Lab 10 Hearbleed Attack

> 16307130212 管佳乐

Task 0: Lab Environment

change the server record and redirect the domain the name to our victim server

```
$ sudo vi /etc/hosts
10.0.2.11 www.heartbleedlabelgg.com
```

Task 1: Launch the Heartbleed Attack

Visit in browser

```
https://www.heartbleedlabelgg.com
```

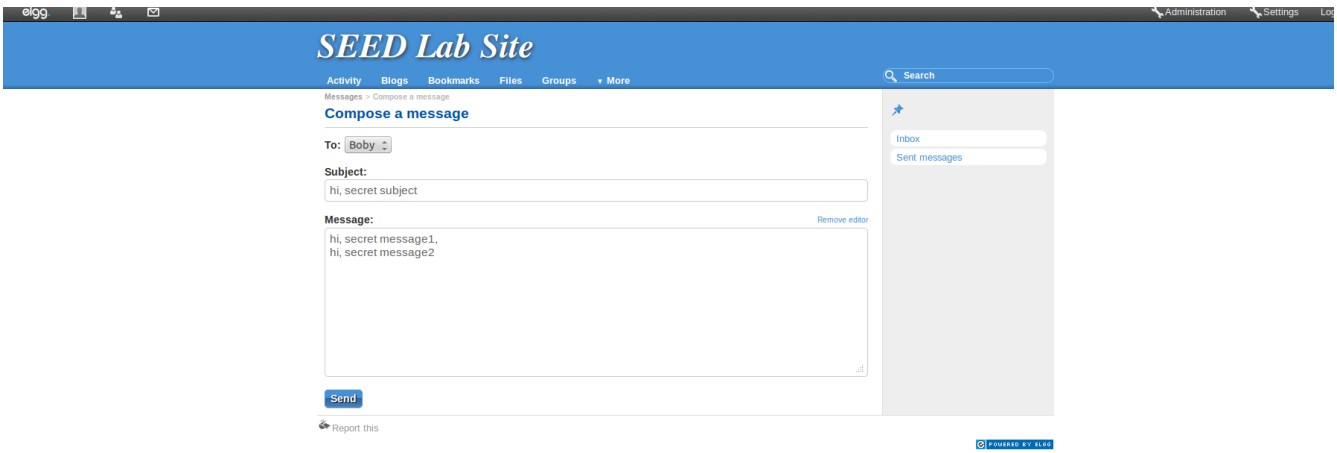
Login as

```
User: admin
Boby: seedelgg
```

Add Boby as friend

```
More -> Members
Boby -> Add Friend
```

Send Boby a private message

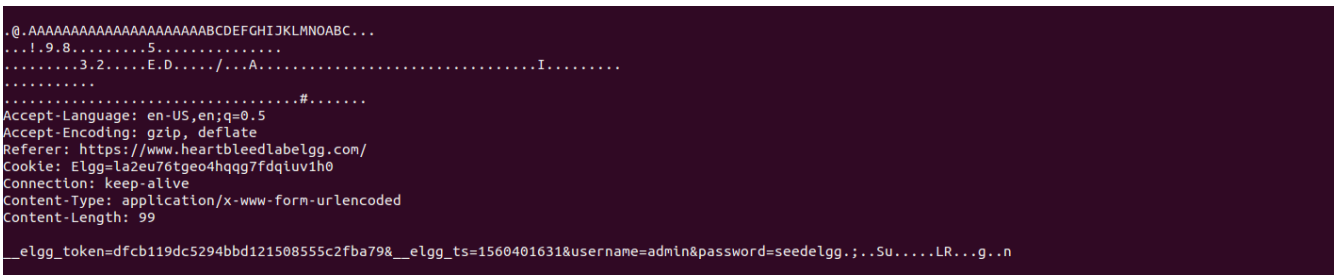


```
./attack.py www.heartbleedlabelgg.com
```

Try and see whether you can get the following information from the target server.

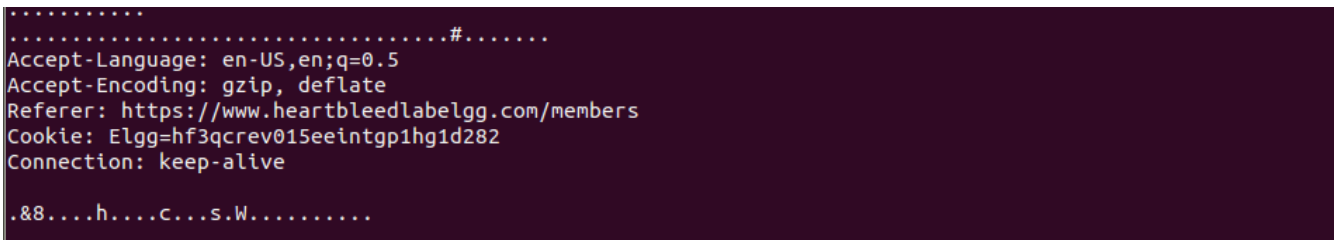
- User name and password.

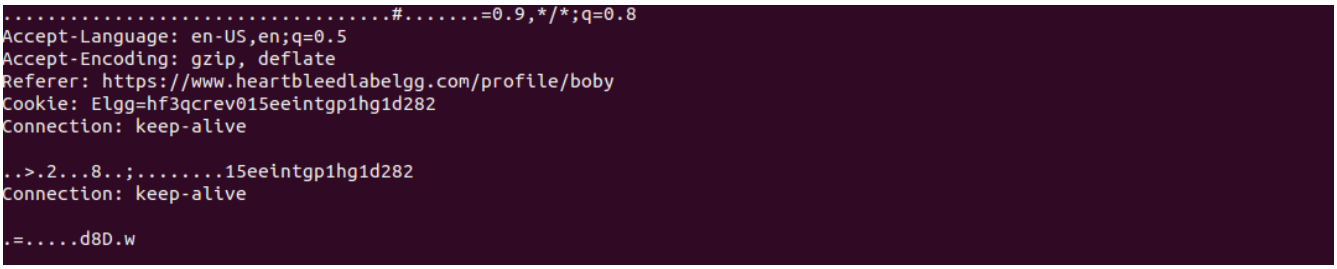
> the username is admin and password is seedelgg from the last line of the screenshot

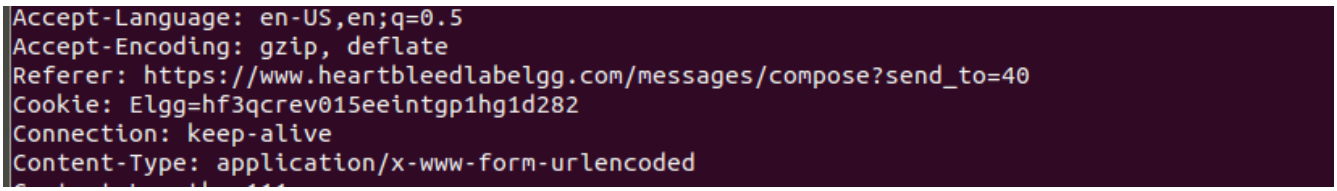


- User's activity (what the user has done)

> From the referer field, the user has visited `members`, `body` and `message` respectively, we can know that the user viewd somebody's profile and sent a message







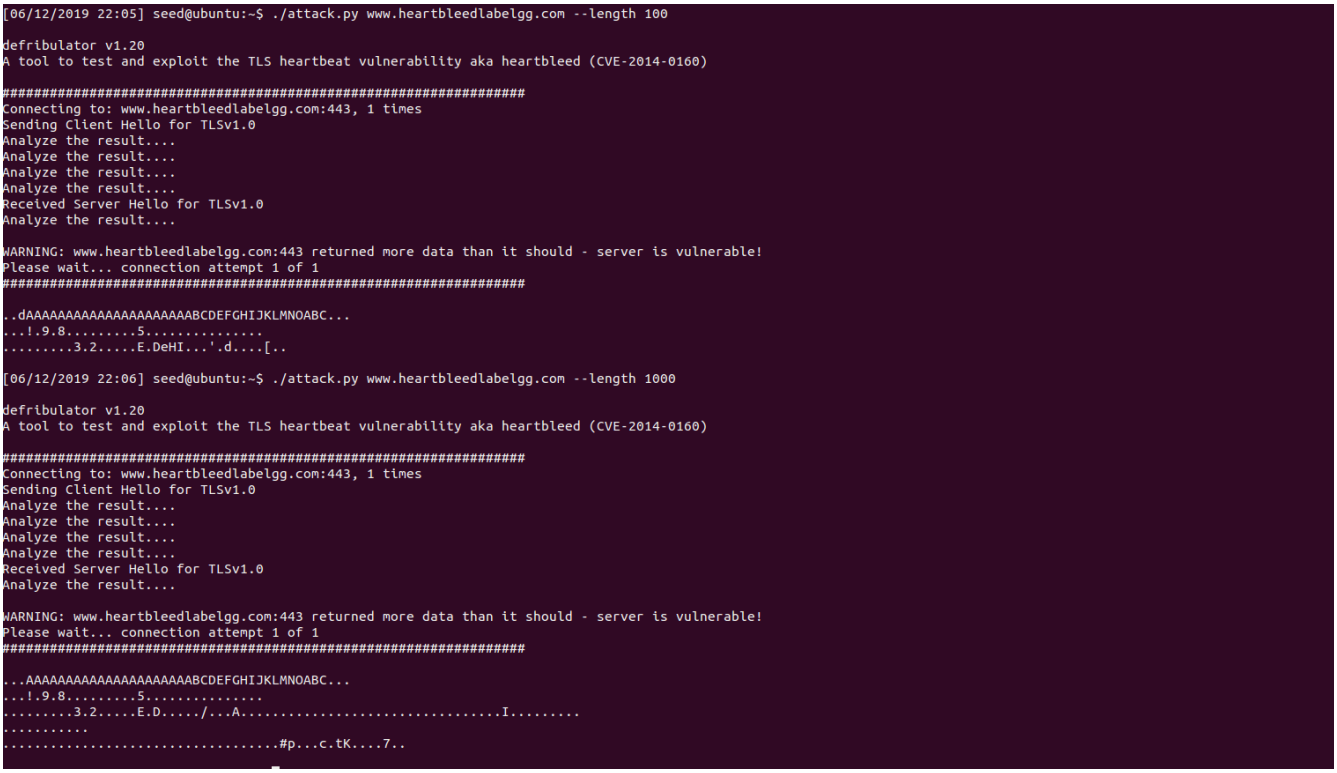
- The exact content of the private message.

> The message and be seen from the last line of the screenshot
>
> It tells the subject is `secret` and body is `secret` as well

Task 2: Find the Cause of the Heartbleed Vulnerability

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

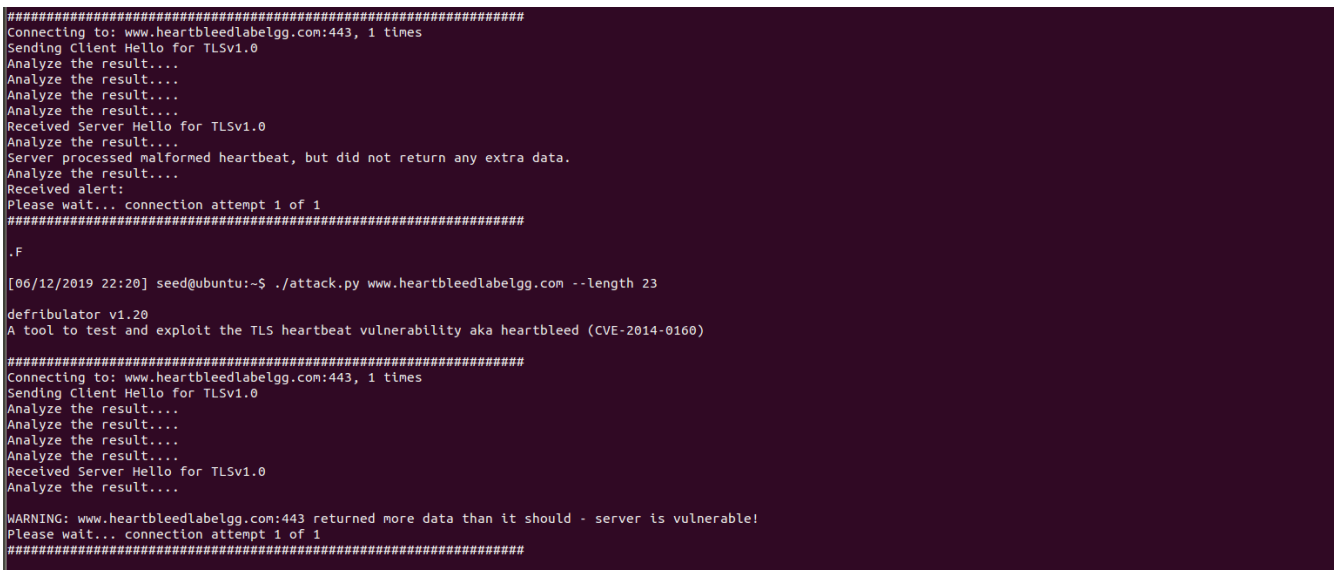
> The length of the response would change accordingly. If we set the length too low, the attack would leak no extra information about the server other than `.F`



- boundary value for the input length variable

```
./attack.py www.heartbleedlabelgg.com --length 22
```

> From the screenshot below, when the length field is 22, the packet is benign. And when we set length to 23, it is not. So 22 is a boudary value for this field.



Task 3: Countermeasure and Bug Fix

packet structure

```
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;
```

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

```
$ sudo apt-get update
$ sudo apt-get install openssl
```

> After updating `openssl` on the server, the attack would not succeed since the updated version has applied a boundary check. No other information than `.F` would be returned

- Please point out the problem from the code

> It used the user-specified `payload` field to dertermine the length of the response packet. And this could leak much information of the server.

- describe how you can fix the problem

> We could check whether the length would outweight the effective length of the packet. This works like a boudary check

```
if(1+2+payload+16 > s->s3->rrec.length)
    return 0
```

- Alice thinks the fundamental cause is missing the boundary checking during the buffer copy

> Yes, the program used the alleged length of the packet but not check it. This is why our attack would succeed

- Bob thinks the cause is missing the user input validation

> Yes, in this case, user could specify the length of the packet. There is a need to validate the input from the user

- Eva thinks that we can just delete the length value from the packet to solve everything.

> It is not true. We have padding in our packet. If we cannot specify the length, it would be very hard to do the decoding.

- Why 22 is a threshhold here

> In the code of `attack.py`

```
if len(pay) > 0x29:
   if firstrun or opts.verbose:
       print '\nWARNING: ' + targ + ':' + str(opts.port) + ' returned more data than it
should - server is vulnerable!'
```

> if the total length is more than 41, there will be a warning

> So if we set our payload length to `total-type-length-padding = 41-1-2-16=22`, the warning will be reported