# A technical view of theOpenSSL ‘Heartbleed’ vulnerability

## A look at the memory leak in the OpenSSL Heartbeat implementation

Bipin Chandra
bipin.chandra@in.ibm.com

Version 1.2.1, May 13, 2014

**Abstract**:
This article describes ‘OpenSSL Heartbleed Vulnerability’ in detail. It describes the ‘HeartBleed’ problem, its causes and its impact. The purpose of this article is to increase awareness about Heartbleed vulnerability in OpenSSL library, using which attackers can get access to passwords, private keys or any encrypted data. It also explains how Heartbleed works, what code causes data leakage and explains the resolution with code fix.

a security on developerWorks
community white paper
ibm.biz/dwsecurity

## • Table of Contents

1Introduction to Heartbleed Vulnerability.....3
2Heartbleed Explanation.....3
2.1The OpenSSL Project.....3
2.2SSL, TLS and DTLS Protocols.....3
2.3TLS/DTLS Heartbeat Extension.....4
2.3.1How the heartbeat works.....4
2.3.2Heartbeat Implementation in OpenSSL.....5
3The real-world impact of Heartbleed.....13
4Heartbleed resolutions, precautions and preventions.....15
4.1Immunize your application from Heartbleed.....16
4.2Does this resolve all the problems?.....16
4.3Heartbleed detector tools.....16
5Immediate aftermath of Heartbleed for different stakeholders.....17
6Conclusion.....17
7References.....17
8About Author.....18

## • Table of Figures

Figure 1: Common internet protocol layers.....4
Figure 2: Memory leak.....11
Figure 3: The OpenSSL code fix for the Heartbleed bug.....12
Figure 4: OpenSSL Security Advisory.....13
Figure 5: Exploiting the Heartbleed vulnerability.....14

## • Table of Listings

Listing 1: openssl-1.0.1/ssl/d1_both.c dtls1_heartbeat function .....6
Listing 2: excerpt from dtls1_heartbeat.....7
Listing 3: OpenSSL code that builds the HeartBeatRequest payload.....7
Listing 4: openssl-1.0.1/ssl/t1_lib.c tls1_process_heartbeat function.....9
Listing 5: OpenSSL excerpt that builds the heartbeat response message.....10
Listing 6: incorrect memcpy in the code that builds the heartbeat response message.....10
Listing 7: Heartbeat payload zero length check.....12
Listing 8: Heartbeat payload actual length check.....12

# 1 Introduction to Heartbleed Vulnerability

The name ‘Heartbleed’ itself explains the vulnerability – ‘Heart’ of the Heartbleed came from Heartbeat protocol and ‘bleed’ stands for data leakage. That means data leakage in the Heartbeat protocol implementation, specifically the OpenSSL implementation of the protocol. OpenSSL is an open source and widely used library for the Secure Socket Layer (SSL) and Transport Layer Security (TLS) protocols. In this white paper, we will learn how serious this vulnerability is and what are the ways to help prevent this data leakage.

Encryption is the backbone of Internet security. It protects users data, passwords and transaction details from attackers. To achieve encryption over Internet, one of the famous and widely used protocols is HTTPS. HTTPS is simply HTTP over SSL/TLS. For example any online payment or banking transactions over Internet happens through HTTPS as it is secured. But this new vulnerability – Heartbleed- has put a question mark on this security of Internet itself and has broken a trust on the open source community.

# 2 Heartbleed Explanation

## 2.1 The OpenSSL Project

OpenSSL library provides implementation of cryptographic protocols such as SSL and TLS. It is open source software written in C programming language. The development is completely volunteer driven and the library is free to use for commercial and non-commercial purposes under an Apache-style license.

## 2.2 SSL, TLS and DTLS Protocols

Security over Internet can be achieved in many ways. Network layer security is one of them. Security over TCP/IP can be improved by using the Secure Socket Layer (SSL) or its follow-on protocol Transport Layer Security (TLS). These two protocols are commonly referred to together as SSL/TLS.

HTTP is a stateless application level protocol to format and transmit data between web servers and web browsers. With the increase in the threats and frauds over Internet, there is always a need for a more secure transmission of data. HTTPS is used for improve the security of communication over a network by providing a layer of SSL/TLS the between HTTP and TCP layer.

DTLS (Datagram Transport Layer Security) is a communication protocol which implements TLS over unreliable transport protocol i.e. Datagram Congestion Control Protocol (DCCP) or User Datagram Protocol (UDP).

Figure 1: Common internet protocol layers illustrates the relationship between these protocols.

| | RTCPeerConnection | DataChannel |
|---|---|---|
| WebSocket | SRTP | SCTP |
| HTTP | DTLS | |
| SSL/ TLS | ICE. STUN, TURN | |
| Transport Layer (TCP) | Transport Layer (UDP) | |
| Network Layer | | |

*Figure 1: Common internet protocol layers*

### 2.3 TLS/DTLS Heartbeat Extension

The heartbeat extension to the TLS/DTLS protocol is used to check if the connection between two communication devices using TLSDTLS are still "alive," i.e. able to communicate. It was introduced in 2012 by RFC 6520 (http://tools.ietf.org/html/rfc6520). Per the RFC, the Heartbeat protocol runs on top of the TLS Record Layer and maintains the connection between the two peers alive requiring them to exchange a "heartbeat." The heartbeat extension was introduced because the then-current TLS/DTLS renegotiation technique to figure out if a peer is still alive was a costly process.

#### 2.3.1 How the heartbeat works

The heartbeat extension protocol consists of two message types: HeartbeatRequest message and HeartbeatResponse message and the extension protocol depends on which TLS protocol is being used as describe below:

- When using reliable transport protocol:
  One side of the peer connection sends a HeartbeatRequest message to the other side. The other side of the connection should immediately send a HeartbeatResponse message. This makes one successful Heartbeat and thus, keeping connection alive – this is called 'keep-alive' functionality. If no response is received within a specified timeout, the TLS connection is terminated.
- Unreliable transport protocol:
  One side of the peer connection sends HeartbeatRequest message to the other side. The other side of the connection should immediately send a HeartbeatResponse message. If no response is received within specified timeout another HeartbeatRequest message is retransmitted. If expected response is not received for specified number of retransmissions, the DTLS connection is terminated.

When a receiver receives a HeartbeatRequest message, the receiver should send back an exact copy of the received message in the HeartbeatResponse message. The sender verifies that the HeartbeatResponse message is same as what was originally sent. If it is same, the connection is kept alive. If the response does not contain the same message, the HeartbeatRequest message is retransmitted for a specified number of retransmissions.

### 2.3.2 Heartbeat Implementation in OpenSSL

The OpenSSL team implemented the heartbeat extension in December 2011. This section briefly explains the code for heartbeat implementation for both HeartbeatRequest message and HeartbeatResponse message. It also explains the bug in the code and its fix in detail. The OpenSSL source code can be downloaded from the groups web site ate https://www.openssl.org/source/ or ftp://ftp.openssl.org/source/ . The bug exists in OpenSSL from version 1.0.1 to 1.0.1f.

#### *Sending Heartbeat Requests*

The OpenSSL implementation of the hesartbeat request code is shown in Listing 1: openssl-1.0.1/ssl/d1_both.c dtls1_heartbeat function below:

```
int
dtls1_heartbeat(SSL *s)
        {
        unsigned char *buf, *p;
        int ret;
        unsigned int payload = 18; /* Sequence number + random bytes */
        unsigned int padding = 16; /* Use minimum padding */

        /* Only send if peer supports and accepts HB requests... */
        if (!(s->tlsext_heartbeat & SSL_TLSEXT_HB_ENABLED) ||
            s->tlsext_heartbeat & SSL_TLSEXT_HB_DONT_SEND_REQUESTS)
                {
                SSLerr(SSL_F_DTLS1_HEARTBEAT,SSL_R_TLS_HEARTBEAT_PEER_DOESNT_ACCEPT);
                return -1;
                }

        /* ...and there is none in flight yet... */
        if (s->tlsext_hb_pending)
                {
                SSLerr(SSL_F_DTLS1_HEARTBEAT,SSL_R_TLS_HEARTBEAT_PENDING);
                return -1;
                }

        /* ...and no handshake in progress. */
        if (SSL_in_init(s) || s->in_handshake)
                {
                SSLerr(SSL_F_DTLS1_HEARTBEAT,SSL_R_UNEXPECTED_MESSAGE);
                return -1;
                }

        /* Check if padding is too long, payload and padding
         * must not exceed 2^14 - 3 = 16381 bytes in total.
         */
        OPENSSL_assert(payload + padding <= 16381);

        /* Create HeartBeat message, we just use a sequence number
         * as payload to distuingish different messages and add
         * some random stuff.
         *  - Message Type, 1 byte
         *  - Payload Length, 2 bytes (unsigned int)
         *  - Payload, the sequence number (2 bytes uint)
         *  - Payload, random bytes (16 bytes uint)
         *  - Padding
         */
        buf = OPENSSL_malloc(1 + 2 + payload + padding);
        p = buf;
        /* Message Type */
        *p++ = TLS1_HB_REQUEST;
        /* Payload length (18 bytes here) */
        s2n(payload, p);
        /* Sequence number */
        s2n(s->tlsext_hb_seq, p);
        /* 16 random bytes */
        RAND_pseudo_bytes(p, 16);
        p += 16;
        /* Random padding */
        RAND_pseudo_bytes(p, padding);

        ret = dtls1_write_bytes(s, TLS1_RT_HEARTBEAT, buf, 3 + payload + padding);
        if (ret >= 0)
                {
                if (s->msg_callback)
                        s->msg_callback(1, s->version, TLS1_RT_HEARTBEAT,
                                buf, 3 + payload + padding,
                                s, s->msg_callback_arg);

                dtls1_start_timer(s);
                s->tlsext_hb_pending = 1;
                }

        OPENSSL_free(buf);

        return ret;
        }
```

*Listing 1: openssl-1.0.1/ssl/d1_both.c dtls1_heartbeat function*

Listing 2: excerpt from dtls1_heartbeat shows the code snippet that says "payload and padding must not exceed 16381 bytes in total." Here, the maximum heartbeat request size is 16KByte (i.e. 16384 byte) which also includes one byte to identify that this message is a TLS Heartbeat request message and two bytes are dedicated for the length of the Heartbeat request message. So ‘payload and padding’ should not be more than 16381 bytes.

```
/* Check if padding is too long, payload and padding
 * must not exceed 2^14 - 3 = 16381 bytes in total.
 */

OPENSSL_assert(payload + padding <= 16381);
```

*Listing 2: excerpt from dtls1_heartbeat*

The OpenSSL implementation of the HeartbeatRequest message has a Message Type of 1 byte to identify that this message is a ‘TLS Heartbeat Request’ message, 2 bytes for the payload length, a 2 byte sequence number in the payload to identify to specified number of messages sent before a timeout, and 16 bytes for actual payload and any padding. The code excerpt that builds this message is show in in Listing 3: OpenSSL code that builds the HeartBeatRequest payload.

```
/* Create HeartBeat message, we just use a sequence number
        * as payload to distuingish different messages and add
        * some random stuff.
        *  - Message Type, 1 byte
        *  - Payload Length, 2 bytes (unsigned int)
        *  - Payload, the sequence number (2 bytes uint)
        *  - Payload, random bytes (16 bytes uint)
        *  - Padding
        */
       buf = OPENSSL_malloc(1 + 2 + payload + padding);
       p = buf;
       /* Message Type */
       *p++ = TLS1_HB_REQUEST;
       /* Payload length (18 bytes here) */
       s2n(payload, p);
       /* Sequence number */
       s2n(s->tlsext_hb_seq, p);
       /* 16 random bytes */
       RAND_pseudo_bytes(p, 16);
       p += 16;
       /* Random padding */
       RAND_pseudo_bytes(p, padding);
```

*Listing 3: OpenSSL code that builds the HeartBeatRequest payload*

The Heartbeat request message is created and sent to the receiver. The timer for timeout starts and the specified number of retransmission is updated. There in no problem in the OpenSSL Heartbeat request

implementation.

### *Heartbeat Response*

Heartbeat response sends a copy of the received Heartbeat request payload data which verifies that the secured connection between the peers is still alive, as shown in the OpenSSL excerpt in “Listing 4: openssl-1.0.1/ssl/t1_lib.c tls1_process_heartbeat function.”

```
int
tls1_process_heartbeat(SSL *s)
	{
	unsigned char *p = &s->s3->rrec.data[0], *pl;
	unsigned short hbtype;
	unsigned int payload;
	unsigned int padding = 16; /* Use minimum padding */

	/* Read type and payload length first */
	hbtype = *p++;
	n2s(p, payload);
	pl = p;

	if (s->msg_callback)
		s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT,
			&s->s3->rrec.data[0], s->s3->rrec.length,
			s, s->msg_callback_arg);

	if (hbtype == TLS1_HB_REQUEST)
		{
		unsigned char *buffer, *bp;
		int r;

		/* Allocate memory for the response, size is 1 bytes
		 * message type, plus 2 bytes payload length, plus
		 * payload, plus padding
		 */
		buffer = OPENSSL_malloc(1 + 2 + payload + padding);
		bp = buffer;

		/* Enter response type, length and copy payload */
		*bp++ = TLS1_HB_RESPONSE;
		s2n(payload, bp);
		memcpy(bp, pl, payload);
		bp += payload;
		/* Random padding */
		RAND_pseudo_bytes(bp, padding);

		r = ssl3_write_bytes(s, TLS1_RT_HEARTBEAT, buffer, 3 + payload + padding);

		if (r >= 0 && s->msg_callback)
			s->msg_callback(1, s->version, TLS1_RT_HEARTBEAT,
				buffer, 3 + payload + padding,
				s, s->msg_callback_arg);

		OPENSSL_free(buffer);

		if (r < 0)
			return r;
		}
	else if (hbtype == TLS1_HB_RESPONSE)
		{
		unsigned int seq;

		/* We only send sequence numbers (2 bytes unsigned int),
		 * and 16 random bytes, so we just try to read the
		 * sequence number */
		n2s(pl, seq);

		if (payload == 18 && seq == s->tlsext_hb_seq)
			{
			s->tlsext_hb_seq++;
			s->tlsext_hb_pending = 0;
			}
		}

	return 0;
	}
```

*Listing 4: openssl-1.0.1/ssl/t1_lib.c tls1_process_heartbeat function*

As shown in "Listing 5: OpenSSL excerpt that builds the heartbeat response message," the heartbeat

response implementation first checks to determine if the received message type is ‘TLS Heartbeat Request’ message and extracts the request payload length. It then allocates memory for the HeartbeatResponse message. The HeartbeatResponse message has a 1 byte of message type to indicate it is the ‘TLS Heartbeat Response’ message and 2 bytes to indicate the payload length. It copies the payload from the HeartbeatRequest message to the HeartbeatResponse message and sends the response message back to the requestor.

```
/* Allocate memory for the response, size is 1 byte
 * message type, plus 2 bytes payload length, plus
 * payload, plus padding
 */
buffer = OPENSSL_malloc(1 + 2 + payload + padding);
bp = buffer;

/* Enter response type, length and copy payload */
*bp++ = TLS1_HB_RESPONSE;
s2n(payload, bp);
memcpy(bp, pl, payload);
bp += payload;
/* Random padding */
RAND_pseudo_bytes(bp, padding);

r = dtls1_write_bytes(s, TLS1_RT_HEARTBEAT, buffer, 3 + payload + padding);
```

*Listing 5: OpenSSL excerpt that builds the heartbeat response message*

Requestor receives the Heartbeat response message and validates it with the original message sent. Thus, OpenSSL Heartbeat request and response implementation ensures that the secured connection between the peers is still alive or not.

### *Data Leakage Leading to Heartbleed*

There is a bug in the above implementation of the Heartbeat reply to the received Heartbeat request message. Heartbeat reply copies the received payload to the Heartbeat response message to verify that the secured connection is still active, without checking if the payload length is same as the length of the request payload data. The line of OpenSSL code with the bug is show in “Listing 6: incorrect memcpy in the code that builds the heartbeat response message.”

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

*Listing 6: incorrect memcpy in the code that builds the heartbeat response message*

The problem here is that the OpenSSL heartbeat response code does not check to make sure that the payload length field in the heartbeat request message matches the actual length of the payload. If the heartbeat request payload length field is set to a value larger than the actual payload, the memcpy code will copy the payload from the heartbeat message and whatever is in memory beyond the end of the payload.  A heartbeat request the payload length can be set to a maximum value of 65535 bytes. Therefore the bug in the OpenSSL heartbeat response code could copy as much as 65535 bytes from the the machine's memory and send it to the requestor.

This bug is illustrated below in “Figure 2: Memory leak.”

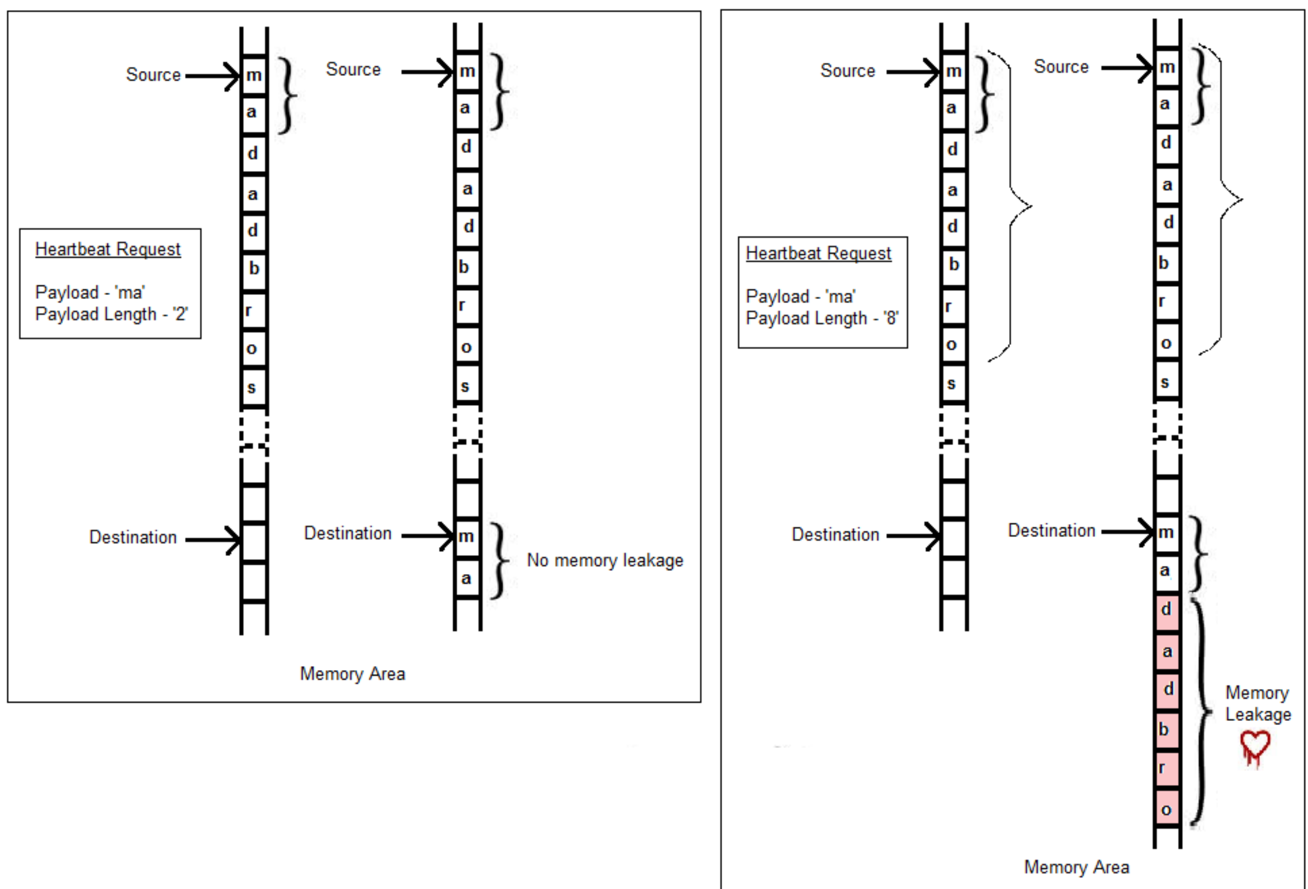


*Figure 2: Memory leak*

“Figure 2: Memory leak” shows that when the request payload data is ‘ma’ and payload length is ‘2’ then ‘memcpy’ works as expected – 2 bytes from source (i.e. ‘ma’) is copied to the ‘destination’ memory area.

But when the request payload data is ‘ma’ and payload length falsely indicates that it is 8 bytes instead of 2, the ‘memcpy’ function copies 8 bytes (i.e. ‘madadbro’) from the ‘source’ memory area to the ‘destination’ memory area. This ‘destination’ data is finally sent to the requestor, causing the memory leak that is now known as the Heartbleed bug.

### *Code Fix*

“Figure 3: The OpenSSL code fix for the Heartbleed bug” shows the change in OpenSSL's file t1_lib.c between version 1.0.1 and OpenSSL version 1.0.1g that was made to fix the Heartbleed bug.

\openssl-1.0.1\ssl\t1_lib.c

```
/* Read type and payload length first */
hbtype = *p++;
n2s(p, payload);
pl = p;

if (s->msg_callback)
    s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT
        &s->s3->rrec.data[0], s->s3->rrec.length,
        s, s->msg_callback_arg);
```

\openssl-1.0.1g\ssl\t1_lib.c

```
if (s->msg_callback)
    s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT,
        &s->s3->rrec.data[0], s->s3->rrec.length,
        s, s->msg_callback_arg);

/* Read type and payload length first */
if (1 + 2 + 16 > s->s3->rrec.length)
    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;
```

*Figure 3: The OpenSSL code fix for the Heartbleed bug*

This code fix has two tasks to perform:

First, it checks to determine if the length of the payload is zero or not. It simply discards the message if the payload length is 0 as shown in “Listing 7: Heartbeat payload zero length check.”

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

*Listing 7: Heartbeat payload zero length check*

The second task performed by the bug fix makes sure that the heartbeat payload length field value matches the actual length of the request payload data. If not, it discards the message. The code excerpt that performs this task is shown in “Listing 8: Heartbeat payload actual length check.”

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

*Listing 8: Heartbeat payload actual length check*

The official notice about the bug was published by the OpenSSL group at [https://www.openssl.org/news/secadv_20140407.txt](https://www.openssl.org/news/secadv_20140407.txt) and is reproduced in “Figure 4: OpenSSL Security Advisory.”

```
OpenSSL Security Advisory [07 Apr 2014]
=========================================

TLS heartbeat read overruns (CVE-2014-0160)
===========================================

A missing bounds check in the handling of the TLS heartbeat extension can be
used to reveal up to 64k of memory to a connected client or server.

Only 1.0.1 and 1.0.2-beta releases of OpenSSL are affected including
1.0.1f and 1.0.2-beta1.

Thanks for Neel Mehta of Google Security for discovering this bug and to
Adam Langley <agl@chromium.org> and Bodo Moeller <bmoeller@acm.org> for
preparing the fix.

Affected users should upgrade to OpenSSL 1.0.1g. Users unable to immediately
upgrade can alternatively recompile OpenSSL with -DOPENSSL_NO_HEARTBEATS.

1.0.2 will be fixed in 1.0.2-beta2.
```

*Figure 4: OpenSSL Security Advisory*

# 3 The real-world impact of Heartbleed

By exploiting the Heartbleed vulnerability, an attacker can send a Heartbeat request message and retrieve up to 64 KB of memory from the victim's server. The contents of the retreived memory depends on what's in memory in the server at the time, but could potentially contain usernames, passwords, session IDs or secret private keys or other sensitive information. Following figure illustrates how an attacker can exploit this vulnerability. This attack can be made multiple times without leaving any trace of it. "Figure 5: Exploiting the Heartbleed vulnerability" illustrates how an attacker can exploit the Heartbleed vulnerability.

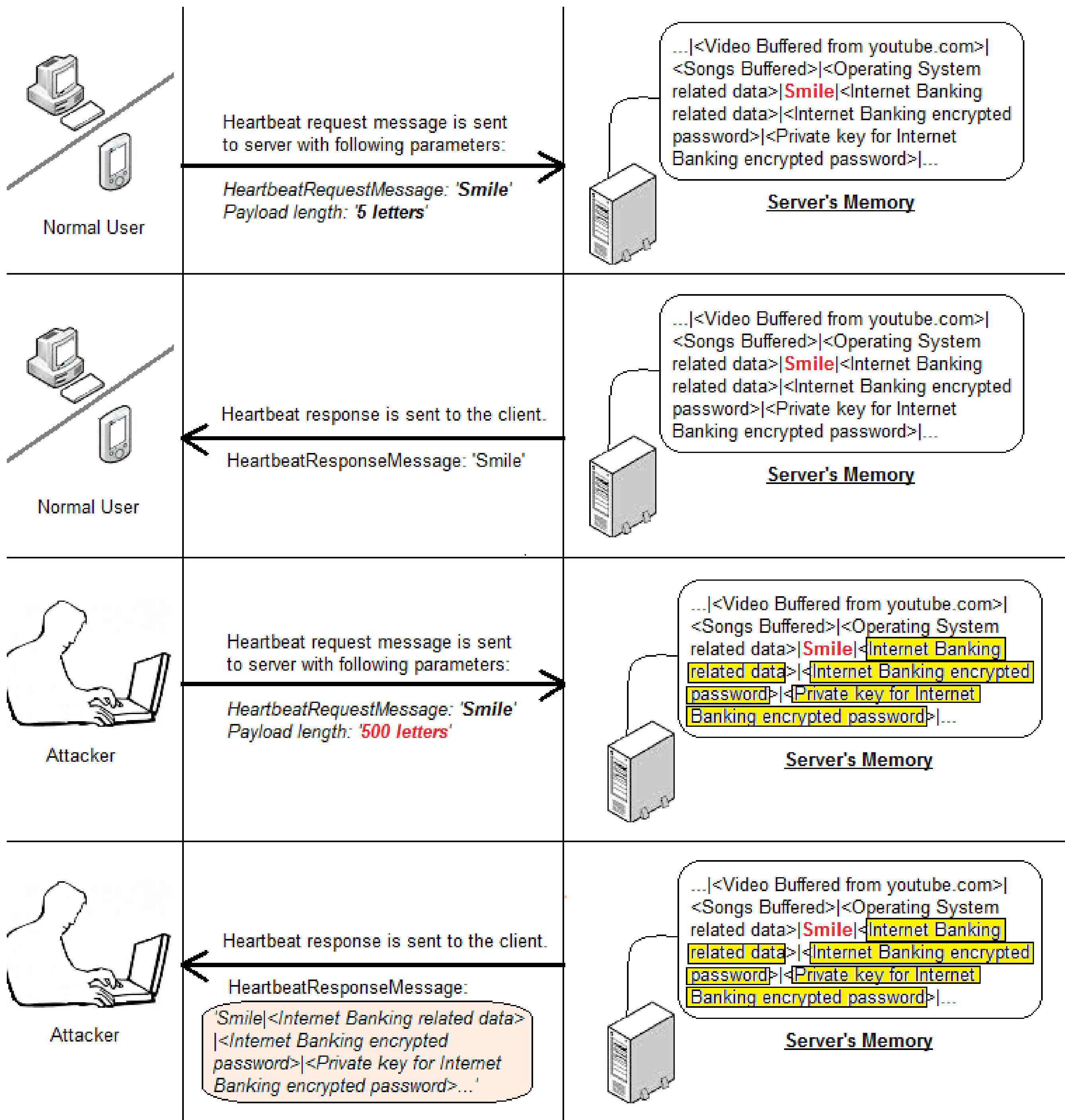


*Figure 5: Exploiting the Heartbleed vulnerability*

It is little early to estimate the impact of this vulnerability, but no one can deny that this scenario is an important one for Internet users, potentially putting their private, secret and encrypted data at risk. Bruce Schneier, in his blog (https://www.schneier.com/blog/archives/2014/04/heartbleed.html), has classified the Heartbleed bug as "Catastrophic" and has given it a rating of 11 on the scale of 1 to 10.

The Pew Research Internet Project (http://www.pewinternet.org/2014/04/30/heartbleeds-impact/) states that '*39% of Internet users have changed passwords or cancelled accounts; 6% think their personal information was swiped*'.

***Affected devices***

To add more on that, Heartbleed has not only affected the 'web' but also the embedded devices. Many home routers and operating systems incorporate OpenSSL. Wikipedia has collected reports of affected devices. Some of these devices are:

- Android smartphones running version 4.1.1 (Jelly Bean) of Android.
- Cisco routers.
- Juniper routers.
- Western Digital My Cloud product family firmware

***Affected Operating Systems***

The website http://heartbleed.com/ maintains a list of affected operatinging systems, some of which include:

- Debian Wheezy (stable), OpenSSL 1.0.1e-2+deb7u4
- Ubuntu 12.04.4 LTS, OpenSSL 1.0.1-4ubuntu5.11
- CentOS 6.5, OpenSSL 1.0.1e-15
- Fedora 18, OpenSSL 1.0.1e-4
- OpenBSD 5.3 (OpenSSL 1.0.1c 10 May 2012) and 5.4 (OpenSSL 1.0.1c 10 May 2012)
- FreeBSD 10.0 - OpenSSL 1.0.1e 11 Feb 2013
- NetBSD 5.0.2 (OpenSSL 1.0.1e)
- OpenSUSE 12.2 (OpenSSL 1.0.1c)

# 4 Heartbleed resolutions, precautions and preventions

All Heartbleed-vulnerable systems should immediately upgrade to OpenSSL 1.0.1g. If you are not sure whether an application you want to access is Heartbleed vulnerable or not - try any one of the Heartbleed detector tools from section "Heartbleed detector tools." No action required if your application is not vulnerable. But if the application is vulnerable, wait for it to be patched with OpenSSL 1.0.1g. Once the patch is applied, all the users of such applications should follow the application's release documents from the service providers. Typically, steps to follow once the patch is applied are:

- changing your password
- generating private keys again
- certificate revocation and replacement

An important step is to restart the services that are using OpenSSL (like HTTPS, SMTP etc).

Before accessing any SSL/TLS application such as HTTPS, check to see if the application is vulnerable. Do not access or login to any affected sites. Ensure all such vendors or enterprises related to

your business have applied this security patch. Keep your eyes open on such news of security vulnerabilities.

The Heartbleed bug has shaken the Internet community on its dependency on the open source software. Even though OpenSSL is a very popular library, it was not properly scrutinized. One reason might be because of lack of resources and funds. The organizations and developers using open source software should contribute back to these open source communities in terms of donations, reviewing the code, testing and designing. Amazon, Facebook, Google have recently come forward to donate funds to improve open-source security systems (Source: http://www.csmonitor.com/Innovation/2014/0424/Major-tech-companies-back-Heartbleed-prevention-measure).

## 4.1 Immunize your application from Heartbleed

To obtain the fix in your application simply upgrade to OpenSSL 1.0.1g.

If upgrading is not practical, you can rebuild your current version of OpenSSL from source without TLS Heartbeat support by adding the following compile switch:

```
-DOPENSSL_NO_HEARTBEATS
```

This switch ensures that the defected code never gets executed.

## 4.2 Does this resolve all the problems?

No, not at all. This is the scariest part of the OpenSSL Heartbleed bug is that,even after taking these measures, no one can completely relax. This vulnerability has existed for more than 2 years. No one knows if their application has been exploited because the attack leaves no traces of it. There is a possibility that attackers might have been reading passwords, secret keys and other encrypted data. This theft can not be known unless the misuse of the data is observed or the attacker discloses it.

## 4.3 Heartbleed detector tools

The following list of tools may help you detect whether a website is vulnerable to Heartbleed:

- https://filippo.io/Heartbleed/
- http://csc.cyberoam.com/cyberoamsupport/webpages/webcat/2014-0160.jsp
- http://news.netcraft.com/archives/2014/04/08/half-a-million-widely-trusted-websites-vulnerable-to-heartbleed-bug.html
- http://possible.lv/tools/hb/
- http://heartbleed.criticalwatch.com/
- https://blog.lookout.com/blog/2014/04/09/heartbleed-detector/
- https://pentest-tools.com/vulnerability-scanning/openssl-heartbleed-scanner/#
- https://lastpass.com/heartbleed/
- http://www.tripwire.com/securescan/?home-banner/
- http://www.arbornetworks.com/asert/2014/04/heartbleed/
- https://www.ssllabs.com/ssltest/index.html

# 5 Immediate aftermath of Heartbleed for different stakeholders

The Heartbleed bug affects many different stakeholders:

**Developers**

The immediate action for developers is to upgrade their application to OpenSSL 1.0.1g. If not possible they should disable OpenSSL Heartbeat by recompiling OpenSSL as described in "Immunize your application from Heartbleed." using following option:

**System Administrators**

System administrator should ensure that no impacted certificates could be reused. All impacted certificates should be revoked and replaced. Restart all such vulnerable services after applying patches. Users should be required to change the passwords after the patch has been applied.

***User**s*

Do not access any vulnerable sites. Check it using any Heartbleed detector tools. Follow the released document of the patched sites before using their application.

***Organizations and Service Providers using OpenSSL***

The damage caused by this vulnerability could not be traced. So, organizations should presume the worst and prepare themselves accordingly. They should be prepared if attacker has already got access to their secured data. They should also apply these patches and provide a '*to-do*' document for their users.

# 6 Conclusion

Heartbleed is a big stain on today's fast moving technology world. It is time to halt a little bit and do some introspection. Are we running too fast but forgot to tie our shoelaces? We can not afford even such a minute mistake.

Nothing has changed and the world will move on, but there is a big question mark on the trust this security vulnerability has broken. It will be hard to close this trust gap as Heartbleed will always remind us. Only time will tell how much actual damage it has caused, since it existed for more than two years. Nevertheless, it's about owning more responsibility towards creating more secured system by industry, organizations, developers and the open source community.

# 7 References


https://www.openssl.org/about/

http://www.webopedia.com/TERM/H/HTTP.html

https://tools.ietf.org/html/rfc5746#page-3

http://www.tutorialspoint.com/c_standard_library/c_function_memcpy.htm

https://www.openssl.org/news/secadv_20140407.txt

http://news.netcraft.com/archives/2014/04/02/april-2014-web-server-survey.html

http://www.pewinternet.org/2014/04/30/heartbleeds-impact/

https://www.schneier.com/blog/archives/2014/04/heartbleed.html

http://www.christiantoday.com/article/android.jelly.bean.phones.still.vulnerable.to.heart.bleed.bug/36814.htm

http://en.wikipedia.org/wiki/Heartbleed

http://mashable.com/2014/04/24/facebook-google-microsoft-join-forces-to-prevent-another-heartbleed/

https://www.ssllabs.com/ssltest/index.html

https://www.thesslstore.in/support/openssl-heartbleed.aspx

http://www.cnet.com/news/how-to-protect-yourself-from-the-heartbleed-bug/

http://www.forbes.com/sites/josephsteinberg/2014/04/10/massive-internet-security-vulnerability-you-are-at-risk-what-you-need-to-do/2/

http://www.huffingtonpost.com/morgan-reed/the-trust-gap-heartbleed-_b_5170194.html

http://www.publicsafety.gc.ca/cnt/rsrcs/cybr-ctr/2014/al14-005-eng.aspx

http://blog.csdn.net/fanbird2008/article/details/18623141

## 8 About Author

Bipin Chandra has extensive experience in application development with a focus on Java/J2EE and middleware technologies. He has domain experience in B2B, finance and banking, and investment. He specializes in web technologies and is highly skilled at presentation, estimation, development, and problem discovery. Mr. Chandra is a practitioner of Agile development methodologies and holds SCJP, SCWCD, and SCDJWS Java certifications. He is currently working for IBM Software Labs. His main research interests are data analytics and mobile computing.