HeartBleed

Meraj Ahmed

2019MCS2565

1 SSL / TLS

SSL stands for the Secure Socket Layer. It is also known as TLS, short for 'Transfer Layer Security.' SSL certificates are widely used in websites, mobile apps, emails, fax, messaging, etc.

When we access a website, communication takes place between the web browser of our PC or mobile device and the web server of the website. Information is then transferred from both sides. An SSL certificate protects the information transferred between both. This is compelling from a security and privacy point of view

1.1 SSL Protocol Stack

1. **SSL record protocol** SSL Record provide two services to SSL connection. (i) Confidentiality, (ii) Message Integrity
2. **Handshake Protocol** Handshake Protocol is used to establish sessions. This protocol allow client and server to authenticate each other by sending a series of messages to each other. Handshake protocol uses four phases to complete its cycle.
   (a) **Phase-1** In Phase-1 both Client and Server send hello-packets to each other. In this IP session, cipher suite and protocol version are exchanged for security purpose.
   (b) **Phase-2** Server send his certificate and Server-key-exchange. Server end the phase-2 by sending Server-hello-end packet.
   (c) **Phase-3** In this phase Client reply to the server by sending his certificate and Client-exchange-key.
   (d) **Phase-4** In Phase-4 Change-cipher suite occurred and after this Handshake Protocol ends.
3. **Change-cipher Protocol** This protocol uses SSL record protocol. Unless Handshake Protocol is completed, the SSL record Output will be in pending state. After handshake protocol the Pending state is converted into Current state.

4. **Alert Protocol** This protocol is used to convey SSL-related alerts to the peer entity. Level is further classified into two parts: (i) Warning, (ii) Fatal Error

2 Heart Bleed

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.

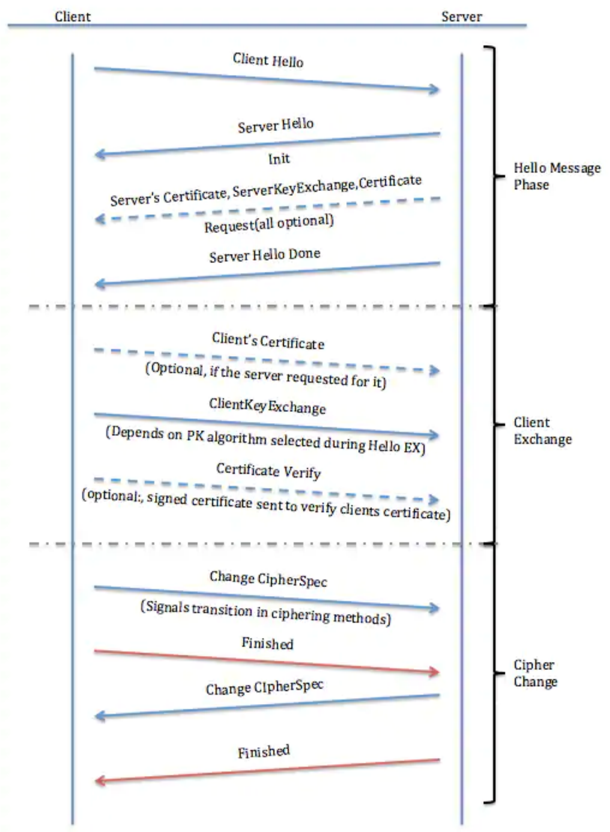

Client
Server
Client Hello
Server Hello
Init
Server's Certificate, ServerKeyExchange,Certificate
Request(all optional)
Server Hello Done
Hello Message Phase
Client's Certificate
(Optional, if the server requested for it)
ClientKeyExchange
(Depends on PK algorithm selected during Hello EX)
Certificate Verify
(optional:, signed certificate sent to verify clients certificate)
Client Exchange
Change CipherSpec
(Signals transition in ciphering methods)
Finished
Change CipherSpec
Finished
Cipher Change

Figure 1: SSL client-Server packet transaction

2.1 Heart beat

Heartbeat is an echo functionality where either side (client or server) requests that a number of bytes of data that it sends to the other side be echoed back. The idea appears to be that this can be used as a keep-alive feature, with the echo functionality presumably meant to allow verifying that both ends continue to correctly handle encryption and decryption. The problem, of course, is that until the recent patch, OpenSSL did not guard against sending back more data than was provided in the first place.

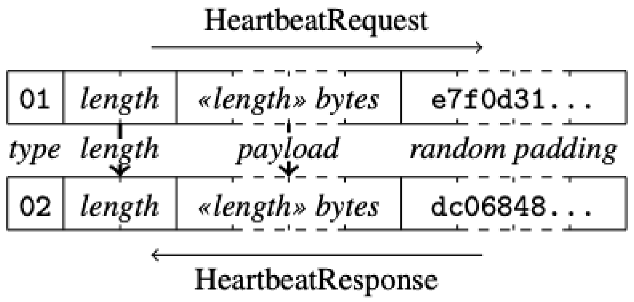


Figure 2: Heartbeat Protocol. Heartbeat requests include user data and random padding. The receiving peer responds by echoing back the data in the initial request along with its own padding

2.2 Vulnerability

The Heartbleed bug is a programming error in the versions 1.0.1 to 1.0.1f of the open-source OpenSSL cryptography library. This critical security gap makes it possible to read encrypted data of clients and servers connected via TLS. It was fixed in version 1.0.1g on april 7th, 2014.

2.3 Source of vulnerability

The coding mistake that caused Heartbleed can be traced to a single line of code:

```
memcpy(bp, pl, payload);
```

memcpy() is the command that copies data. bp is the place it's copying it to, pl is where it's being copied from, and payload is the length of the data being copied. The problem is that there's never any attempt to check if the amount of data in pl is equal to the value given of payload.

2.4 Fix

```
* Read type and payload length first */

if (1 + 2 + 16 > s->s3->relent)

return 0;

/* silently discard */

hbtype = *p++;

n2s(p, payload);

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

return 0;

/* silently discard per RFC 6520 sec. 4 */

pl = p;
```

The above patch is applied to fix the heartbleed vulnerability. The first part of this code makes sure that the heartbeat request isn't 0 KB, which can cause problems. The second part makes sure the request is actually as long as it says it is.

2.5 Exploitation

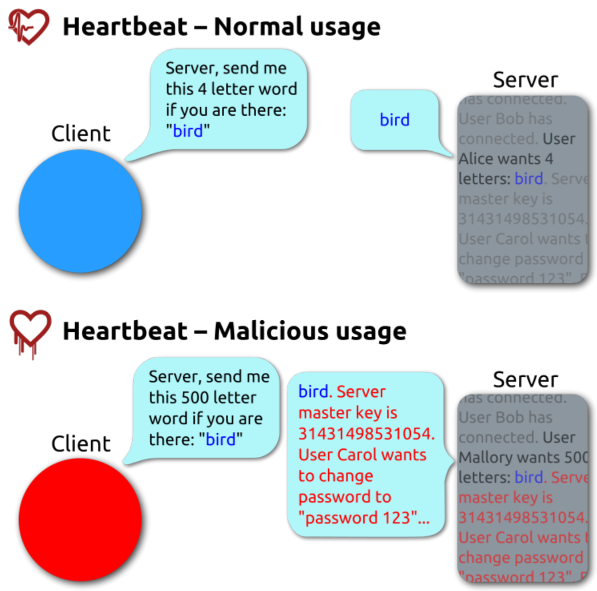

Heartbeat – Normal usage
Client
Server, send me this 4 letter word if you are there: "bird"
bird
Server
User Bob has connected. User Alice wants 4 letters: bird. Server master key is 31431498531054. User Carol wants to change password to "password 123".
Heartbeat – Malicious usage
Client
Server, send me this 500 letter word if you are there: "bird"
bird. Server master key is 31431498531054. User Carol wants to change password to "password 123"...
Server
User Bob has connected. User Mallory wants 500 letters: bird. Server master key is 31431498531054. User Carol wants to change password to "password 123".

Figure 3: Heart bleed exploitation

Let's say a client sends a Heartbeat request to the server saying send me the four letter word 'bird'. This message in encrypted form received by the server and then server acknowledges the request by sending back the exact same encrypted piece of data i.e. 'bird'. This Heartbeat message request includes information about its own length. Now in the malicious usage scenario the client sends the request by saying send me the word 'bird' consisting of 500 letters. This let the server to store more in memory buffer based on the reported length of the requested message and sends him back more information present on the web server. The attacker can perform this attack many times to extract the useful information including login credentials.
In my code after handshake with I will try to send heartbeat with excess length and server will reply with excess data in its heartbeat payload. Excess data server gets from the its memory heap.

3 Code

My code for heart bleed vulnerabilities consists of 3 files

1. **SSLPacket.py** It have python class associated with the SSL packet. It defines the structure of the SSL packet and the field associated with packet. It also defined the possible values of the some field in the packet
2. **Server.py** This files consist of the functionalities of SSL server. The main functionality used in this demo is the heartbeat. Server responds with the clients heartbeat request.
3. **Client.py** This file consists of file associated the client. For this demo clients main functionality is the sending heartbeat request to the server.

3.1 How to run

First run server.py file to start the server then run the client

```
python3 server.py
python3 client.py
```

The terminal output of the client will show the what is happening in the background.

3.2 How it works

3.2.1 Handshake

First client and server will establish connection by handshake protocol of SSL. I have implemented the simplified version of handshake. First client sends the hello packet to server and waits for server response. Server responds to hello requests by sending hello packet, certificate and ends hello response by sending HelloDone packet.After exchanging hello packet, client sends its own certificate. After this it sends the FIN packet to indicates the completion of handshake from client side and it waits for FIN packet from server. After receiving FIN packet from server, handshake process complete. Now client and server can exchange the application data.

```
def handshake(conn):
    sendHelloPacket(conn)
    recieveHelloPacket(conn)
    recieveCertificate(conn)
    recieveServerHelloDone(conn)
    sendCertificate(conn)
    sendFin(conn)
    recieveFin(conn)
```

3.2.2 HeartBleed attack

We exploit the heartbleed in the attack function of client. It contains two part:

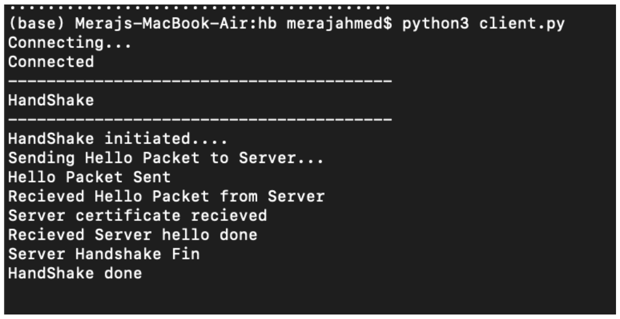
1. In first part client send the normal heart beat with payload 'bird' and length 4. Server echo back with the same message.
2. In second part, we send the malicious heart beat packet. In this packet we put small payload 'bird' but sets the length of payload large like 100 or 200 instead of 4. When server receives this heartbeat packet. Server append the extra data from its memory to meet the length of heartbeat request packet. This extra information from the memory may contain the very sensitive information like session key, cookies or password. Attacker can send the multiple heartbleed packet to get the more information from the server.

```
def attack(conn):
    print("Testing Normal Heart beat...")
    hb_data = "bird"
    length = 4
    print("Sending Heart Beat....")
    sendHeartBeat(conn, hb_data, length)
    response = recieveHeartBeatResponse(conn)
    print("\nServer response to HeartBeat:")
    print(response)

    print("Exploiting Heart Bleed Vulnerability")
    hb_data = "bird"
    length = random.randint(100, 200)
    print("Sending Heart Beat....")
    sendHeartBeat(conn, hb_data, length)
    response = recieveHeartBeatResponse(conn)
    print("\nServer response to HeartBeat")
    print(response)
```

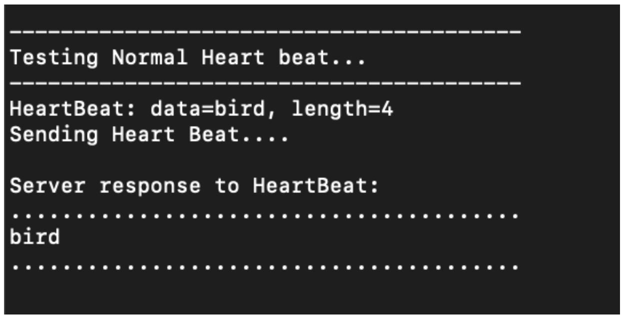
3.3 Screenshots

3.3.1 Client



```
.......................................
(base) Merajs-MacBook-Air:hb merajahmed$ python3 client.py
Connecting...
Connected
---------------------------------------
HandShake
---------------------------------------
HandShake initiated....
Sending Hello Packet to Server...
Hello Packet Sent
Recieved Hello Packet from Server
Server certificate recieved
Recieved Server hello done
Server Handshake Fin
HandShake done
```

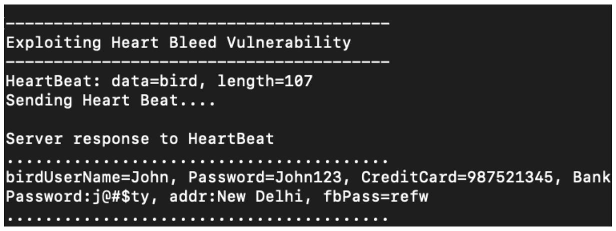
Figure 4: Client initiate handshake



```
---------------------------------------
Testing Normal Heart beat...
---------------------------------------
HeartBeat: data=bird, length=4
Sending Heart Beat....

Server response to HeartBeat:
.......................................
bird
.......................................
```

Figure 5: Testing normal heart beat

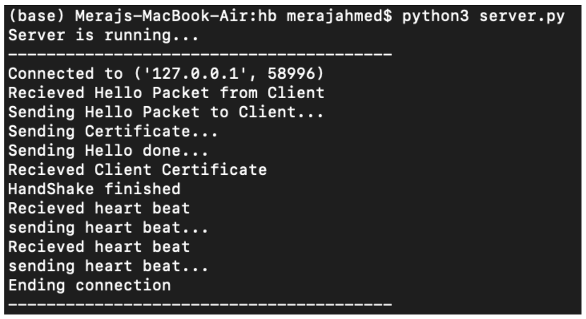


```
---------------------------------------
Exploiting Heart Bleed Vulnerability
---------------------------------------
HeartBeat: data=bird, length=107
Sending Heart Beat....

Server response to HeartBeat
.......................................
birdUserName=John, Password=John123, CreditCard=987521345, Bank
Password:j@#$ty, addr:New Delhi, fbPass=refw
.......................................
```

Figure 6: Exploiting heart bleed to get sensitive information from server

3.3.2 Server



```
(base) Merajs-MacBook-Air:hb merajahmed$ python3 server.py
Server is running...
----------------------------------------
Connected to ('127.0.0.1', 58996)
Recieved Hello Packet from Client
Sending Hello Packet to Client...
Sending Certificate...
Sending Hello done...
Recieved Client Certificate
HandShake finished
Recieved heart beat
sending heart beat...
Recieved heart beat
sending heart beat...
Ending connection
----------------------------------------
```

Figure 7: Server response on the client request

4 References

1. https://www.geeksforgeeks.org/secure-socket-layer-ssl/
2. https://heartbleed.com/
3. https://en.wikipedia.org/wiki/Heartbleed
4. https://null-byte.wonderhowto.com/how-to/hack-like-pro-hacking-heartbleed-vulnerability-0154708/