

Assignment 3 Report COL759 : Heartbleed Vulnerability OpenSSL

2019MCS2574

Vivek Singh

1 What is OpenSSL

OpenSSL is a open source implementation of SSL/TLS protocol. The core program is written in C programming language. OpenSSL is a software library for applications that secure communications over computer networks against eavesdropping or need to identify the party at the other end. It is widely used by Internet servers, including the majority of HTTPS websites.

2 Working of OpenSSL

Since OpenSSL is based on SSL/TSL protocol it works similarly.

- The SSL/TSL protocol allows the server and client to:
 - Authenticate one another.
 - Negotiate and finalize an encryption and MAC algorithm
 - Finalize and share the cryptographic keys to protect and securely transmit payload data.
- Once Client and Server makes a connection, the client request for a secure connection.
- After receiving the request server chooses a most secure option that is compatible with both client and server, and then sends a security certificate signed with the server's public key.
- The client verifies the certificate and generates a secret key to send to the server, encrypted with the server's public key.
- The client and server use the secret key to generate pair of symmetric keys (or two pairs of public-private keys), and communication commences securely.

3 Heartbleed Vulnerability

The OpenSSL provides secure transmission of data from client and server by using cryptographic methods. But in OpenSSL implementation of v1.01 and some other Beta versions a significant vulnerability was discovered called HeartBleed.

By exploiting this vulnerability a attacker may gather and reveal information upto 64K of memory data in stored in server cache.

Hence by reading the server memory attacker can gain access to sensitive data such as login Ids, passwords and even senders private keys. Knowledge of private key will compromise the whole confidentiality of the data and the attacker can also perform man-in-middle attack.

4 Why and How of HeartBleed Vulnerability

One important part of the SSL/TSL protocols is what's called a heartbeat. When client and server are connected to one another using the secure connection, then to let each other know that they are still alive and connected client send the server a special request called **Heartbeat request**. The server responds to the client using the same message back.

The heartbeat request contains a payload message along with a size parameter passed which indicates the size of payload.

When server receives the heartbeat request it copies the payload of the given size present in the memory buffer and send it back to the client which requested the message to check the connection.

Heartbleed code: memcpy(bp, pl, payload);

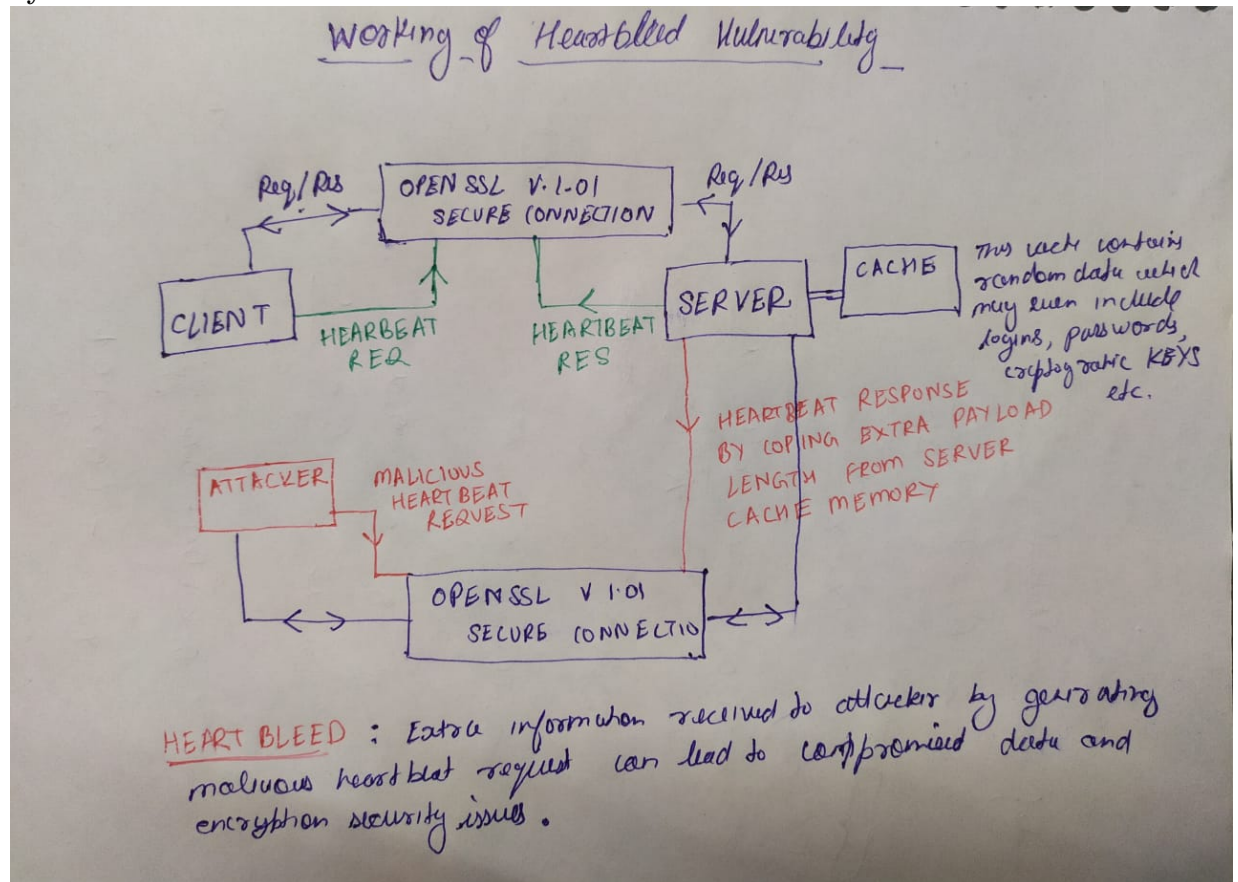
memcpy() is the command that copies data. bp is the place it's copying it to, pl is where it's being copied from, and payload is the length of the data being copied. The problem is that there's never any attempt to check if the amount of data in pl is equal to the value given of payload. This part of code written in C caused the vulnerability in OpenSSL v1.01.

Example of exploiting the vulnerability.

- As an attacker you will send a heartbeat request to the server.
- Let the request contains 10K byte of information
- The request is asking for 60K byte of information in payload length.
- This difference of 50K byte of information is filled in the memory buffer using the server cache. This cache can contain garbage values, login ids, last request or even cryptographic keys.
- So the OpenSSL memcpy() will add all the extra padding in the heartbeat response causing major information loss and compromise the security and confidentiality of the data.

5 Flow-Diagram of simulation

Figure 1: Flow diagram showing the exploitation of heartbeat request by attacker



6 Simulation of heart-Bleed vulnerability

- The simulation is based on following assumptions
 - Connection between server and other computer is secured by OpenSSL v1.01.
 - A server is connected to client and communication is going on a secure channel.
 - After the communication is closed between client and server, attacker also gets connected to server and send carefully crafted malicious HEART-BEAT request to server.
- Server contains a cache memory which contains garbage values as well as recent information of the client server communication, request and responses by server to a client.

Figure 2: Client is connected to server and communicating over OpenSSL secure connection.



```
vivek@vivek-X556UF: ~/Desktop/IITDcourse/S
File Edit View Search Terminal Tabs Help
vivek@vivek-X556UF: ~/Desktop/IITDcourse/Sem2/CryptoSecurity/Assignment3 x
vivek@vivek-X556UF:~/Desktop/IITDcourse/Sem2/CryptoSecurity/Assignment3$ python client.py
Client is now connected to server using simulation based openSSL


OpenSSL provides secure communications over computer networks against eavesdropping.

Cleint will automatically send a HEARTBEAT req to server every few seconds

-> login: mcs192574
Received from server: OK
-> pass: 12345
Received from server: OK
-> hi i am vivek singh
Received from server: OK
-> this is assign3 col 759
Received from server: OK
-> a samll simultaion based on OpenSSL heartbleed vulnerabilty
Received from server: OK
-> bye|
```

- Client send server heartbeat request regularly and periodically to insure that the connection is maintained.
- Once server receives the heartbeat request it replies with appropriate heartbeat response according to the payload information received from in request parameter.

Figure 3: Server side of the connection where it communicates with the client and also ensures safe connection by responding to heartbeat request.



```
vivek@vivek-X556UF: ~/Desktop/IITDcourse/Sem2/CryptoSecurity/Assignment3
File Edit View Search Terminal Tabs Help
vivek@vivek-X556UF: ~/Desktop/IITDcourse/Sem2/CryptoSecurity/Assignment3
vivek@vivek-X556UF:~/Desktop/IITDcourse/Sem2/CryptoSecurity/Assignment3$ python server.py
Welcome to server

All request and response generated are protected using simulation based OpenSSL
Simulation is based on HEARTBLEED Bug which is a serious vulnerability in the popular OpenSSL

Connection from: ('127.0.0.1', 40080)
HEARTBEAT req from client received: alive
from connected user: login: mcs192574
from connected user: pass: 12345
from connected user: hi i am vivek singh
HEARTBEAT req from client received: alive
from connected user: this is assign3 col 759
HEARTBEAT req from client received: alive
HEARTBEAT req from client received: alive
HEARTBEAT req from client received: alive
from connected user: a samll simultaion based on OpenSSL heartbleed vulnerabilty
HEARTBEAT req from client received: alive
from connected user: bye
```

- We can see various heartbeat request generated by client received on server side.
- The client side end the communication by sending a bye text to server.

- Now attacker gain the communication channel with server to generate carefully crafted heartbeat request to perform a heartbleed attack.

Figure 4: **Server side of the connection where it communicates with the attacker and respond to the malicious request generated by attacker and sends sensitive information stored in its cache.**

```
vivek@vivek-X556UF: ~/Desktop/IITDcourse/Sem2/CryptoSecurity/Assignment3
vivek@vivek-X556UF:~/Desktop/IITDcourse/Sem2/CryptoSecurity/Assignment3$ python server.py
Welcome to server

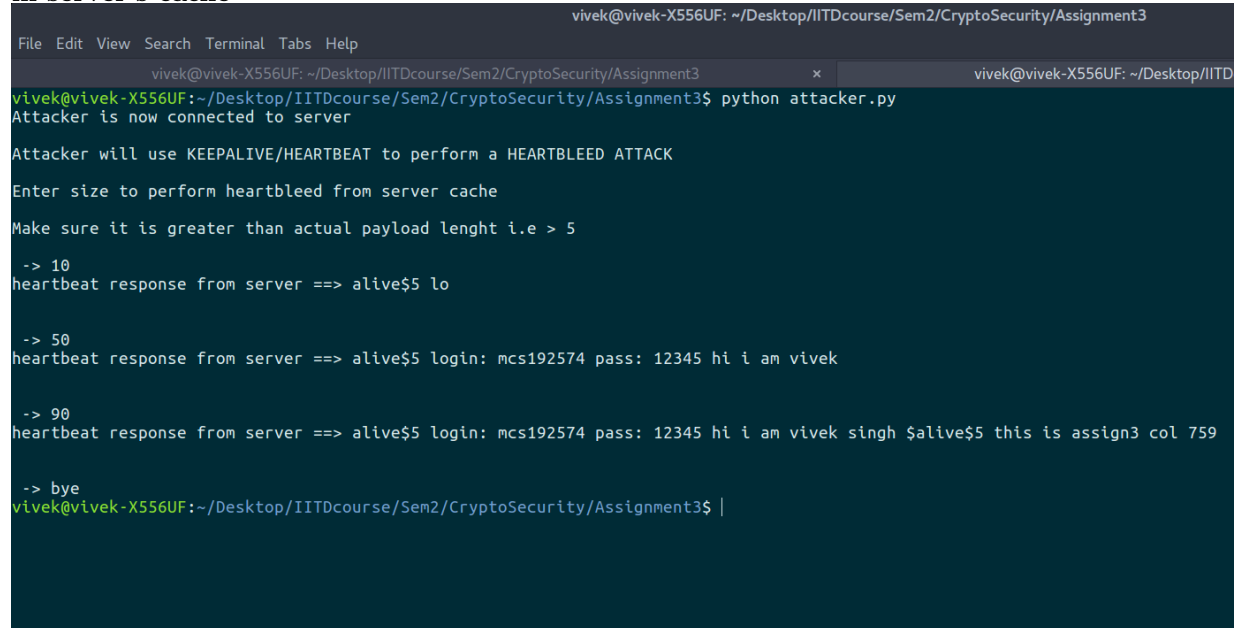
All request and response generated are protected using simulation based OpenSSL
Simulation is based on HEARTBLEED Bug which is a serious vulnerability in the popular OpenSSL

Connection from: ('127.0.0.1', 40080)
HEARTBEAT req from client received: alive
from connected user: login: mcs192574
from connected user: pass: 12345
from connected user: hi i am vivek singh
HEARTBEAT req from client received: alive
from connected user: this is assign3 col 759
HEARTBEAT req from client received: alive
HEARTBEAT req from client received: alive
HEARTBEAT req from client received: alive
from connected user: a samll simultaion based on OpenSSL heartbleed vulnerabilty
HEARTBEAT req from client received: alive
from connected user: bye

Connection from: ('127.0.0.1', 40084)
HEARTBEAT req from client received: alive$
HEARTBEAT req from client received: alive$
HEARTBEAT req from client received: alive$
```

- Here we can see that server when connected to new user(attacker) has served 3 heartbeat request and responded to the request.
- These response might contain information extracted from its cache.

Figure 5: **Attacker sending lager payload length to capture information in server's cache**



```
vivek@vivek-X556UF: ~/Desktop/IITDcourse/Sem2/CryptoSecurity/Assignment3
File Edit View Search Terminal Tabs Help
vivek@vivek-X556UF:~/Desktop/IITDcourse/Sem2/CryptoSecurity/Assignment3$ python attacker.py
Attacker is now connected to server
Attacker will use KEEPALIVE/HEARTBEAT to perform a HEARTBLEED ATTACK
Enter size to perform heartbleed from server cache
Make sure it is greater than actual payload lenght i.e > 5
-> 10
heartbeat response from server ==> alive$5 lo
-> 50
heartbeat response from server ==> alive$5 login: mcs192574 pass: 12345 hi i am vivek
-> 90
heartbeat response from server ==> alive$5 login: mcs192574 pass: 12345 hi i am vivek singh $alive$5 this is assign3 col 759
-> bye
vivek@vivek-X556UF:~/Desktop/IITDcourse/Sem2/CryptoSecurity/Assignment3$ |
```

- Here we can see that attacker is able to extract sensitive information from server cache.
- It includes recent communication along with login ids and password exchanged by client and server.
- A README is provided in the code to help run the simulation. Make sure to send ****bye**** mssg by client and end connection with it before starting the attacker script.
- bye mssg from both client and attacker ensures that client/attacker safely gets disconnected from server.
- Hence attacker was able to gain access to sensitive data using OpenSSL heartbeat request resulting in major security flaw.
- Later updates of OpenSSL have fixed this issue by keeping a check on memcpy() function to detect any malicious request.