40183863
Ricardo Sánchez Marchand

Assessed Practical 2

Heartbleed

ELE8094 SwA

November 2016
rsanchezmarchand01@qub.ac.uk

Introduction

1. This report would explain the heartbleed vulnerability, in the first part it will contain an overview of o OpenSSL, what should it do and his importance, what is heartbeat and heartbleed.
2. Then it will explain the technical details of the bug and the nature of the fix.
3. At the end the important lessons learnt.

OpenSSL

OpenSSL is an open software based project. It is a fork and a successor of the SSLeay. This project helps to implement in a system Secure Sockets Layers (SSL) , Transport Layer Security (TLS), cryptography, etc.

It is intended to guarantee confidentiality, authenticity, and integrity for communications between a client and the server. It should provide a robust, commercial-grade and full featured toolkit.[1] Is widely used which makes it very attractive for attackers, once you find a vulnerability you can exploit it in a large number of devices. Including for example: Amazon, Github, Pinterest, Reddit, SoundCLoud, SourceForge, Tumblr, Wikipedia, etc.

The most notable software using OpenSSL are the open source web servers like Apache and nginx. The combined market share of just those two out of the active sites on the Internet was over 66% according to Netcraft's April 2014 Web Server Survey. Furthermore OpenSSL is used to protect for example email servers (SMTP, POP and IMAP protocols), chat servers (XMPP protocol), virtual private networks (SSL VPNs), network appliances and wide variety of client side software.[2]
If you or your organization chooses to use this project constant updates are important. Only from February 2015 to December there has been 9 High severity vulnerabilities.

Heartbeat

A handshake is required prior to any exchange of the data that should be protected in a connection, but what happens if we are inactive for a short period and then we want to transfer more data or we are no longer using the connection.

So the handshake is not required too often, or a connection kept alive for too long, we can send a heartbeat. Heartbeats are periodical “I am still here messages please do not shut down the connection”. It can be used in more ways for example to indicate the health of really important software or hardware.

[1] https://www.openssl.org/
[2] http://heartbleed.com/

The message sent can have a payload (for example the time and date or some random data). The server receives the payload and should answer back with the same payload plus some more data. The code then transmits an OpenSSL heartbeat request are:

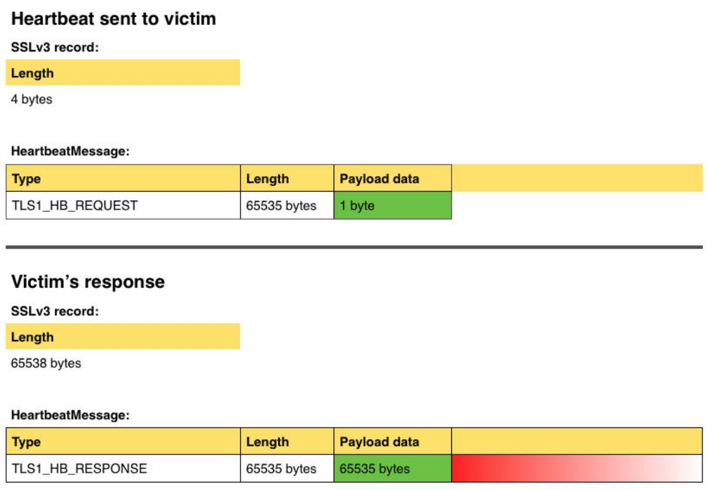
1. The single byte 0x01 (denoting that this is a TLS1_HB_REQUEST).
2. Two bytes containing the 16-bit representation of 34 (size of payload plus padding).
3. Two bytes of payload consisting of a 16-bit sequence number.
4. 16 bytes of random data making up the rest of the 18-byte payload.
5. 16 further random padding bytes, required by the standard.[3]

Technical details

Heartbleed CVE-2014-0160 7th April 2014:

A missing bounds check in the handling of the TLS heartbeat extension can be used to reveal up to 64kB of memory to a connected client or server (a.k.a. Heartbleed). This issue did not affect versions of OpenSSL prior to 1.0.1. Reported by Neel Mehta. Fixed in OpenSSL 1.0.1g (Affected 1.0.1f, 1.0.1e, 1.0.1d, 1.0.1c, 1.0.1b, 1.0.1a, 1.0.1).[4]

It is called heartbleed because it exploits a component of the heartbeat extension (RFC6520).



If for example, an attacker sends 1 byte of information, but it says that he sent 64 kb. The server would read 1 byte (what he received) + all the memory that is after until he reaches the 64k and send everything back. It can retrieve passwords, or any other sensitive information as credit card numbers or private data. Every time an attacker sends a heartbleed, he can receive different information.

Another example is if the attacker sends a 10 byte message but he says that it is 200 in size. OpenSSL would not check if he has 200 he would believe it. If the server start to save data in the slot 1089 (for example). It would read from 1089 to 1099 (the same data) and would keep reading because it would have to retrieve 200. He would stop in the slot 1289 and would send back everything back.

[3] https://nakedsecurity.sophos.com/2014/04/08/anatomy-of-a-data-leak-bug-openssl-heartbleed/
[4] https://www.openssl.org/news/vulnerabilities.html

If you are used to code with high-level programming language, maybe you do not understand this but remember that OpenSSL is written in C and it calls direct slots in memory which is something that a high level programming actively avoids.

Heartbleed Code explanation. [5] [6] [7] [8] [9]

The file with the code vulnerability is mostly in: /ssl/d1_both.c but the declaration of the structure that contains the SSL record is in /ssl/record/record.h as we can see:

```
typedef struct ssl3_record_st {
   int rec_version;
   int type;
   size_t length;
   size_t orig_len;
   size_t off;
   unsigned char *data;
   unsigned char *input;
   unsigned char *comp;
   unsigned int read;
   unsigned long epoch;
   unsigned char seq_num[SEQ_NUM_SIZE];
} SSL3_RECORD;
```

The struct is used in the dtls1_process_heartbeat(SSL *s)

```
 dtls1_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 */
            hbtype = *p++;
            n2s(p, payload);
            pl = p;
```

The n2s(p, payload) takes two bytes from p, and puts them in payload. These bytes are the length of the payload in the heartbeat. The length in record is not checked. After OpenSSL memory is allocated using that same payload length:

```
            unsigned char *buffer, *bp;
            int r;
            buffer = OPENSSL_malloc(1 + 2 + payload + padding);
            bp = buffer;
```

OpenSSL is allocating the amount of memory that the client wants and later in the same function it copy that much memory into the buffer and then would send it back:

[5] https://nakedsecurity.sophos.com/2014/04/08/anatomy-of-a-data-leak-bug-openssl-heartbleed/
[6] https://www.seancassidy.me/diagnosis-of-the-openssl-heartbleed-bug.html
[7] https://wiki.openssl.org/index.php/Manual:Ssl(3)#DATA_STRUCTURES
[8] http://stackabuse.com/heartbleed-bug-explained/
[9] https://git.openssl.org and https://github.com/openssl/openssl

```
*bp++ = TLS1_HB_RESPONSE;
s2n(payload, bp);
memcpy(bp, pl, payload);
```

The nature of the fix

There is code fix and intrusion detection and prevention systems (IDS/IPS) “fix”. Rules can detect malicious heartbeat comparing the size of the request against the size of the reply. This implies that IDS/IPS can be programmed to detect the attack but not to block and is going to be too late.

The code fix on the other hand is very simple and effective avoiding the malicious responses:

```
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;
```

OpenSSL now is checking that the heartbeat is not empty and that the payload and the length matches.

The lessons

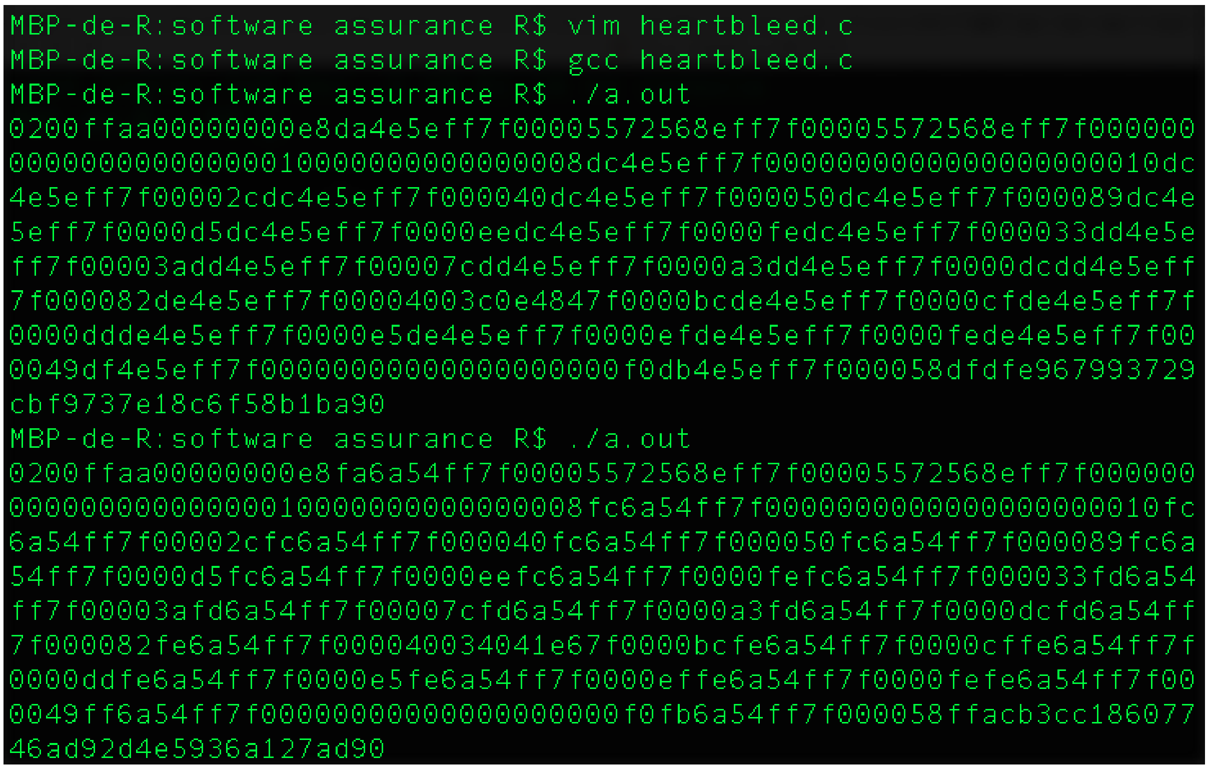
Most of the lessons are very straightforward and obvious, but we keep doing the same mistakes over and over…

1. We should check every input, avoiding insecure interaction between computers and or human with computers. “Do not trust anyone too much” is a good policy.
2. It is amazing how after more than two years there are still heartbleed vulnerable systems. You may think that everyone has fixed but is not that way. In shodan there are still +151,000 vulnerable services.[10]
3. I understand that the OpenSSL is one of the best systems but we should not be installing stuff that we don't fully understand, as the paper: “The Most Dangerous Code in the World” says we should be very careful with what we don´t understand.
4. Have curiosity, if all the competent programmers that installed the code checked it a little bit, the code would be great, but they trust entirely, maybe the curiosity to find out how it works would have fixed this.
5. Understand different programming languages and why they are being used, the pros and the cons. Pointers and memory allocations in this case were very sensitive.
6. Security audits for open codes, for example Core Infrastructure Initiative is a great idea.

[10] https://www.shodan.io/search?query=vuln%3Acve-2014-0160

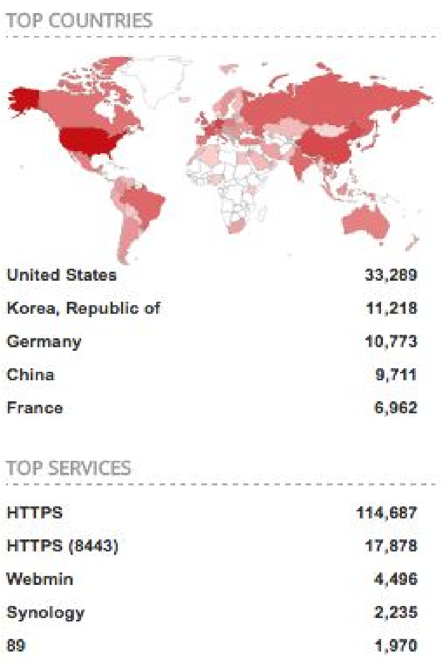
Appendix

Hex dumps/screen shots of output



```
MBP-de-R:software assurance R$ vim heartbleed.c
MBP-de-R:software assurance R$ gcc heartbleed.c
MBP-de-R:software assurance R$ ./a.out
0200ffaa00000000e8da4e5eff7f00005572568eff7f00005572568eff7f000000
000000000000000100000000000000008dc4e5eff7f0000000000000000000010dc
4e5eff7f00002cdc4e5eff7f000040dc4e5eff7f000050dc4e5eff7f000089dc4e
5eff7f0000d5dc4e5eff7f0000eedc4e5eff7f0000fedc4e5eff7f000033dd4e5e
ff7f00003add4e5eff7f00007cdd4e5eff7f0000a3dd4e5eff7f0000dcdd4e5eff
7f000082de4e5eff7f00004003c0e4847f0000bcde4e5eff7f0000cfde4e5eff7f
0000ddde4e5eff7f0000e5de4e5eff7f0000efde4e5eff7f0000fede4e5eff7f00
0049df4e5eff7f00000000000000000000f0db4e5eff7f000058dfdfe967993729
cbf9737e18c6f58b1ba90
MBP-de-R:software assurance R$ ./a.out
0200ffaa00000000e8fa6a54ff7f00005572568eff7f00005572568eff7f000000
000000000000000100000000000000008fc6a54ff7f0000000000000000000010fc
6a54ff7f00002cfc6a54ff7f000040fc6a54ff7f000050fc6a54ff7f000089fc6a
54ff7f0000d5fc6a54ff7f0000eefc6a54ff7f0000fefc6a54ff7f000033fd6a54
ff7f00003afd6a54ff7f00007cfd6a54ff7f0000a3fd6a54ff7f0000dcfd6a54ff
7f000082fe6a54ff7f000040034041e67f0000bcfe6a54ff7f0000cffe6a54ff7f
0000ddfe6a54ff7f0000e5fe6a54ff7f0000effe6a54ff7f0000fefe6a54ff7f00
0049ff6a54ff7f00000000000000000000f0fb6a54ff7f000058ffacb3cc186077
46ad92d4e5936a127ad90
```

Heartbleed vulnerable services:

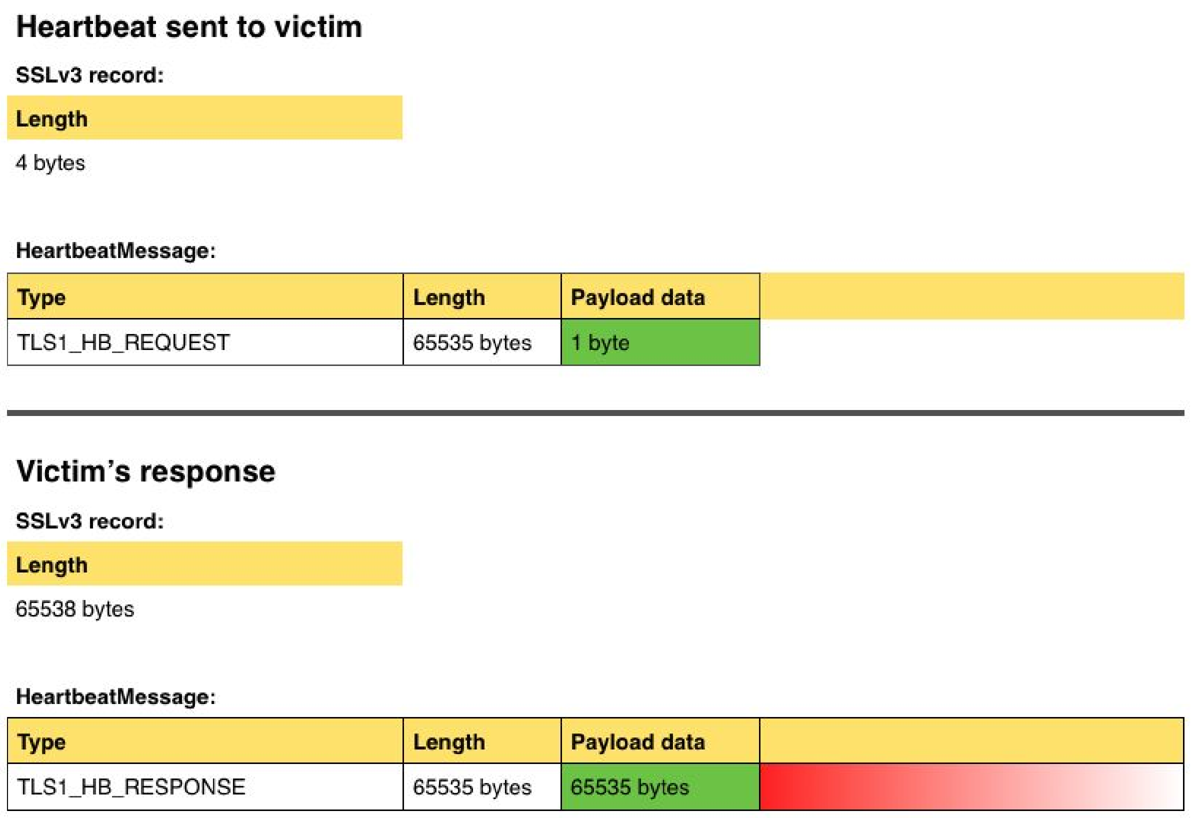


TOP COUNTRIES

United States	33,289
Korea, Republic of	11,218
Germany	10,773
China	9,711
France	6,962

TOP SERVICES

HTTPS	114,687
HTTPS (8443)	17,878
Webmin	4,496
Synology	2,235
89	1,970



Heartbeat sent to victim

SSLv3 record:

Length
4 bytes

HeartbeatMessage:

Type	Length	Payload data	
TLS1_HB_REQUEST	65535 bytes	1 byte	

Victim’s response

SSLv3 record:

Length
65538 bytes

HeartbeatMessage:

Type	Length	Payload data	
TLS1_HB_RESPONSE	65535 bytes	65535 bytes	🟥

[11]

[11] http://www.elladodelmal.com/2014/04/heartbleed-y-el-caos-de-seguridad-en.html