Heartbleed Attacks Implementation and Vulnerability

Shashank Kyatam , Abdullah Alhayajneh
Computer Science Department
New York Institute of Technology
Old Westbury, NY, USA
aalhaya@nyit.edu

Thaier Hayajneh
Computer and Information Science Department
Fordham University
New York, NY, USA
thayajneh@fordham.edu

Abstract— **Several vulnerabilities were detected in the open SSL connection versions 1.0.1 and 1.0.1f. Usually, in the previous versions of SSL/TLS, once an SSL connection is established between a client and a server, the connection will stay until the client or server is idle for a certain amount of time, after which the connection will be dropped. The idea of keeping the session connected was proposed in 2012. The initial idea introduced Heartbeat Messages that are indirectly called "keep alive packets". These "keep alive packets" or "heartbeat packets" are transmitted in between client and server when the SSL session is ideal for a certain amount of time. Regarding "keep alive packets" or "heartbeat packets" mechanisms, these packets are stored in the same memory in which most sensitive information of the client and server is stored. When it is one of the peer's turn to return the heartbeat message, that peer takes the heartbeat packet saved in its random memory location, which is sent by the other peer, and returns it to the other peer to acknowledge the live session. However, the hackers are able to craft a similar Heartbeat Message in a way that makes the peers store it in the same memory location where the sensitive data is stored. Then it returns back the sensitive data along with the crafted heartbeat message sent by the hackers. In this paper, we studied and implemented the heartbleed attack. We also discussed mitigation solutions for this vulnerability.**

Index Terms— **Heartbleed, Heartbeat, SSL, TLS, Reverse Heartbleed.**

I. Introduction

Heartbleed is a vulnerability that exists in open SSL connection and the official reference of this bug was CVE-2014-0160 where CVE stands for Common Vulnerability and Exposures. This reference determines the standard for information security vulnerability whose name is maintained by MITRE. This vulnerability was identified in April 2014. It is not a virus or some malicious code which a hacker can implement and exploit. This is a vulnerability which was pre-existing while designing SSL version 1.0.1 and 1.0.1f. However, it was not identified until 2014 by a team of security engineers from Google named Riku, Antti and Matti as well as Neel Mehta. It all started with the implementation of heartbeat messages.

SSL (Secure Socket Layer) is an encryption scheme which is widely used in VPN connections. Because it belongs to the Transport Layer in the OSI model (Open Systems Interconnection model), it is also called Transport Layer Security or TLS. TLS and SSL are one and the same. This encryption was introduced in 1984 and has continued developing since. As a phase of development in open SSL standards, they introduced Heartbeat on March 14th 2012, and the SSL version was updated to version 1.0.1 and 1.0.1f.

SSL connection is an End-to-End encryption style. An encrypted tunnel will be formed in between the client's PC and the server. All the data will be encrypted in between both the peers (client and server). To form this, a negotiation session will be held between these two peers. This session will be divided into four parts for better understanding. These four sessions will discuss how the encryption has to take place.

In the first step, a hello message is sent from client to server in order to establish an SSL connection with the server. That packet contains the key which the client supports (RSA, Diffie hellman, DSA), the cipher that it can use (RC4, Triple DES, AES) and a hash function which it wants to use (HMAC-MD5, HMAC-SHA). It also contains the version of SSL that it is using along with a random number. The server recognizes that the client was intended to have an SSL connection immediately after receiving the hello packet. Then, the server will negotiate with the client regarding the keys, cipher used and hash function it uses. Then they come to a conclusion that, both the server and client use the same scheme and version.

In the second step, a certificate will be sent from server to client for authentication and trust ability. That certificate contains attributes like serial number, valid from and to, public key, issuer's identification, site data, company, location, address, etc… Then the public key of the client will also be sent to the server.

In the third step, the client tells the server to start the encryption. In this part, both the client and server will compute a session key using the credentials obtained from the previous transactions and using some functions. A client and server key exchange will take place and change the mode to cipher-spec mode in which the client tells the server to start the encryption and the server acknowledges it.

In the final step, the server tells the client to start the encryption and then the encryption and communication will initiate. The server sends a notification to the client to start the encryption, and the authentication procedure of client to server will also be

done by validating the credentials (username and password) of the client. These credentials are in an encrypted format and no one can see the username and password in plaintext.

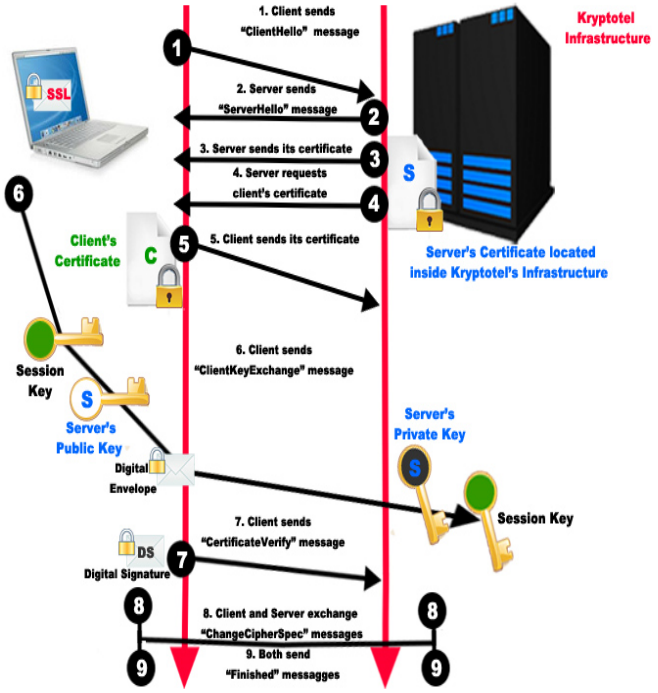

Figure 1. A simple client/server SSL connection steps

This is how the negotiation procedure will go on. However, when the certification is sent from the server to the client, the client has to identify the server by checking the certificate with the Certification Authority or Certification Authority Manager inside its memory. Finally, it confirms whether this certificate is valid and trusted.

But there is no heartbleed vulnerability in this procedure. This vulnerability comes after the connection is established. When the SSL connection is formed, both parties trust each other 100%. In the procedure of communication between two peers, if both of the peers are performing ideally for a certain amount of time, then a message called Heartbeat will be flowing between the two peers to keep the session alive, so they need not renegotiate with each other and waste time and processing.

The remaining sections on this paper are organized as follows. Sections II and III introduce the Heartbeat and its packet and mechanism. Section IV explains the heartbeat attack. The reverse heartbeat is introduced in section V. The detailed implementation of heartbeat is illustrated in Section VI. Section VII discusses the mitigation solution for heartbeat.

II. Heartbeat

Heartbeat was introduced to overcome a few problems like renegotiation and authentication. Once a connection was dropped, then the two peers needed to renegotiate with public and private keys in an asymmetric encryption scheme that took a significant amount of processing power, which might have caused Denial of Service. Moreover, the user needed to provide his credentials once again. To overcome that, Heartbeat Messages were implemented to keep the session alive and avoid renegotiation and authentication.

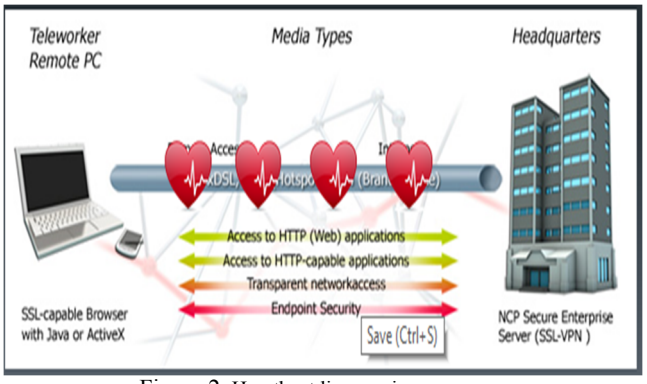

Figure 2. Heartbeat live session messages

III. Heartbeat Packet and Mechanism

In order to keep the session alive, Heartbeat Messages will be produced and sent to the server. The packet sent from the client will have three sections, and each section is the payload which is a random set of strings produced by the client and adds it in the payload area. Then, the size of the payload is also attached to the payload. Along with that, an acknowledgement packet is also attached which will request the server to send an acknowledgement packet in return to this packet to keep the session alive. Then the server will receive it and store it in the random memory location in which sensitive data of client and server is stored. Usually, a server will store such types of sensitive information very randomly into a stack of its memory. And that heartbeat packet is added to the stack of its memory. After that, the server will send the data to the client as an acknowledgement to the heartbeat message.

The server looks at the size of the packet stored inside the stack of memory, and returns the starting part of memory of the packet's size stored. Therefore, the server will send the packet to the client of what the client had sent in the previous heartbeat packet to confirm what it had received. Then the process continues until the session ends. These packets are not audited or inspected because the server and client trust each other to function properly.

As per the example stated previously, "Shashank" is something which is considered a payload and has 8 characters in it. The client sends the whole packet with payload (Shashank), size (8), ACK required to the server. Then the server will store these 8Bytes of messages into the stake of its memory and look at the size of the packet sent which is prescribed in the second part of the packet. Then it digs its stack of memory till it reaches 8 Bytes and replays it back to the client as Shashank and 8. The client will receive the packet and check whether that is what it had sent. Moreover, it repeats the process to keep the session alive.

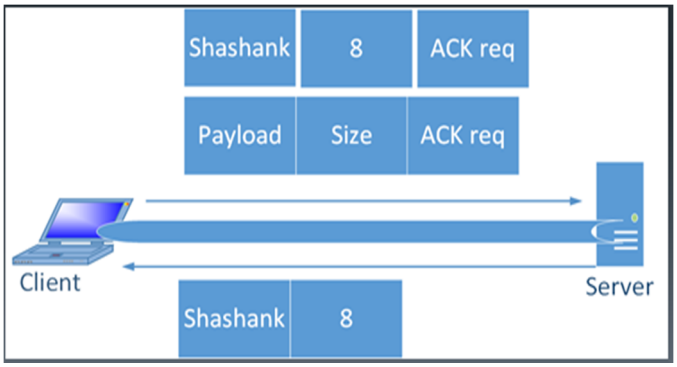

Figure 3. 8 bytes payload Shashank process

IV. Heartbeat Attack

Now that the functionality of a heartbeat attack has been established. the focus will shift to how the vulnerability is formed and how is it exploited. In the process of the heartbeat, many numbers of clients are connected to the server with an SSL connection. They all get authenticated to the server by providing their credentials along with their public keys to the server. Now if a hacker gets authenticated with the same server, he or she will stay ideal for some time to produce Heartbeat Messages to the server and once these are produced, he or she will craft a packet in a way that, the size of the packet is denoted much larger than the packet's payload is. And the server receives it and stores it in its memory. It will reply back by looking at the number which is mentioned as the size of the packet, and irrespective of the payload size sent to the server, the server will dig the number of bytes from its own memory in which the Heartbeat Message as well as some sensitive information will be packetized and sent to the client.

According to the example, the hacker will send a packet which is crafted with a payload "Shashank" and so the true size of that packet should be 8 bytes. Instead of 8 bytes, the attacker will change the number to 65527 bytes. It is sent to the server and stored in the server's memory. The server will consider only the size of the packet mentioned in the heartbeat packet from the client, and ignore the original size of the payload sent. And so, the server now believes that the size of the packet that the client has sent it is the 65527 bytes already stored in its memory. Then it thinks that it needs to give the client 65527 bytes of data and it digs out 65527 bytes of data from its own memory and sends it to the client.

The best thing is that the client will not need to decrypt the packet sent from the server. All the text that is sent by the server will be decrypted automatically when it reaches the client's PC. This process doesn't stop. Then, it's now the clients turn to resend the same packet to the server. Then the attacker will replay the same packet once again to the server, so the server considers the same "Size of the packet as mentioned in the heartbeat packet from client" and sends back 65527 bytes of memory from its own memory which may contain sensitive information like usernames and passwords of the users, public keys, session identifiers, partial information regarding X.509 certificates, etc…

The packet which is coming to the client as a reply to the Heartbeat Message from the server might contain only a portion of the information of these attributes mentioned. But, by repeating this several times, meaningful and confidential information can easily be obtained. Putting together all the data, public keys and session keys can also be made. So, this can lead to a numerous numbers of attacks. It can lead to a huge disaster and might completely collapse the security of an organization. The worst part of this attack is that it's passive. As the client and server trust each other more, the heartbeat messages are neither audited nor inspected, so no one can be caught. This is good for attackers and hard for forensic experts. Because of the functionality, the name "Heartbleed" was suggested; it makes the Heartbeat Messages bleed.

This doesn't end here. Normally, the procedure of heartbeat will be that the client sends Heartbeat Messages to the server and the server will send back that amount of data mentioned in the packet. Furthermore, when the client receives it, that packet is stored in the user's memory in which confidential data like cookies, saved username password, private keys, private crypto systems, etc… are present. When the client looks at the packet, and it indicates the size of the packet attached to the payload, the client will respond to the server by digging its own memory of the number of bytes mentioned in the packet and replay it back to the server. Now, the client is also vulnerable in this reverse technique of Heartbleed.

V. Reverse Heartbleed

Now that we've described how the vulnerability will take place on the server's side and exploited from the client's side, it's time to investigate the reverse condition. The Reverse Heartbleed Technique will investigate how an attack can be launched on the client's side from the server's side. There are millions of clients in the world using this vulnerable SSL connection. So, the target can be achieved when we try to grab the client's connection to an attacker server. It can be possible in multiple ways using many known methods like phishing, spam messaging, etc. If a client's interest is grabbed, then the client will establish a secured channel using SSL to the server which the attacker got access on.

Then the attacker on the server will send a heartbeat packet as a reply to the client by manipulating the size of the packet as the size column says, more than the number of characters in the payload. Next, the client will dig its memory and send back confidential data as well as the payload sent by the server. And this procedure will be repeated multiple times in order to accumulate all the private keys and the private cookies in stored in the user's memory. Then a new attack begins from this point.

In the example mentioned above, the server sends a heartbeat acknowledgement with morphed credentials of the packet that is 65527 bytes but with an actual size of 8 bytes. Then the client looks at 65527 and sends it to the server from its own memory set. Then the server takes the packet and replies to the previous message which says payload = Shashank and size of the payload = 65527 to the client. And the client repeatedly sends

this information from its own memory, so the attacker can easily get access to confidential information.

Therefore, Heartbleed is a very high level vulnerability which has to be taken care of. And on this, not only the servers, but also the clients are vulnerable in a passive attack. Actually, this is not a design fault of open SSL connection, but it is a bug generated with a programmer's mistake. So, the attackers are not required to install any malicious content on the target. And it is the biggest threat to the barrier of security for all the organizations. Already there are billions of dollars lost and most of the biggest companies are vulnerable and got themselves patches as soon as the vulnerability was detected. And most of the companies did not come out and admit that they are vulnerable because of trust factor.

VI. Heartbleed Implementation

Initially, we tried to execute these attacks in a local environment using virtualization on a PC. We downloaded GNS3 latest Beta version. Then we configured a CISCO router 3725 advertising and its networks using OSPF in area 0. One faster Ethernet port is configured as 192.168.144.20/24 as a subnet mask. That interface was attached to the kali Linux on the virtual machine and given a static IP:192.168.144.10/24 so that created a separate network. This kali Linux is the means by which we are going to launch an attack on a webserver.

The host machine working on windows 8.1 is running acts as a loopback which is attached to another fast Ethernet port on router R1 and assigned IPv4 network 1.0.0.0/8. In this network we have taken 1.1.1.2/8 to the host loopback and 1.1.1.1/8 as the fast Ethernet port of the router. The other two interfaces of R1 are connected with serial cables, through which the networks are advertised from LAN to WAN.

The serial network on the right hand side is assigned 200.20.20.0/24 network. That network's other end is connected with Router R10 which is an SSL gateway which acts as a proxy for a webserver in the network 10.0.0.0/8.

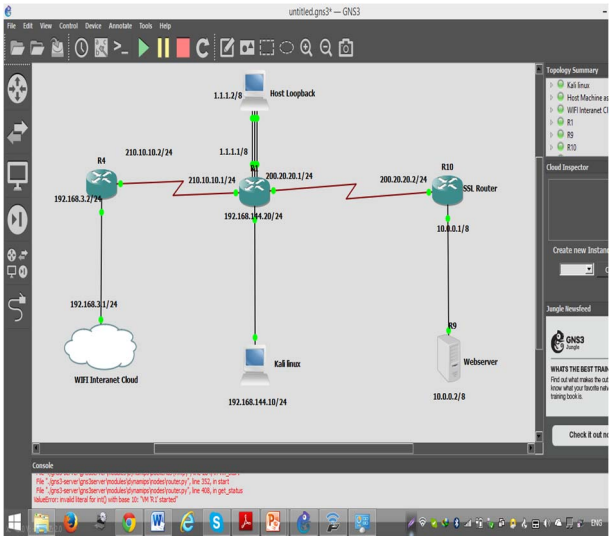

Figure 5. Local network diagram for the attacks

The other side of R1 router is connected with a serial cable to router R4 on 210.10.10.0 network and the other end of that router is connected to a WIFI internet cloud on the network 192.168.3.0/24. This Wi-Fi intranet cloud is throwing a WI-FI signal from the PC to the surrounding area in which we can connect that network to mobile devices like iPads, laptops, mobile phones, etc. and will be in this network and can access the servers or the other services that this network provides.

We configured a webserver on IP address 10.0.0.2 which is directly connected to R10 router. This R10 router is a SSL router, which will initiate the SSL negotiation as well as authenticate the user. Then, a secure SSL connection between any systems which are trying to use the webserver, occurs only if the peer proves its authenticity to the R10 router. And this router acts as a proxy server to the webserver.

Thereafter, we tried the WI-FI cloud. The PC was throwing WIFI and acting as a new router which will route all the packets sent by the peers or mobile devices which are connected to the WIFI. We were using "Connectify" software to make the network interface card as a WIFI router. Then we used a PC on which we were running Windows XP. We tried HTTPS://200.20.20.2:8080/HR which is the gateway of the SSL router. Because this SSL connection was not certified by any trusted source, it says that this is an untrusted site using Internet explorer as a browser. However, we can add exceptions to that site going to the SSL certification manager in the web browser settings. In addition, we can get that certification exception handling button when we enter the URL in the URL bar. Then we can add an exception for this particular website and tell the browser that we trust this site.

Then after the SSL negotiation, we needed to get authenticated with the SSL router in case R10. Therefore, we created a few usernames and passwords while configuring the SSL VPN. Then, when the login screen popped up asking for the username and password, we entered "user1" as the username and "Admin" as the password. In the same way, we also created multiple usernames and passwords for multiple users. Immediately after getting authenticated to the server, we were redirected to a HTTP server which is on the 10.0.0.2/8 IP address. Then a SSL connection was formed. So, the SSL router acts as a proxy server and it will not let any of the users get communicated directly with the webserver until the user gets authenticated with the SSL router.

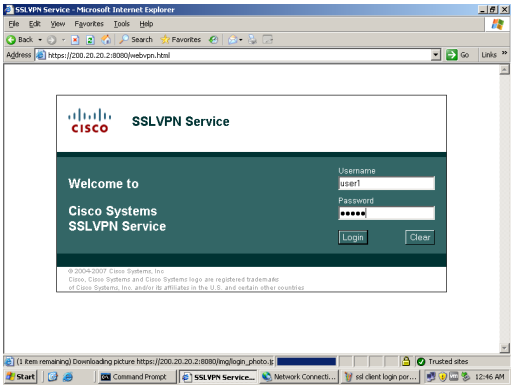

Figure 6. Login webpage page to SSL Server

We got a webpage in which we could navigate to an HTTP server when we clicked on HTTP_SERVER in the Bookmark section. After that, immediately we were redirected to the webserver's page which is a CISCO System's auto generated HTTP page.

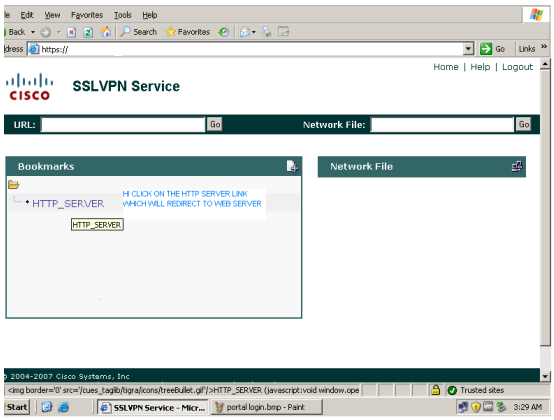

Figure 7. HTTP Server page

After that, an SSL connection was formed from a host system to the web server. In here we could launch an attack from kali Linux on to SSL router's outer interface because, authentication procedure and all the sensitive data is present in the flash memory of a router which is using SSL version 1.0.1.
Initially, in order to know that we are vulnerable we can use:
#nmap –sV –script=ssl-heartbleed 192.168.31.1

In our case, we used an SSL connection that was vulnerable to the Heartbleed attack.

In order to exploit that vulnerability, we can use our Kali machine which has Metasploit tool in-build. Although it is in-built, we need to update it by typing:
 # msfupdate
#msfconsole

This opened a new console on the kali Linux, called a Metaspolit tool. A Heartbleed exploiter was already present inside the auxiliary scanner. And that will be available by using:
#msf> use auxiliary/scanner/ssl/openssl_heartbleed

- There are options that we can see when we use the command in the console:
>Show options

```
   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   RHOSTS                      yes       The target address range or
CIDR identifier
   RPORT      443              yes       The target port
   STARTTLS   None             yes       Protocol to use with STARTTLS,
None to avoid STARTTLS  (accepted: None, SMTP, IMAP, JABBER, POP3)
   THREADS    1                yes       The number of concurrent
threads
   TLSVERSION 1.1              yes       TLS version to use (accepted:
1.0, 1.1, 1.2)
```

Figure 8. Ports Option with the descriptions

- We can use verbose below command to set a port
>Set RHOSTS 200.20.20.2
>show options
>Set RPORT 443

- We can also use verbose command in order to know every message and activity going on with my attack..
>set verbose true

- Start attack by using this command.
>exploit.

This is how a Heartbeat Message can grab the information stored inside the webserver. So, the version that the present router is using can easily be exploitable. As you have seen, just a couple of commands on kali Linux can craft a heartbeat packet so that we could perform a Heartbleed attack efficiently.

VII. Mitigation of Heartbleed

To support the main idea of a secure use of the SSL connection. We should not send more than what we have received in the Heartbeat Message, which is the fundamental rule to patch-up the SSL connection that was resolved in SSL version 1.0.1g. Therefore, every peer either a server or a client should patch-up this vulnerability by updating their SSL version.

- Here are some commands for updating Fedora/CentOS as well as Ubuntu/Debain:

Fedora/CentOS:
 # yum clean all
 # yum check-update
 # yum update

Ubuntu/Debian:

 # sudo apt-get update
 # sudo apt-get upgrade openssl

-After the update, confirm that we are using the correct version of Open SSL connection,

Fedora/CentOS

#rpm –qa | grep openssl
Openssl-1.0.1e-16.e16_5.7.x86_64
OpenSSl-devel-1.0.1e-16.e16_5.7.x86_64

Ubuntu/Debian

#openssl version
opensSSL 1.0.1g 14 May 2014

After making sure that we were updated, we needed to re-generate a new certificate, because in the Heartbleed attack, most of the user's and the admins's private keys were stolen

passively. Therefore, we needed to generate two new pairs of keys (Public and Private Key) and get that one certified from a trusted certification authority and update our certificate manager list. Then we could restart all the services on the network as well as the systems which are using SSL services. This procedure has to be done to all the servers and services not only webservers but also MySQL, Nginx and vsftpd etc.
Restart Command:

Fedora/CentOS
 # /etc/init.d/httpd restart

Ubuntu/Debian
#Sudo service apache2 restart

VIII. Conclusion

It is highly recommended that we all need to be aware of such a dreadful vulnerability in the Open SSL connection. This bug is a programmer's mistake which was not caught while testing Open SSL connection. And it was not identified until April 2012. Although the mistake is minor, it can result in a major disaster. The Heartbleed is vulnerability in open SSL heartbeat messages which were implemented in 2012 for keep a live sessions.
It can be exploited in multiple ways in both the server as well as the client. This attack is in both directions and the client should also take multiple precautions to over this.
The safer browsers are Firefox, Chrome and Internet Explorer, as all of them do not use open SSL. To open the SSL, the application uses a Microsoft implementation which is IIS(Internet Information services), and these servers are also not vulnerable.
Therefore, it is recommended to stop all the services used by the current SSL connection on in the devices and upgrade the SSL with 1.0.1g. After that, a new certificate for new pair of asymmetric keys should be generated and be confident with the communications.

References

[1] Todd Lammle, CCSI, CCNA study Guide, OSPF protocol, 6th edition,Wiley-India Edition, 2008 pp.327-607

[2] Mohamed Ramadan How to Exploit HeartBleed Vulnerability in REAL WORLD!, Apr 22, 2014 http://attack-secure.com/how-to-exploit-heartbleed-vulnerability-in-real-world/

[3] Edge, Detect Exploit openSSL Heartbleed Vulnerability using Nmap and Metasploit on Kali Linux, May 05, 2014 http://www.linuxtoday.com/security/detect-exploit-openssl-heartbleed-vulnerability-using-nmap-and-metasploit-on-kali-linux.html

[4] Linux Digest, Explout Heartbleed OpenSSL Vulenerability using kali Linux, June 10, 2014 https://sathisharthars.wordpress.com/2014/06/10/exploit-heartbleed-openssl-vulnerability-using-kali-linux/

[5] Heartbleed, The Heartbleed Bug, April 2014 http://heartbleed.com/

[6] Justin Ellingwood, How to instaill, Configure and use modules in the apache webserver, Aug 8 2013 https://www.digitalocean.com/community/tutorials/how-to-install-configure-and-use-modules-in-the-apache-web-server

[7] Vexxhost, How to mitigate and fix openSSL, Sep 11, 2014 http://vexxhost.com/blog/how-to-mitigate-and-fix-openssl-heartbeat-on-centos-or-ubuntu/

[8] Digicert Ubuntu server with Apache2 SSL Certificate Installation, 2014. https://www.digicert.com/ssl-certificate-installation-ubuntu-server-with-apache2.htm

[9] Tony Bradley, Reverse Heartbleed puts your PC and devices at risk of open SSL attack. http://www.pcworld.com/article/2142808/reverse-heartbleed-puts-your-pc-and-the-internet-of-things-at-risk.html

[10] Logmein, Testing for "Reverse" Heartbleed, April 10, 2014. http://blog.meldium.com/home/2014/4/10/testing-for-reverse-heartbleed

[11] Zulfikar Ramzan, Reverse Heartbleed can atrack PCs and mobiles, April 24, 2014 http://www.scmagazine.com/reverse-heartbleed-can-attack-pcs-and-mobile-phones/article/344108/

[12] Ivan Ristic, SSL labs Test for the Heartbleed Attack, Apr 8, 2014. https://community.qualys.com/blogs/securitylabs/2014/04/08/ssl-labs-test-for-the-heartbleed-attack

[13] Trend Micro, Open SSL Heartbleed: Are you?, 2014. http://www.trendmicro.com/us/security/heartbleed/

[14] Tod Beardsley, Metasploit's Brand New Heartbleed Scanner Module, Apr 9, 2014. https://community.rapid7.com/community/metasploit/blog/2014/04/09/metasploits-heartbleed-scanner-module-cve-2014-0160

[15] Ken Westin, Heart Attack: Detecting Heartbleed Exploits In Real-Time, Apr 14, 2014. http://www.tripwire.com/state-of-security/incident-detection/heart-attack-detect-heartbleed-exploits-in-real-time-with-active-defense/

[16] Matthew Green, Attack of the week: OpenSSL Heartbleed, Apr 8, 2014. http://blog.cryptographyengineering.com/2014/04/attack-of-week-openssl-heartbleed.html

[17] P.JMalloy, How to Detect a Prior Heartbleed Exploit, Apr 9, 2014. http://www.riverbed.com/blogs/Retroactively-detecting-a-prior-Heartbleed-exploitation-from-stored-packets-using-a-BPF-expression.html

[18] Sanchit Karve, Heartbleed Vulnerability Opens the Door to SSL Heartbeat Exploits. Apr 09, 2014 http://blogs.mcafee.com/mcafee-labs/heartbleed-vulnerability-opens-the-door-to-ssl-heartbeat-exploits