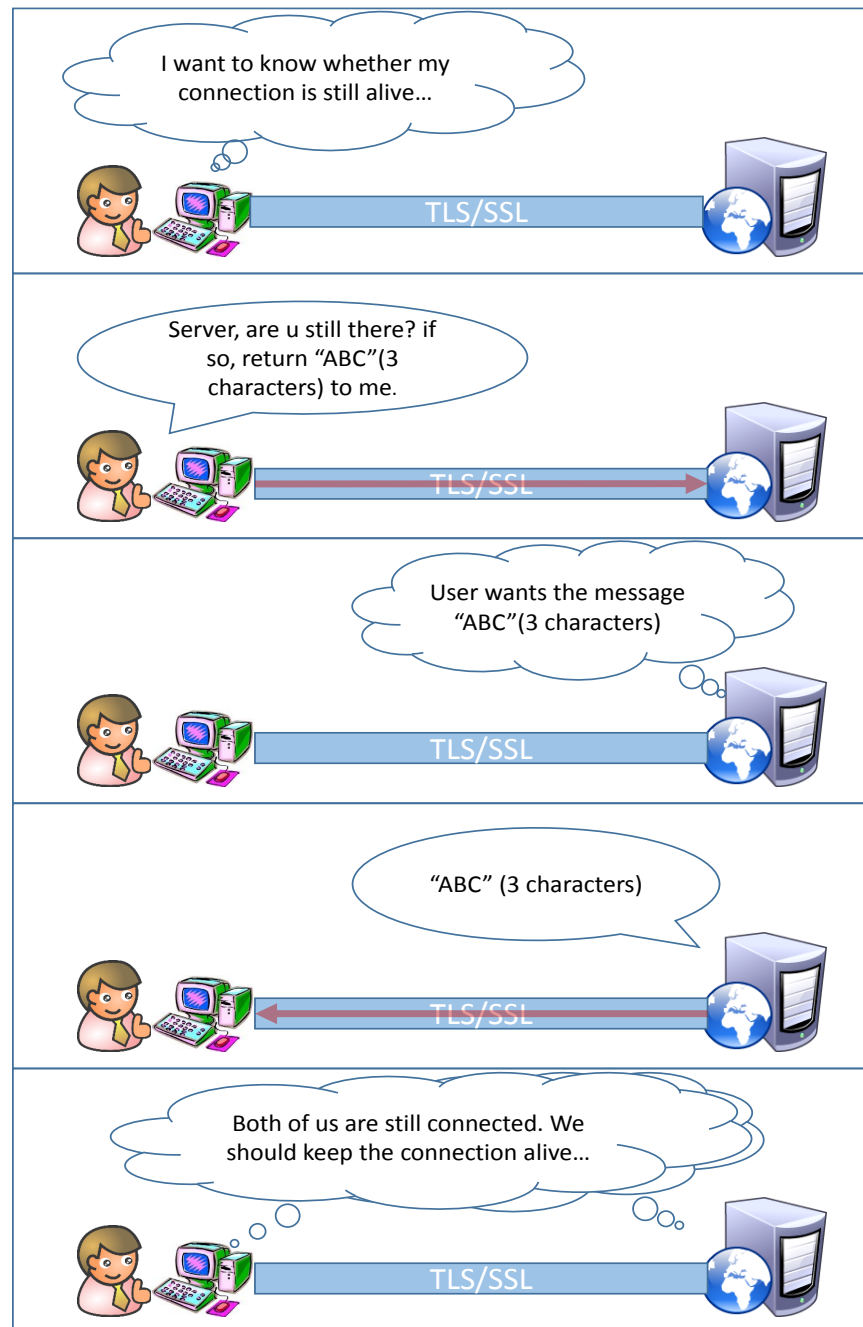


Figure 1: Overview of the Heartbeat Protocol

3.1 Task 1: Launch the Heartbleed Attack.

In this task, students will launch the Heartbleed attack on our social network site and see what kind of damages can be achieved. The actual damage of the Heartbleed attack depends on what kind of information is stored in the server memory. If there has not been much activity on the server, you will not be able to steal useful data. Therefore, we need to **interact** with the web server as legitimate users. Let us do it as the

administrator, and do the followings:

- Visit `https://www.heartbleedlabelgg.com` from your browser.
- Login as the site administrator. (User Name:admin; Password:seedelgg)
- Add Bobby as friend. (Go to More -> Members and click Bobby -> Add Friend)
- Send Bobby a private message.

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. You can download the code from the lab's web site, change its permission so the file is executable. You can 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.

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

For each piece of secret that you steal from the Heartbleed attack, you need to show the **screen-dump** as the proof and explain how you did the attack, and what your observations are.

3.2 Task 2: Find the Cause of the Heartbleed Vulnerability

In this task, students 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 are 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.**

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 2. 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 2). 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

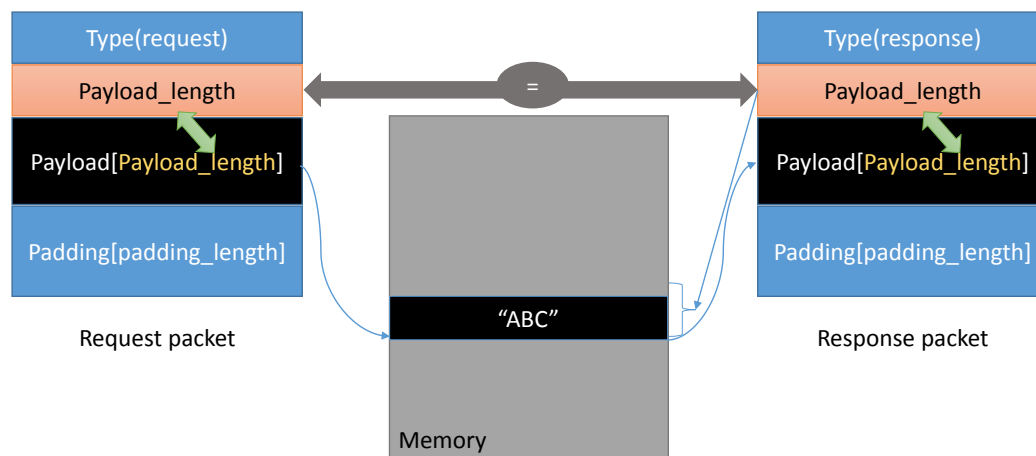


Figure 2: The Benign Heartbeat Communication

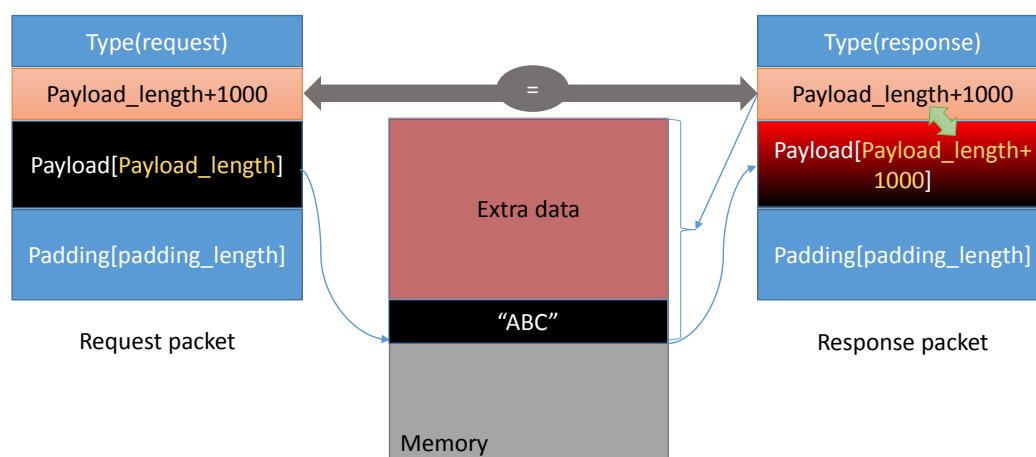


Figure 3: The Heartbleed Attack Communication

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 3. 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.

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:

- **Question 2.1:** As the length variable decreases, what kind of difference can you observe?
- **Question 2.2:** 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."

3.3 Task 3: Countermeasure and Bug Fix

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
```

Task 3.1 Try your attack again after you have updated the OpenSSL library. Please describe your observations.

Task 3.2 The objective of this task is to figure out how to fix the Heartbleed bug in the source code. 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.

Listing 1: Process the Heartbeat request packet and generate the response packet

```
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  // Read from type field first
10 hbtype = *p++; /* After this instruction, the pointer
11                * p will point to the payload_length field *.
12
13 // Read from the payload_length field
14 // from the request packet
15 n2s(p, payload); /* Function n2s(p, payload) reads 16 bits
16                  * from pointer p and store the value
17                  * in the INT variable "payload". */
18
19
20 pl=p; // pl points to the beginning of the payload content
21
22 if (hbtype == TLS1_HB_REQUEST)
23 {
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 }
```

Please point out the problem from the code in Listing 1 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 in your lab report.

Moreover, please 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.

4 Submission

Students need to submit a detailed lab report to describe what they have done and what they have observed. Report should include the evidences to support the observations. Evidences include packet traces, screen dumps, etc.

References

- [1] Heartbleed attack - Implementation: https://alexandreborgesbrazil.files.wordpress.com/2014/04/heartbleed_attack_version_a_1.pdf
- [2] Heartbleed attack - Interesting explanation: <http://xkcd.com/1354/>