© 2015 IJIRT | Volume 1 Issue 12 | ISSN: 2349-6002

Analysing Heartbleed vulnerability in OpenSSL

Shuchi B. Shah
Silver Oak College of Engineering And Technology, Ahmedabad

Abstract–A OpenSSL versions 1.0.1 through 1.0.1f—introduced beginning in March 2012—had a serious memory handling flaw in their implementation of the TLS Heartbeat Extension.

This extension helps maintain user Internet connections without the need for continuous data transfers. Attackers could exploit the bug to access an application's memory, including sensitive data and private encryption keys. The latter could let them launch man-in-the middle attacks and decode intercepted communications. This has been introduced as the worst vulnerability in the modern internet.

Existence of this vulnerability for two years without being discovered illustrates that a lot of important open source Internet software is not funded, developed, or reviewed carefully enough and the testing tools are not advanced enough to detect it.

I. INTRODUCTION

In the history of modern internet there is a serious Heartbleed vulnerability in the OpenSSL cryptographic software library. Through which you can steal the information, protected by the SSL/TLS encryption used to secure the Internet. SSL/TLS encryption provides communication security and privacy over the Internet for applications such as web, email, instant messaging (IM) and VPNs (private virtual networks).

Through this vulnerability anyone on the Internet can read the memory of the system protected by the vulnerable versions of the OpenSSL software. This compromises the secret keys used to identify the service providers and to encrypt the traffic, the usernames and passwords and the actual content. This allows attackers to eavesdrop on communications and steal data directly from these

A client is not doing bound checking and had put a complete trust on user-supplied input. Placing trust in user-supplied input is often a bad idea. A client should not blindly trust the payload length presented in the heartbeat_request packet.

II RELATEDWORK

OpenSSLOpenSSL is the open-source implementation of SSL/TLS protocols. OpenSSL is the core library written in C language, which implements basic cryptographic functionalities. Wrappers allowing the use of the OpenSSL library in a variety of computer languages are available

protocols of OpenSSL

SSL/TLS Protocol:

Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), are cryptographic protocols designed to provide communication security over the Internet. SSL protocol is originally created by Netscape[5]

SSLV3.0 is equivalent to TLS 1.0. Although there are some slight differences between SSL 3.0 and TLS 1.0, it refers to the protocol as TLS/SSL.

Most electronic commerce (e-commerce) applications in use today employ the Secure Sockets Layer (SSL) or Transport Layer Security (TLS) protocol to authenticate the server to the client and to cryptographically protect the communication channel between them.

TLS is application protocol independent. It runs above TCP/IP and below application protocols such as HTTP or IMAP. The HTTP running on top of TLS or SSL is often called HTTPS.

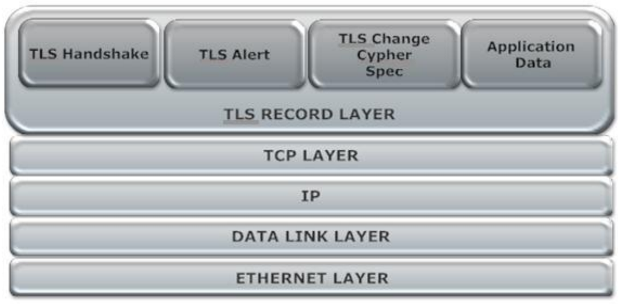


Fig :SSL/TLS protocol layer

Handshake Protocol

The TLS Handshake protocol, used to negotiate and instantiate the security parameters for the record layers, to authenticate the users, and to report error conditions. The TLS handshake protocol is used by server and a client to establish a session.

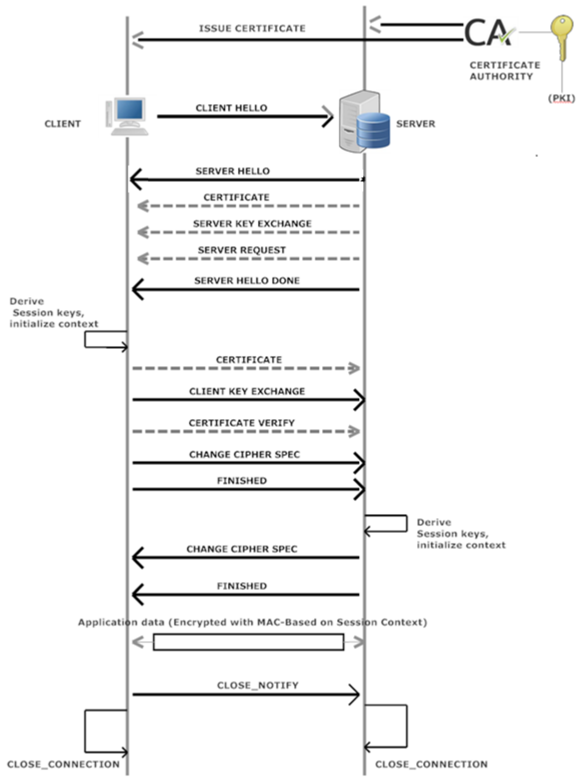


Fig - TLS Handshake process

STEP 1:
The server and client exchange hello message to negotiate algorithms, exchange random values, and and initiate or resume the session.

STEP 2:
The client and server exchange cryptographic parameters to allow them to agree on a parameter secret. If necessary, they exchange certificates and cryptographic information to authenticate each-other. They then generate a master secret from the premaster secret and exchange random values.

STEP:3
The client and server provide their record layer with the security parameters. Theclient and server verify that their peer has calculated the same security parameters and that the handshake occurred without tampering by an attacker.

DTLS

DTLS is an implementation of TLS over UDP. UDP communications exist as streams of packets with no ordering, delivery reliability, or flow control. Applications that use datagram protocols need to make sure they can handle these concerns internally.

The drawback of using TLS is, it cannot be used to secure datagram traffic as it requires a reliable transport channel TCP. TLS will give more delay than DTLS.DTLS (Datagram Transport Layer Security) is used where low latency or "delay sensitive" data must be secured, such as Voice over IP, VPN, Video Conferencing, and various real-time and Massively Multiplayer Online Games

HEARTBEAT EXTENSION OF TLS/DTLS

Heartbeat extension of TLS (transport Layer Security) and DTLS(Datagram transport layer security) gives a fresh protocol to provide the stay-alive function by not performing a repeat negotiation.

DTLS is designed such that it secures traffic on non-reliable transport protocol (UDP). There is no session management on such unreliable transport protocols. Costly Renegotiation, in particular when the application uses unidirectional traffic is the only way to keep the session alive.

TLS is based on reliable protocol. But once the session breaks, there is no such feature to let the session going without non-stop data transfer.[4]

The Heartbeat Extension overcomes these limitations. This protocol is a new protocol running on top of the Record Layer

Heartbeat Hello Extension

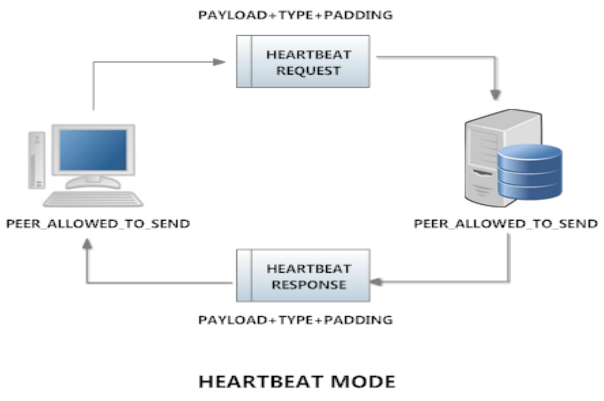


Fig: TLS channel Heartbeat communication

Heartbeat extension of TLS is depicted as TLS Hello Extension. Here A peer can chose whether it wants to accept HeartBeat request or just wants to send HeartBeat request. If the peer set to peer_allowed_to_send .then that peer is willing to accept the HeartBeat request. If peer is set to peer_not_allowed_to_send , that peer will discard the incoming HeartBeat request.

This decision can change with every renegotiation. While this process HeartBeat Mode is set. With the reception of any unknown mode the illegal_parameter error message will be thrown.

III VULNERABILITY IN OPENSSL

There was a security bug Heartbleed disclosed recently in April 2014 in the OpenSSLcryptography library, which is the open-source implementation of the Transport Layer Security (TLS) protocol. Heartbleed may be exploited regardless of whether the party using a vulnerable OpenSSL instance for TLS is a server or a client.This vulnerability results from improper input validation and due to a missing **bounds check** in the implementation of the TLS heartbeat extension, thus the bug's name derives from "heartbeat". The vulnerability is classified as a **buffer over-read** a situation where more data can be read than should be allowed.

The Heartbleed vulnerability was originally found by a Google computer security employee Neel Mehta, in March 2014. Upon finding the bug and patching its servers, Google notified the core OpenSSL team on April 1. Independently, on April 2, a security consulting firm, Codenomicon, found the vulnerability and reported that to National Cyber Security Centre Finland (NCSC-FI).Finally in got publically disclosed in April 2014.

The implementation of RFC 6520 was, in theory, simple: send a packetheartbeat_request, along with an arbitrary payload, payload length and padding. The request should be answered by heartbeat_response that contains exact copy of the payload.

This mechanism would allow for more streamlined checking of connection state and lower client/ server overhead on long-lived connections. Unfortunately, as history shows, a lot can go wrong between design and implementation.

Version 1.0.1 of OpenSSL added support for the Heartbeat functionality and enabled it by default, thereby inadvertently making the implementation vulnerable by default. This vulnerability remained until 1.0.1g—or roughly two years.

Specification of the vulnerability

While secured communication between 2 peers, to keep the session alive, sender will send a heartbeat request to the receiver, which is, suppose, of 1 byte. This request packet contains some data. As payload and the size of the payload which is mentioned in the header of that request packet. The peer who is receiving his request will store payload and size information on its memory buffer and while responding it will send the same heartbeat response, having the same payload information and little padding.

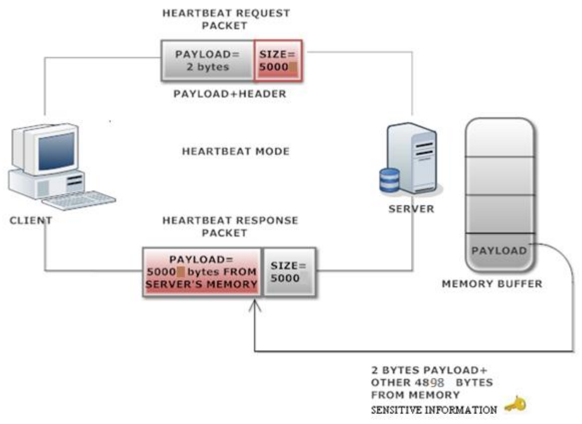


Fig: Specification of Vulnerability

The way the attack works is:

The attacker will crack a special heartbeat request. After cracking it, he will change the size of the payload mentioned in the header.

The actual size of heartbeat payload is very small. For an instance, the payload size of the request is 2 bytes. So, the size mentioned in the header should be 2. Here the attacker will change the size as suppose 50000. But the payload is of 2 bytes. The receiver, who is getting this heartbeat request, will store the payload and size of payload in its buffer And respond back with 2 bytes of payload which was sent by sender and other 48000 bytes of information from its memory buffer.

This information can be some private key, passwords, username, credit card password etc. So the hacker can get all the critical information just by changing the size of the payload of the heartbeat request packet.

This whole vulnerability is graphically represented below:

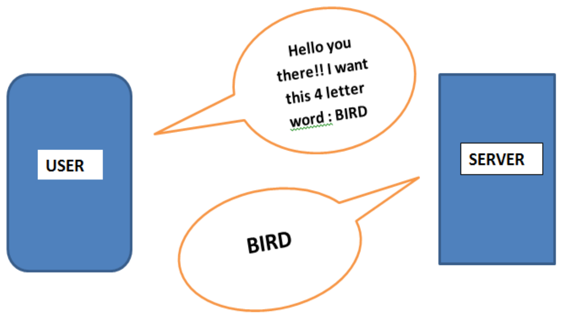


Fig: Heartbeat request that keeps the session alive

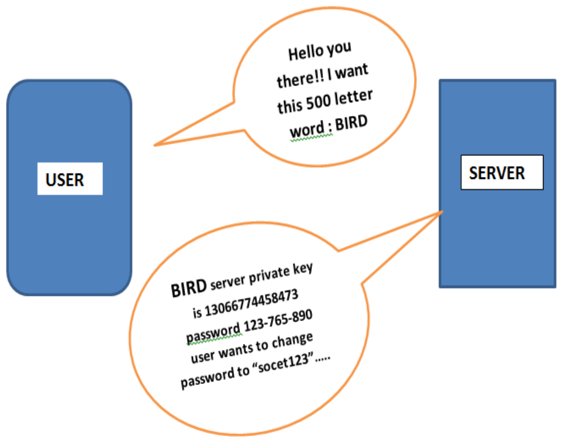
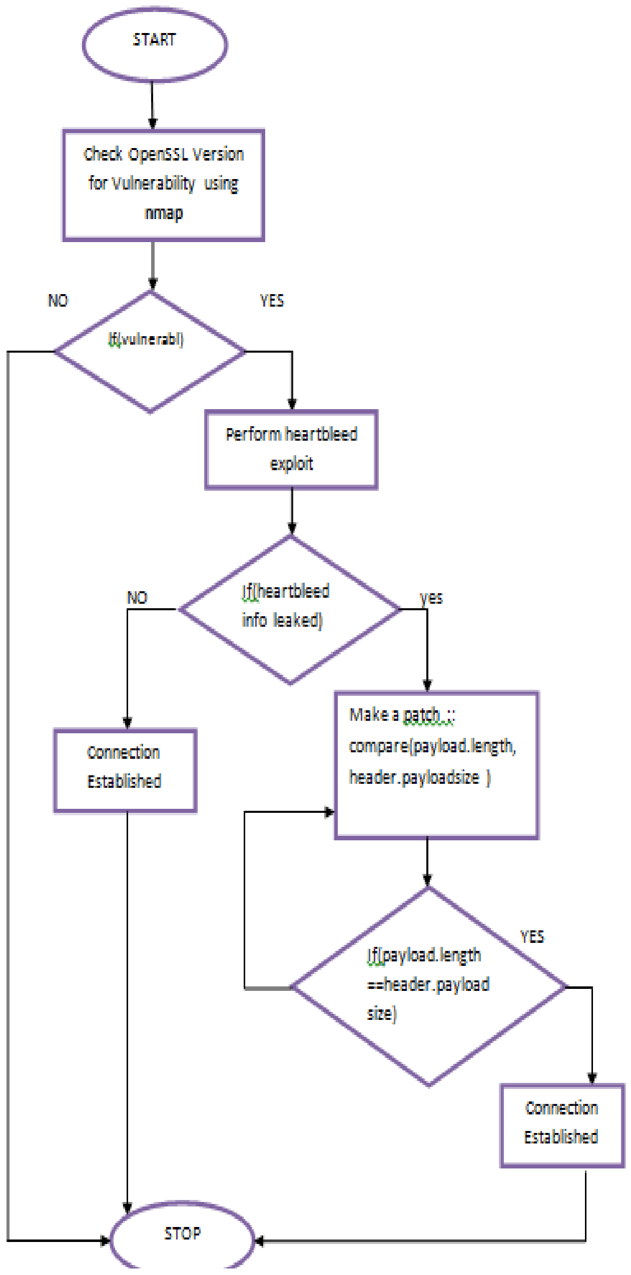

USER
Hello you there!! I want this 500 letter word : BIRD
SERVER
BIRD server private key is 13066774458473 password 123-765-890 user wants to change password to "socet123"…..

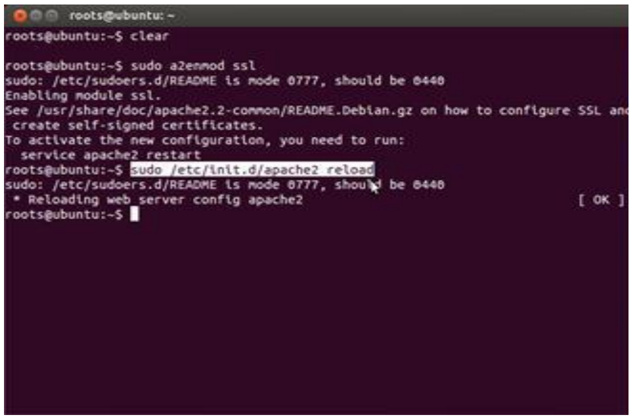
Fig: Buffer Over read in SSL Heartbeat Response

IV. PROPOSED WORK


START
Check OpenSSL Version for Vulnerability using nmap
NO
YES
If(vulnerabl)
Perform heartbleed exploit
NO
yes
If(heartbleed info leaked)
Connection Established
Make a patch :: compare(payload.length, header.payloadsize)
YES
If(payload.length ==header.payload size)
Connection Established
STOP

V. IMPLEMENTATION

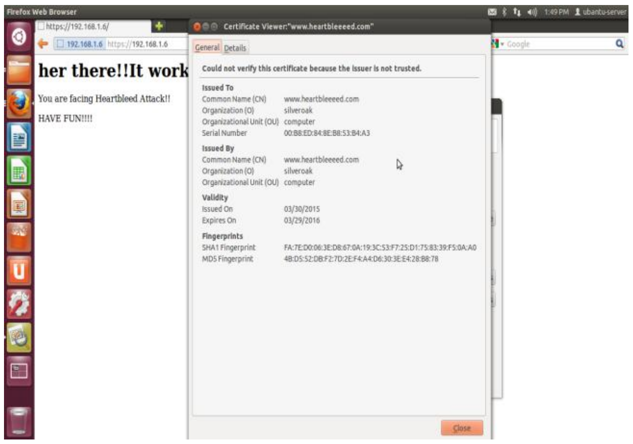
Enable OpenSSL and Apache



```
roots@ubuntu:~$ clear

roots@ubuntu:~$ sudo a2enmod ssl
sudo: /etc/sudoers.d/README is mode 0777, should be 0440
Enabling module ssl.
See /usr/share/doc/apache2.2-common/README.Debian.gz on how to configure SSL and
 create self-signed certificates.
To activate the new configuration, you need to run:
  service apache2 restart
roots@ubuntu:~$ sudo /etc/init.d/apache2 reload
sudo: /etc/sudoers.d/README is mode 0777, should be 0440
 * Reloading web server config apache2                                    [ OK ]
roots@ubuntu:~$ 
```

Self-generated Certificate



her there!!It work

You are facing Heartbleed Attack!!

HAVE FUN!!!!

Certificate Viewer:"www.heartbleeeed.com"

Could not verify this certificate because the issuer is not trusted.

Issued To

Common Name (CN)	www.heartbleeeed.com
Organization (O)	silveroak
Organizational Unit (OU)	computer
Serial Number	00:B8:ED:84:BE:B8:53:B4:A3

Issued By

Common Name (CN)	www.heartbleeeed.com
Organization (O)	silveroak
Organizational Unit (OU)	computer

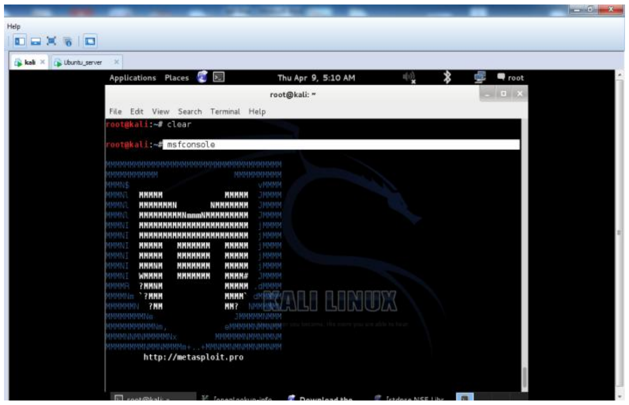
Validity

Issued On	03/30/2015
Expires On	03/29/2016

Fingerprints

SHA1 Fingerprint	FA:7E:D0:06:3E:D8:67:0A:19:3C:53:F7:25:D1:75:83:39:F5:0A:A0
MD5 Fingerprint	4B:D5:52:DB:F2:7D:2E:F4:A4:D6:30:3E:E4:28:B8:78

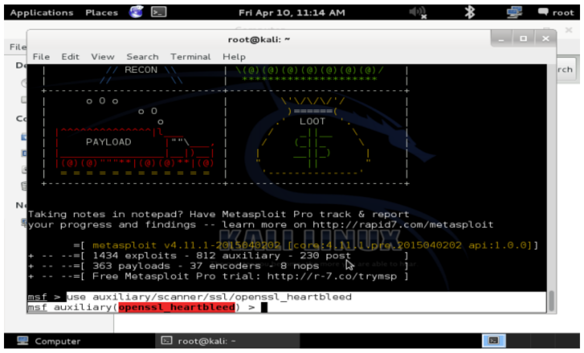
Exploiting with Metasploit



```
root@kali:~# clear

root@kali:~# msfconsole

        http://metasploit.pro
```

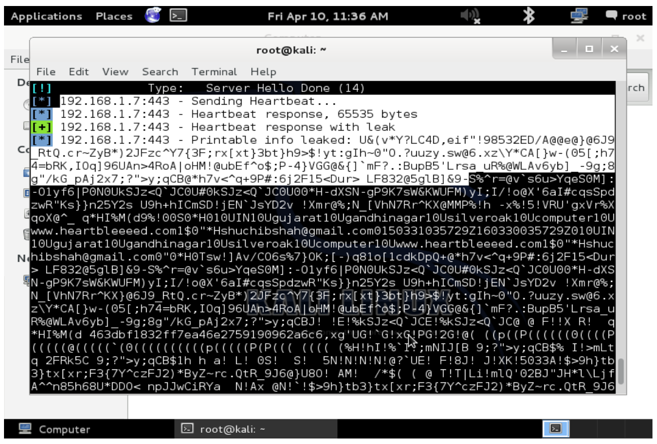


```
Taking notes in notepad? Have Metasploit Pro track & report
your progress and findings -- learn more on http://rapid7.com/metasploit

       =[ metasploit v4.11.1-2015040202 [core:4.11.1.pre.2015040202 api:1.0.0]]
+ -- --=[ 1434 exploits - 812 auxiliary - 230 post        ]
+ -- --=[ 363 payloads - 37 encoders - 8 nops             ]
+ -- --=[ Free Metasploit Pro trial: http://r-7.co/trymsp ]

msf > use auxiliary/scanner/ssl/openssl_heartbleed
msf auxiliary(openssl_heartbleed) > 
```

Exploit Heartbleed



```
[!]          Type:   Server Hello Done (14)
[*] 192.168.1.7:443 - Sending Heartbeat...
[*] 192.168.1.7:443 - Heartbeat response, 65535 bytes
[+] 192.168.1.7:443 - Heartbeat response with leak
[*] 192.168.1.7:443 - Printable info leaked: U&(v*Y?LC4D,eif"!98532ED/A@@e@}@6J9
_RtQ.cr~ZyB*)2JFzc^Y7{3F;rx[xt}3bt}h9>$!yt:gIh~0"0.?uuzy.sw@6.xz\Y*CA[}w-(05[;h7
4=bRK,I0q]96UAn>4RoA|oHM!@ubEf^o$;P-4}VGG@&{]`mF?.:BupB5'Lrsa_uR%@WLAv6yb]_-9g;8
g"/kG_pAj2x7;?">y;qCB@*h7v<^q+9P#:6j2F15<Dur> LF832@5glB]&9-S%^r=@v`s6u>YqeS0M]:
-01yf6|P0N0UkSJz<Q`JC0U#0kSJz<Q`JC0U00*H-dXSN-gP9K7sW&KWUFM)yI;I/!o@X'6aI#cqsSpd
zwR"Ks}}n25Y2s U9h+hICmSD!jEN`JsYD2v !Xmr@%;N_[VhN7Rr^KX@MMP%!h -x%!5!VRU'gxVr%X
qoX@^_ q*HI%M(d9%!00S0*H010UIN10Ugujarat10Ugandhinagar10Usilveroak10Ucomputer10U
www.heartbleeeed.com1$0"*Hshuchibshah@gmail.com0150331035729Z160330035729Z010UIN
10Ugujarat10Ugandhinagar10Usilveroak10Ucomputer10Uwww.heartbleeeed.com1$0"*Hshuc
hibshah@gmail.com0"0*H0Tsw!]Av/C06s%7}0K;[-)q81o[1cdkDpQ+@*h7v<^q+9P#:6j2F15<Dur
> LF832@5glB]&9-S%^r=@v`s6u>YqeS0M]:-01yf6|P0N0UkSJz<Q`JC0U#0kSJz<Q`JC0U00*H-dXS
N-gP9K7sW&KWUFM)yI;I/!o@X'6aI#cqsSpdzwR"Ks}}n25Y2s U9h+hICmSD!jEN`JsYD2v !Xmr@%;
N_[VhN7Rr^KX}@6J9_RtQ.cr~ZyB*)2JFzc^Y7{3F;rx[xt}3bt}h9>$!yt:gIh~0"0.?uuzy.sw@6.x
z\Y*CA[}w-(05[;h74=bRK,I0q]96UAn>4RoA|oHM!@ubEf^o$;P-4}VGG@&{]`mF?.:BupB5'Lrsa_u
R%@WLAv6yb]_-9g;8g"/kG_pAj2x7;?">y;qCBJ! !E!%kSJz<Q`JCE!%kSJz<Q`JC@ @ F!!X R!   q
*HI%M(d 463dbf1832ff7ea46e2759190962a6c6,xg'UG!`G!xG!PG!2G!@( ((p((P((((((0((((P
(((((@(((((`(0((((((((((p(((((P(P((( (((( (%H!hI!%`I*;mNIJ[B 9;?">y;qCB$% I!>mLt
q 2FRk5C 9;?">y;qCB$1h h a! L! 0S!  S!  5N!N!N!N!@?`UE! F!8J! J!XK!5033A!$>9h}tb
3}tx[xr;F3{7Y^czFJ2)*ByZ~rc.QtR_9J6@}U80! AM!  /*$( ( @ T!T|Li!mlQ'02BJ"JH*l\Ljf
A^^n85h68U*DD0< npJJwCiRYa  N!Ax @N!`!$>9h}tb3}tx[xr;F3{7Y^czFJ2)*ByZ~rc.QtR_9J6
```

VI.IMPLEMENTATION TOOL

Wireshark

Wireshark is a network packet analyzer. It is also called network sniffer. The network sniffer will capture Network packets and display that packet data as detailed as possible.

Thought wireshark we can get to know what's going on inside a network cable. Wireshark is perhaps one of the best open source packet analyzers available today.

VII. CONCLUSION AND FUTURE WORK

This poster is about the worst vulnerability since e-commerce began on the Internet, **Heartbleed**which can cause devastating effects to the web security.

Even the testing tools were not able to find this subtle bug. We need to improve our testing toolsthat they can catch this kind of vulnerabilities in the future. This was the buffer over- read vulnerability in OpenSSL. Which was failed to do bound checking?

The root cause behind that was , the receiver was accepting request from sender blindly. It was not making any check whether the incoming request is trustworthy or not. It should check the actual payload with the size mentioned in the header before accepting the packet

IV. REFFERENCE

[1]Marco Carvalho, Jared DeMott, Richard Ford, David A. Wheeler, Florida Institute of Technology,*Heartbleed 101*,©August 2014 IEEE

[2] David A. Wheeler, Institute for Defense Analyses, *Preventing Heartbleed*, ©August 2014 IEEE

[3] *News Brief about Heartbleed Vulnerability*, Published by the IEEE Computer Society

[4]R. Seggelmann,M. Tuexen,Muenster Univ. of Appl. Sciences,M. Williams,GWhiz Arts & Sciences, *Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension*, February 2012**,** Internet Engineering Task Force (IETF)

[5]T. Dierks, E. Rescorla, RTFM, Inc. *The Transport Layer Security (TLS) Protocol Version 1.2*, August 2008, Internet Engineering Task Force (IETF)

[6] Christopher Meyer,*20 Years of SSL/TLS Research, An Analysis of the Internet's Security Foundation,* 9th February 2014

[7] Nadhem J. AlFardan and Kenneth G. Paterson Information Security Group Royal Holloway, University of London, *Breaking the TLS and DTLS Record Protocols*, 27th February 2013

[8] ZakirDurumeric, James Kasten, David Adrian, J. Alex Halderman, Michael Bailey
University of Michiga University of Illinois, Urbana Champaign, *The Matter of Heartbleed*

[9]James A. Kupsch and Barton P. Miller, University of Wisconsin-Madison, *Why Do Software Assurance Tools Have Problems Finding Bugs Like Heartbleed?*,22nd april 2014

[10]Sean cassidy, *Diagnosis of the OpenSSLHeartbleed Bug,* Mon 07 April 2014,http://www.seancassidy.me/diagnosis-of-the-openssl-heartbleed-bug.html

[11]Rolf Oppliger, Ralf Hauser, David Basin, Aldo Rodenhaeuser and Bruno Kaiser,*A Proof of Concept Implementation of SSL/TLS Session-Aware User Authentication (TLS-SA)*

[12] *HeartBleed Memory Disclosure*
Available: http://vrt-blog.snort.org/2014/04/heartbleed-memory-disclosure-upgrade.html

[13] *Everything you need to know about the Heartbleed SSL bug ,9 April,2014*

Available:http://www.troyhunt.com/2014/04/everything-you-need-to-know-about.html

[14]*Why Heartbleed is dangerous? Exploiting CVE-2014-0160,*Paweł Krawczyk,4 April,2014
Available:http://ipsec.pl/ssl-tls/2014/why-heartbleed-dangerous-exploiting-cve-2014-0160.html

[15]*How the Heartbleed bug reveals a flaw in online security,* 11 April 2014

Available:http://theconversation.com/how-the-heartbleed-bug-reveals-a-flaw-in-online-security-25536

[16]*Heartbleed*
,http://en.wikipedia.org/wiki/Heartbleed