The OpenSSL 'HEARTBLEED' vulnerability – A CASE STUDY

IT20059422 WICKRAMASINGHE W A N
Department of Computer Systems Engineering Sri Lanka Institute of Information Technology
New Kandy Road Malabe 10115 Sri Lanka
It20059422@my.sliit.lk

Abstract **- The "OpenSSL Heartbleed Vulnerability" is explained in length on this page. It examines the 'HeartBleed' phenomena, including its causes and consequences. The goal of this paper is to increase awareness of the Heartbleed vulnerability in the OpenSSL library. As a result of this flaw, attackers can acquire access to passwords, private keys, and any encrypted data. It also explains how Heartbleed works, identifies the code that causes data leaking, and shows how the problem may be fixed by modifying the code.**

Key words – HeartBleed, OpenSSL, Encrypt, Illustrate, Origin, Leakage

I. INTRODUCTION

The title "Heartbleed" appropriately conveys the security issue in and of itself - the term "bleed" refers to data loss, while the name "Heartbleed" derives from the Heartbeat protocol. This indicates that the implementation of the Heartbeat protocol, specifically the OpenSSL-provided implementation, has a data leak. The OpenSSL library is used to implement the Secure Socket Layer (SSL) and Transport Layer Security (TLS) protocols. It is extensively used and is free to download as open source software. In this white paper, we'll understand how critical this vulnerability is, as well as the several measures we may employ to assist prevent the loss of sensitive data.

DTLS (Datagram Transport Layer Security) is a communication protocol that implements TLS over an insecure transport protocol, such as Datagram Congestion Control Protocol (DCCP) or User Datagram Protocol (UDP) (UDP).

The heartbeat is an addition to the TLS/DTLS protocol that is used to determine whether or not a connection established between two TLS/DTLS-enabled communication devices is still "alive," or able to communicate. RFC 6520 (http://tools.ietf.org/html/rfc6520) was the document that started its use in 2012. The Heartbeat protocol, which is specified in the RFC, runs on top of the TLS Record Layer and needs the peers to communicate in the form of a "heartbeat" to keep the connection between them alive. The present TLS/DTLS renegotiation mechanism for determining whether or not a peer is still alive was time-consuming and costly, prompting the development of the heartbeat extension.

shattered people's faith in the open-source software community and placed doubt on the Internet's overall security.

The OpenSSL library implements a number of cryptographic protocols, including SSL and TLS, among others. It is open-source software that was created in the C programming language and is available to the general public. The library's development is entirely driven by volunteers, and owing to an open-source license styled after Apache's, anybody can use it for free, both for commercial and non-commercial applications. There are several various ways that can be used to ensure internet security. The network layer's security is one of them. One option to increase the security provided by TCP/IP is to use the Secure Socket Layer (SSL) or its follow-on protocol, Transport Layer Security (TLS). Although SSL and TLS are two separate protocols, they are usually referred to as SSL/TLS.

The Hypertext Transfer Protocol, or HTTP, is a stateless application level protocol used by web servers and browsers to prepare and transport data. Because of the increasing number of risks and cons carried out over the internet, there is a constant need to improve data transmission security. HTTPS increases the secrecy and integrity of communications across a network by introducing an SSL/TLS layer between the TCP layer and the HTTP layer.

I. HOW THE HEARTBEAT WORKS – *Technological overview*

The heartbeat extension protocol is made up of two types of messages: HeartbeatRequest message and HeartbeatResponse message. The extension protocol depends on which TLS protocol is being used, as described below.

• When using the reliable transport protocol, one side of the peer connection sends a HeartbeatRequest message to the other side. The other end of the link should send a HeartbeatResponse message right away. This makes one Heartbeat work, so the connection stays alive. This is called "keep-alive" functionality. If there is no response within a certain amount of time, the TLS connection is broken.

• Unreliable transport protocol: One side of the peer connection sends a HeartbeatRequest message to the other side. The other end of the link should send a HeartbeatResponse message right away. If there is no response in the timeout period, another HeartbeatRequest message is sent. If the expected response doesn't come back after a certain number of retries, the DTLS connection is broken.

When a HeartbeatRequest message is received, the HeartbeatResponse message should be an exact copy of the HeartbeatRequest message. The sender checks to see if the HeartbeatResponse message is the same as the message that was sent. If it's the same, the link stays alive. If the response doesn't have the same message as the request, the HeartbeatRequest message is sent again for a certain number of times.

II.I. Heartbeat Implementation in OpenSSL

In December 2011, the OpenSSL team implemented the heartbeat feature. This section describes briefly the code for implementing heartbeat for both HeartbeatRequest and HeartbeatResponse messages. It also describes the coding problem and its cure in depth. Download the OpenSSL source code from the group's website at https://www.openssl.org/source/ or ftp:/ftp.openssl.org/source/. The problem affects OpenSSL versions 1.0.1 through 1.0.1f.

Sending Heartbeat Requests

```
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

A code snippet from dtls1 heartbeat says that the total size of the payload and padding must not be more than 16381 bytes. Here, the maximum size of a heartbeat request is 16KByte, or 16384 bytes. This includes one byte that says this is a TLS Heartbeat request message and two bytes that say this is a heartbeat request.

The length of the Heartbeat request message is set in bytes. So, "payload and padding" be no bigger 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

OpenSSL's implementation of the HeartbeatRequest message includes a Message Type of 1 byte to indicate that this is a 'TLS Heartbeat Request' message, 2 bytes for the payload length, a 2 byte sequence number in the payload to identify the specified number of messages sent before a timeout, and 16 bytes for the actual payload and any padding. The code that generates this message is displayed in Listing 3: OpenSSL code that generates 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 generated and transmitted to the recipient. The countdown for timeout begins and the number of retransmissions selected is adjusted. No problem exists with the OpenSSL Heartbeat request.

Heartbeat Response

Heartbeat response transmits a copy of the Heartbeat request payload data, which confirms that the secure connection between the peers is still active, as demonstrated by the OpenSSL code 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

The response implementation first determines whether the message received is a 'TLS Heartbeat Request' message before extracting the request payload length. After that, memory is allocated to the HeartbeatResponse message. The HeartbeatResponse message has two bytes of payload length and one byte of message type to indicate that it is the 'TLS Heartbeat Response' message. After copying the payload from the HeartbeatRequest message to the HeartbeatResponse message, it 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

The Heartbeat response message is received by the requestor, who compares it to the original message delivered.

As a result, the implementation of OpenSSL Heartbeat request and response assures that the secured connection between the peers is still alive or not.

Data Leakage Leading to Heartbleed

There is a bug in the way the Heartbeat reply to the Heartbeat request message was set up above. Heartbeat reply copies the received payload into the Heartbeat response message to make sure the secure connection is still working. It doesn't check to see if the length of the payload is the same as the length of the request payload data. "Listing 6: improper memcpy in the code that creates the heartbeat response message" shows the line of OpenSSL code with the fault.

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

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

The problem is that the OpenSSL heartbeat response code does not check to see if 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 number that is bigger than the actual payload, the memcpy code will copy the payload from the heartbeat message and whatever is in memory after the end of the payload. The length of the payload for a heartbeat request can be set to a maximum of 65535 bytes.

So, the bug in the OpenSSL heartbeat response code could copy up to 65535 bytes from the machine's memory and send them to the requestor.

"Figure 2: Memory leak" shows a picture of this bug.

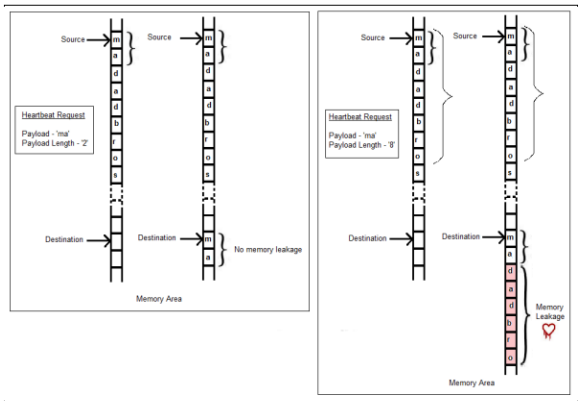


Figure 2: Memory leak

"Figure 2: Memory Leak" indicates that when the request payload data is'ma' and the payload length is'2','memcpy' works as intended. Two bytes from the source ('ma') are copied to the 'destination' memory location in this scenario.

The "memcpy" function copies 8 bytes (i.e. "madadbro") from the "source" memory area to the "destination" memory area when the request payload data is "ma" and the payload length is mistakenly reported as 8 bytes instead of 2. This "destination" data is eventually transferred to the requestor, resulting in the Heartbleed problem, also known as a memory leak.

Code Fix

"Figure 3: The Heartbleed bug fix in OpenSSL code" shows the change that was made to the file t1 lib.c between OpenSSL versions 1.0.1 and 1.0.1g to fix the Heartbleed bug.

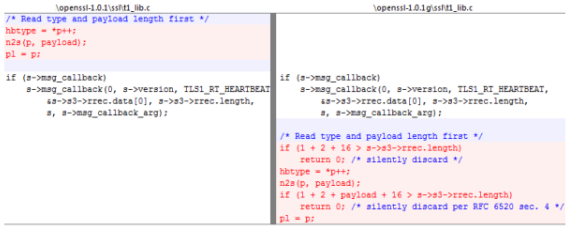


Figure 3: The OpenSSL code fix for the Heartbleed bug

II. THE REAL-WORLD IMPACT OF HEARTBLEED

By exploiting the Heartbleed vulnerability, an attacker can send a Heartbeat request message to the victim's server and access up to 64 KB of memory. Usernames, passwords, session IDs, secret private keys, and other sensitive data may be found in the recovered memory, depending on what was in the server's memory at the time. The figure below shows how an attacker could take advantage of this weakness. This attack can be carried out forever with no repercussions. The Heartbleed vulnerability is depicted in "Figure 5: Exploiting the Heartbleed Vulnerability," which shows how an attacker could exploit it.

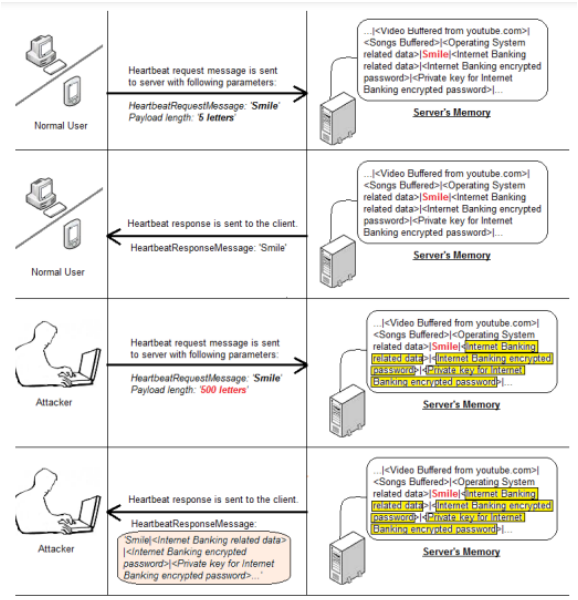
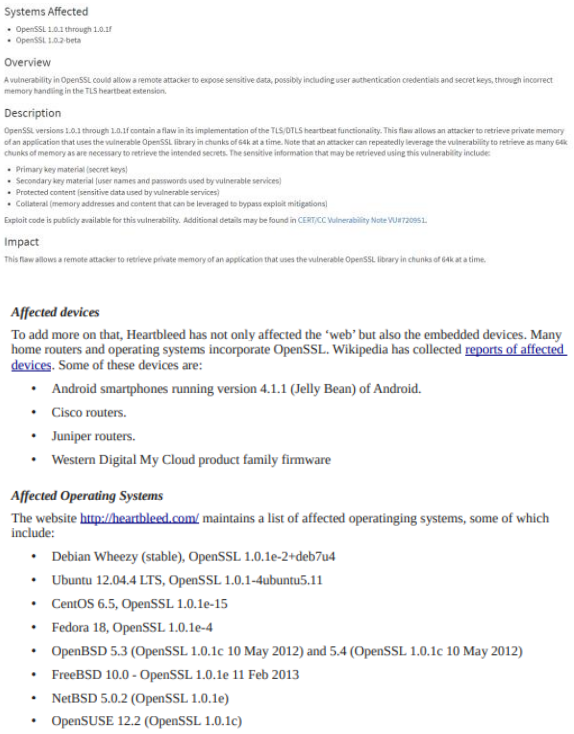


Figure 5: Exploiting the Heartbleed vulnerability

A updated CVE/CWE for the Vulnerability

Alert (TA14-098A)

OpenSSL 'Heartbleed' vulnerability (CVE-2014-0160)



Systems Affected

- OpenSSL 1.0.1 through 1.0.1f
- OpenSSL 1.0.2-beta

Overview

A vulnerability in OpenSSL could allow a remote attacker to expose sensitive data, possibly including user authentication credentials and secret keys, through incorrect memory handling in the TLS heartbeat extension.

Description

OpenSSL versions 1.0.1 through 1.0.1f contain a flaw in its implementation of the TLS/DTLS heartbeat functionality. This flaw allows an attacker to retrieve private memory of an application that uses the vulnerable OpenSSL library in chunks of 64k at a time. Note that an attacker can repeatedly leverage the vulnerability to retrieve as many 64k chunks of memory as are necessary to retrieve the intended secrets. The sensitive information that may be retrieved using this vulnerability include:

- Primary key material (secret keys)
- Secondary key material (user names and passwords used by vulnerable services)
- Protected content (sensitive data used by vulnerable services)
- Collateral (memory addresses and content that can be leveraged to bypass exploit mitigations)

Exploit code is publicly available for this vulnerability. Additional details may be found in CERT/CC Vulnerability Note VU#720951.

Impact

This flaw allows a remote attacker to retrieve private memory of an application that uses the vulnerable OpenSSL library in chunks of 64k at a time.

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)

Affected devices

• 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

• 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)

Heartbleed resolutions, precautions and preventions

All systems that are vulnerable to Heartbleed should update right away to OpenSSL 1.0.1g. If you're not

You can use any of the tools in the "Heartbleed detecting tools" section to see if an application you want to utilize is vulnerable to Heartbleed. You don't need to do anything if your application isn't susceptible. If the application is insecure, however, you should wait till OpenSSL 1.0.1g is installed to remedy it. After installing the patch, all users of these apps should follow the directions in the service providers' release documentation. Following the application of the patch, you should normally take the following steps:

• 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).

III. Heart bleed bug update: smartphones running on android jelly bean 4.1.1 affected

Google says that all versions of Android are safe from the Heartbleed bug, except for Jelly Bean 4.1.1.

Google refers to the Jelly Bean flaw as a "limited exception" on its blog. Android 4.1.1 is the version that has the vulnerability, and it is used by less than 10% of active smartphones. However, because there are over 900 million Android devices in circulation worldwide, the OpenSSL weakness impacts tens of millions of people. In July 2012, the most popular version of Android, Jelly Bean, was released. There were several versions available till October 2013, ranging from 4.1 to 4.3.1. Only version 4.1.1 is affected, and "patching information" is being supplied to cellular carriers and phone manufacturers so they can address it. These Android partners are in responsible of updating Android software, which slows things down.

To find out what version of Android your phone is running, go to "Settings" and then "About phone." Lookout, a mobile security app, allows users to verify if their Android version is susceptible. The "bug," dubbed "Heartbleed," is a hole in OpenSSL's cryptographic software that allows SSL/TLS encryption to be used against computer users. The "https" protocol, which is designed to indicate that a website is secure, instead informs hackers that the site is vulnerable. The hackers can then use this

information to mislead a computer's server into delivering data that is already stored in its memory. Google security researcher Neel Mehta discovered Heartbleed initially, and internet security firm Codenomicon validated the flaw. Researchers found the Heartbleed bug in OpenSSL two years ago, which is alarming. We have no way of knowing if there have been any attacks because the software weakness leaves no trace.

Vendors and service providers must apply the Fixed OpenSSL software, which was published on Monday, to remove Heartbleed's grip on the server. According to Codenomicon, "operating system and distribution distributors, appliance vendors, and independent software vendors must embrace the update and notify their consumers." "As soon as the fix for their operating systems, networked appliances, and software becomes available, service providers and consumers must install it."QUESTIONARIES

IS HEARTBLEED A VIRUS?

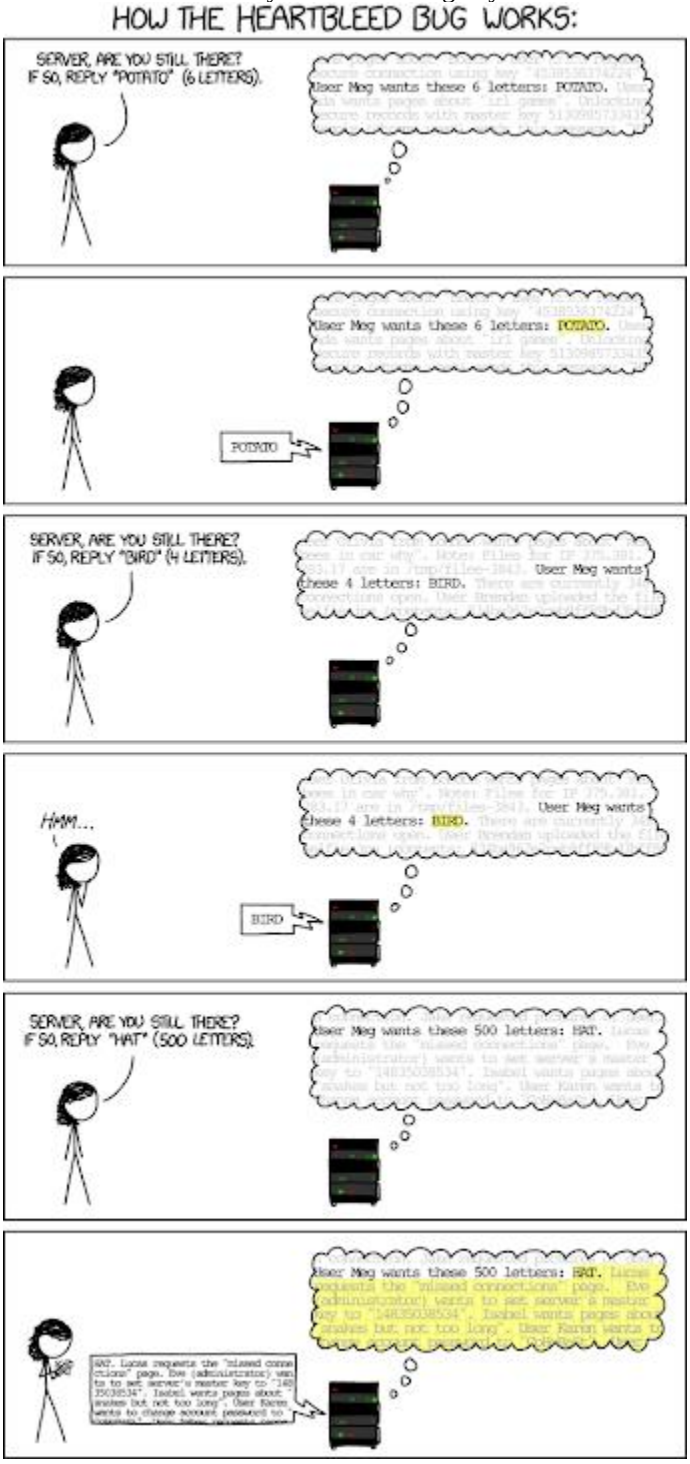
It is, without a doubt, not a virus. The "Heartbleed Bug" is a vulnerability in the Transport Layer Security (TLS) heartbeat mechanism used by some OpenSSL distributions, a widely used encryption library.

HOW IT WORKS?

Your computer must send 'heartbeats' to the server in order for SSL to work. These heartbeats inform the server that the client (computer) is online (alive).

The Heartbleed attack lets an attacker to obtain a 64-kilobyte block of memory from a vulnerable server by sending a malicious heartbeat, and the number of attacks that may be carried out is unlimited. [Technically described by Rahul Sasi on Garage4hackers]

It enables cyber criminals to recover sensitive data from the server's memory without leaving any traces.



HEARTBLEED ATTACK RELIES ON MAN-IN-THE-MIDDLE ATTACK?

No, a Man-in-the-Middle (MitM) attack is not addressed. One can obtain the secret encryption key

for an SSL/TLS certificate and create a bogus website that passes security checks by exploiting the Heartbleed vulnerability.

An attacker may decrypt communication travelling between a client and a server, resulting in a flawless man-in-the-middle attack for HTTPS connections.

IS IT A CLIENT SIDE OR SERVER SIDE VULNERABILITY?

Attackers can send TLS heartbeats from either side of a TLS connection, allowing them to target both clients and servers. A server or client that uses a Heartbleed-vulnerable OpenSSL implementation can give an attacker up to 64K RAM (CVE-2014-0160).

The Heartbleed Bug, according to researchers, has infected two-thirds of the world's servers, or half a million, including websites, email, and instant messaging apps.

HOW HEARTBLEED AFFECTS SMARTPHONES?

The smartphone is the best example of client-side attacks.

Although all versions of Android OS have outdated versions of the OpenSSL library, only Android 4.1.1 Jelly Bean has the vulnerable heartbeat feature enabled by default. Although Blackberry has confirmed that certain of its products are vulnerable to the Heartbleed problem, Apple's iOS devices are not affected by the OpenSSL flaw.

Despite the fact that Google has patched the impacted version of Android 4.1.1, the latest Android version will take a long time to reach end users, as most smartphone manufacturers and cellular carriers limit updates. Hackers will surely exploit this public information until users running the impacted versions are exposed to the attacks.

WHAT ELSE COULD BE VULNERABLE TO HEARTBLEED?

Because updates from Google's Android partners are unlikely to arrive soon, IP phones, routers, medical equipment, smart TV sets, embedded devices, and millions of other devices that rely on OpenSSL for secure communications may also be vulnerable to the Heartbleed bug.

Critical infrastructure companies (such as energy, utilities, and financial services industries) were also warned by Industrial Control Systems-CERT to tighten up their systems in order to protect themselves from Heartbleed assaults yesterday.

WHO IS RESPONSIBLE FOR HEARTBLEED?

We can't really fault any developer, especially those that contribute to Open Source projects without expecting anything in return.

Dr. Robin Seggelmann, a 31-year-old German developer who added the Heartbeat concept to OpenSSL on New Year's Eve 2011, believes that the "Heartbleed" vulnerability was caused unintentionally by a programming error in the code.

"Unfortunately, I missed validating a variable with a length in one of the new features," went unreported for over two years by code reviewers and everyone else. He admitted, 'I did it unintentionally.'

WHO HAS EXPLOITED THIS BUG YET?

The National Security Agency (NSA) had been aware of the Heartbleed problem for two years, according to Bloomberg. Not only that, but the government was allegedly utilizing it to gather information rather than disclosing it to the OpenSSL developers on a regular basis, according to the article. If this is the fact, it would be one of the most significant breakthroughs in wiretapping history. The National Security Agency, on the other hand, denied this, claiming that it was unaware of Heartbleed until it became public.

When it comes to exploiting any known weakness, hackers, on the other hand, are most likely to be at the top of the list. Because the flaw was so pervasive that it affected half a million websites worldwide, fraudsters were able to get access to the sites after it was made public and steal credentials, passwords, and other information before the site owners applied the freely available remedy.

For the Heartbleed bug, there are several proof-of-concept exploits available:

Python Script
Metasploit Module
C Code
NMAP script
Python Script by Rahul Sasi

CHANGING ACCOUNT PASSWORDS CAN SOLVE THE ISSUE?

Not quite, because the Heartbleed bug can expose anything on the server, including passwords, credit card numbers, and other personal data. To be safe, you should change your passwords right away for both the sites that corrected the problem and the sites that were not affected by the flaw.
To begin, check to see if the websites you frequent on a regular basis are affected by the Heartbleed bug or if they use any of the following services or apps: If you see a red alert, you should exit the site for the time being.

- Provensec Scanner: https://filippo.io/Heartbleed/
- SSL Configuration Checker by GlobalSign
- Checker for ADTsys
- Chromebleed, a new add-on for the Chrome browser built by security researcher Jamie Hoyle, is the easiest method to stay protected.
- You can use the Bluebox Heartbleed Scanner, which is available on the Google Play Store, to see if your Android devices are safe. The Bluebox Heartbleed Scanner looks for apps on your smartphone that have included their own version of OpenSSL, as well as the library's version and whether or not heartbeat is enabled.

WHAT SHOULD I DO TO PROTECT MYSELF?

First and foremost, DO NOT EXTREMELY If your password was previously vulnerable, you must update it everywhere to guarantee that you are now protected. However, bear with me... If specific websites remain vulnerable, your efforts will be in futile, as it is the site's job to fix the problem as soon as possible, because changing the password before the problem is fixed may compromise your new password as well.
If you run an SSL service that is susceptible, you should take precautions.
the following steps:

- Upgrade to version 1.0.1g of OpenSSL.
- Request that the present SSL certificate be revoked.
- Replace your private key with a new one.
- The SSL certificate should be requested and replaced.

Use two-factor authentication, which means that in addition to the password, the account requires a freshly issued pass code that only shows on your personal smartphone to gain access to specific sites.

Nobody knows for sure at this point because Heartbleed is undetectable because it leaves no trail, and the situation is rapidly worse.
It's conceivable you'll never know whether you've been hacked with the bug. This means you won't be able to tell if your information was taken from a site or service before it was corrected.
Your passwords and financial information are still accessible to criminals and other eavesdropping authorities if you haven't updated your passwords on popular sites.

If you own an SSL service that is vulnerable, you should take precautions.

IV. ACKNOWLEDGEMENT

Would like to thank our Senior lecturer, Mr. Kavinga Yapa and the co-lecturer Miss Chethana Liyanapathirana, for their valuable advice, comments, and recommendations over the course of the project. I'd want to express my gratitude to all of the other lecturers in the department of IT for contributing critical expertise and assisting with the completion of this review paper in addition

Finally, I would like to thank all the unnamed parties who helped me in various ways for the successful completion of this assignment."

V. CONCLUSION

Heartbleed is a major issue in today's fast-paced technological world. It's time to take a breather and reflect about things. Was it possible that we were driving too fast and forgot to fasten our shoes? Even a minor blunder is too much for us to bear.
Nothing has changed, and the world will continue to turn, but this security weakness has shattered the trust of many people. Heartbleed will serve as a constant reminder of how difficult it will be to bridge the trust gap. Only time will tell how much damage it has caused since it has been around for more than two years. Nonetheless, it is about industry, organizations, developers, and the open source community assuming greater responsibility for creating better secure systems.

VI. REFERENCES

1. D. E. 3rd. Transport Layer Security (TLS) Extensions: Extension.2011.IETF RFC6066.
2. Alexa Top 1 Million Domains. http://s3.amazonaws.com/alexastatic/tsv.zip.
3. E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid. Recommendation for Key Management -- Part 1: General (Revision 3), 2012. NIST Special Publication 800--57.
4. BotanSSLLibrary.http://botan.randombit.net .
5. CERT Vulnerability Note VU\#720951: OpenSSL TLS heartbeat extension read overflow discloses sensitive information. http://www.kb.cert.org/vuls/id/720951.
6. D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. IETF RFC-5280, May 2008.
7. R. Duncan. How certificate revocation (doesn't) work in practice, 2013. http://news.netcraft.com/archives/2013/05/13/howcertificaterevocationdoesntworkiticml.
8. Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman. Analysis of the HTTPS certificate ecosystem. In ACM InternetMeasurement Conference(IMC),2013.
9. Z. Durumeric, J. Kasten, F. Li, J. Amann, J. Beekman, M. Payer, N. Weaver, J. A. Halderman, V. Paxson, and M. Bailey. The matter of Heartbleed. In ACM InternetMeasurement Conference(IMC),2014.
10. Z. Durumeric, E. Wustrow, and J. A. Halderman. ZMap: Fast Internet-wide Scanning and Its Security Applications.InUSENIX SecuritySymposium,2013.
11. P. Eckersley and J. Burns. An observatory for the SSLiverse. In Defcon 18, 2010. https://www.eff.org/files/DefconSSLiverse.p
12. F. F. Elwailly, C. Gentry, and Z. Ramzan. QuasiModo: Efficient certificate validation and revocation. InPublicKeyCryptography(PKC), 2004.
13. P. Evans. Heartbleed bug: RCMP asked Revenue Canada to delay news of SIN thefts, 2014. http://www.cbc.ca/news/business/heartbleed-bug-rcmp-asked-revenue-canadatodelaynews-of-sin-thefts-1.2609192.
14. Faketime library. http://www.code-wizards.com/projects/libfaketime/.
15. B. Grubb. Heartbleed disclosure timeline: who knew what and when, 2014. http://www.smh.com.au/it-pro/security-it/heartbleed-disclosure-timeline-who-knewwhat-and-when-20140415-zqurk.html.
16. N. Heninger, Z. Durumeric, E. Wustrow, and J. A. Halderman. Mining your Ps and Qs: Detection of widespread weak keys.InUSENIXSecuritySymposium,2012.
17. R. Holz, L. Braun, N. Kammenhuber, and G. Carle. The SSL landscape -- A thorough analysis of the X.509 PKI using active and passive measurements. In ACM InternetMeasurementConference(IMC),2011 .
18. Revocation doesn't work. https://www.imperialviolet.org/2011/03/18/
19. S. Kornexl, V. Paxson, H. Dreger, A. Feldmann, and R. Sommer. Building a time machine for efficient recording and retrieval of high-volume network traffic. In ACM Internet Measurement Conference (IMC),

20. Mac OS X 10.9.2 Root Certificates.http://support.apple.com/kb/H65

21. S. Micali. NOVOMODO: Scalable certificate validation and simplified PKI management. IPKIResearchWorkshop,2002.

22. P. Mutton. Half a million widely trusted websites vulnerable to heartbleed bug, 2014. http://news.netcraft.com/archives/2014/04/08/half-a-million-widely-trusted-websitesvulnerable-to-heartbleed-bug.html.

23. M. Naor and K. Nissim. Certificate revocation and certificate update.InUSENIXSecuritySymposium,1998

.

24. OpenSSL Project. https://www.openssl.org.

VEDIO-

https://drive.google.com/file/d/1r0GHkX6DoEtrh4_ViRtl6I0mx8CrXbSZ/view?usp=sharing