Heartbleed

Giuseppe Caruso, Simone Galota, Matthew Steckman

St. John's University

CSS 1011: Network Security

Professor Denise Dragos

April 25, 2023

Heartbleed

Abstract

The Heartbleed vulnerability, related to the SSL/TLS implementation of the Heartbeat extension provided by the OpenSSL library, is a famous vulnerability, thanks to which, an attacker is able to steal information stored within the server's memory. Confidential and critical information can be stolen thanks to the exploitation. In this paper, after providing a technical background of the Transport Layer Security Protocol, the affected library's code, the heartbeat extension, details about the vulnerability and how the exploitation can be conducted are shown. Furthermore, a practical example is presented by leveraging a well-known tool called Metasploitable and a vulnerable Linux VM in which an Apache2 vulnerable Server is employed. Finally, along with the conclusions, possible mitigations are discussed.

Heartbleed (CVE-2014-0160) [8] is the name given to a popular vulnerability related to the SSL/TLS protocol, and in particular to the Heartbeat extension introduced in the RFC6520. This vulnerability affected the well-known OpenSSL software library, used to implement the underlined secure communication protocol. Classified as Critical, with a CVSS (Common Vulnerability Scoring System) score of 7.5 out of 10.0, allows an attacker to retrieve information from the memory of the system by replying to the attacker's request.

The TLS Protocol

TLS (earlier SSL) is a client-server secure transport protocol. Defined for the first time within the RFC 2246 [1], its latest version is the TLSv1.3, described more in-depth in the RFC 8446 [2]. As stated in the RFCs related to the different versions, different goals are set for this cryptographic protocol [3]:

1. Cryptographic Security: a secure connection is created between the communicating parties.
2. Interoperability: the protocol implementation and the application using such a feature can work without the knowledge of the other.
3. Extensibility: new encryption and public key methods can be added without the need to develop another protocol.
4. Efficiency: since cryptographic functionalities are highly resource-consuming, a session caching scheme is incorporated within the implementation to avoid the re-creation of connections that should remain alive.

The only requirement is to use a reliable transport channel over which this protocol will be layered. This feature is provided by the TCP protocol. A different version of the TLS protocol has been developed, called DTLS, to accommodate the need to use a secure communication protocol over UDP. Thanks to the usage of this secure communication, confidentiality, authentication, and integrity are achieved:

1. Authentication: since it is client-server architecture-based, authentication server side is compulsory meanwhile client side is optional. This feature is obtained thanks to asymmetric cryptography.
2. Confidentiality: after creating the secure channel, the data sent is encrypted, hence only the endpoints are able to understand the communication.
3. Integrity: the protocol provides methods to check the integrity of data exchanged.

One of the drawbacks of the protocol is related to the establishment of new sessions since there is the need to renegotiate the parameters and re-initialize the handshake. The heartbeat extension, as shown in the following sections, was introduced to solve this issue, trying to improve the performance.

OpenSSL

OpenSSL [4] is a cryptographic library that provides an open-source implementation of the SSL/TLS protocol. Applications in which secure communication is needed can leverage this library to achieve this goal without any worries since it is certified by the National Institute of Standards and Technology [5]. In particular, the vulnerability addressed within this paper was found in the OpenSSL implementation of the Heartbeat extension introduced in the TLS protocol [6].

Heartbeat extension [7]

Heartbeat is an extension implemented in the TLS protocol, useful for keeping the connection alive without the need to constantly renegotiate the SSL/TLS session.

The protocol consists of two messages:

- HeartbeatRequest: type field equal to 1.
- HeartbeatResponse: type field equal to 2.

The request is sent out and, successively, the sender is waiting for the response which comes back from the other side. However, no request should be sent during the 3-way handshake.

Another important feature is that there must not be more than one Request in sending at a time. Moreover, a HeartbeatRequest can be used to have the certainty that the peer is alive and reachable.

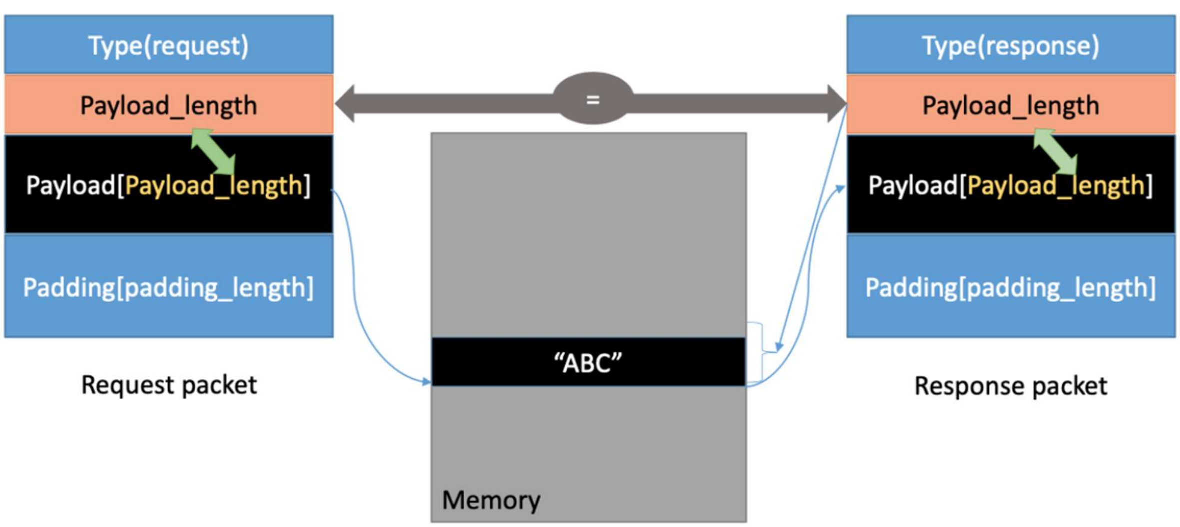

Figure 1 Heartbeat Extension feature

Each message can be modeled as a struct in C language containing the following fields:

- type: it specifies if it is a request or a response.
- payload_length: it is needed to understand what is the effective length of the payload sent by the client. Without this field, the protocol can't know how long is the payload, because there is no knowledge of how long is the padding.

  Two bytes (16 bits) are reserved for this field. It means that the payload can be at most 2^16 = 65536 bytes long.

- payload[payload_length]: they are arbitrary data. It can be anything the sender likes to send, as well as some unique value. The main feature of this protocol is that when the server receives the request and it responds, it sends back the same payload data.
- padding: it is random content of at least 16 bytes (128 bits) for bulking out the packet. This part of the packet is ignored by the other side.

Thus, the job of the receiver is to store the payload, create the response packet and send back the payload to the sender, by copying it from the server's memory.

Moreover, the RFC highlights that if the field indicating the length is too large, the packet should be ignored quietly.

The weakness of the Heartbeat implementation in the OpenSSL library resides just in this detail. In the next paragraph, we will detail where the vulnerability stays in the code and how it can be exploited for acquiring potentially very dangerous information present in memory.

Technical details about the vulnerability

The attack scenario requires that an on-purpose crafted packet will be sent to the server, in order to receive as a response a dump of the memory containing potentially sensitive information such as session keys, cookies in clear, private keys, etc.…

More in the detail, we can see the snippet of the vulnerable code in the C language that was running on the server.

```c
unsigned int payload;
unsigned int padding = 16; /* Use minimum padding */

// Read from type field first
hbtype = *p++; /* After this instruction, the pointer
                * p will point to the payload_length field */

// Read from the payload_length field from the request packet
n2s(p, payload); /* Function n2s(p, payload) reads 16 bits
                  * from pointer p and store the value
                  * in the INT variable "payload". */

pl = p; // pl points to the beginning of the payload content

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

    /* 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);

    // copy payload
    memcpy(bp, pl, payload);   /* pl is the pointer which
                                * points to the beginning
                                * of the payload content */
    bp += payload;

    // Random padding
    RAND_pseudo_bytes(bp, padding);
```

Figure 2 Server's code

What the server does is the following:

- The buffer for the response is allocated for containing all the content (there is already a mistake, because there is no check about the space allocated)

- the memcpy function copies the payload content from the source pl pointer to the destination bp pointer, for a length equal to the payload_length field.

Now a practical use case is examined.

Practical exploitation

In this section, the exploitation of the vulnerability will be shown. We recreated an environment using two machines that are a Kali Linux instance and a vulnerable Ubuntu VM obtained from the SEED Project [9] which is a project led by Wenliang Du from Syracuse University. To check that the machine was vulnerable, the Nmap Scripting Engine, running the script ssl-heartbleed was deployed. Thanks to this script, it is possible to check if a machine running an Apache Server is vulnerable to such an exploit.

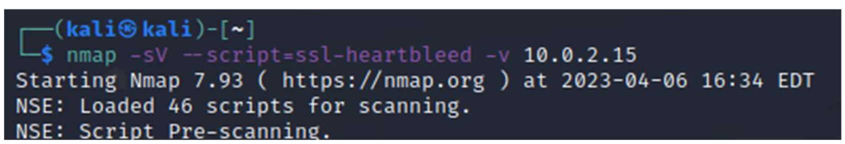

Figure 3 Nmap Scripting Engine

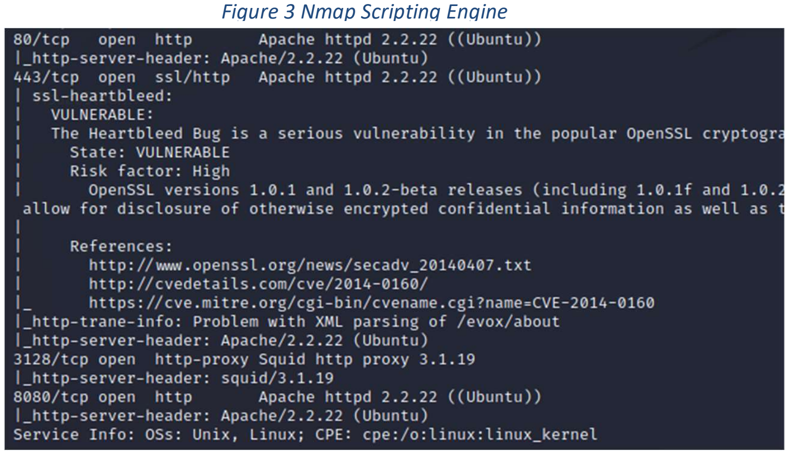

Figure 4 Nmap Scripting Engine Finding

The fastest way to exploit such a vulnerability is by relying on the Msfconsole tool provided within the Kali Distro. This tool provides different payloads to be deployed in exploiting vulnerabilities, paired with the chance to configure it in the best way according to the context in which the exploitation should be conducted. After selecting the correct payload and the proper settings (e.g. port that is used by the Apache server, victim's IP address), it is possible to perform two actions:

- Obtain the server's memory dump. This has been performed by sending requests with the highest size possible for the payload so that the highest memory dump obtainable is retrieved.
- Research the server's private key. This is performed through continuous memory dump until the server's memory section in which the key is stored is retrieved.

The first two memory dumps were obtained.

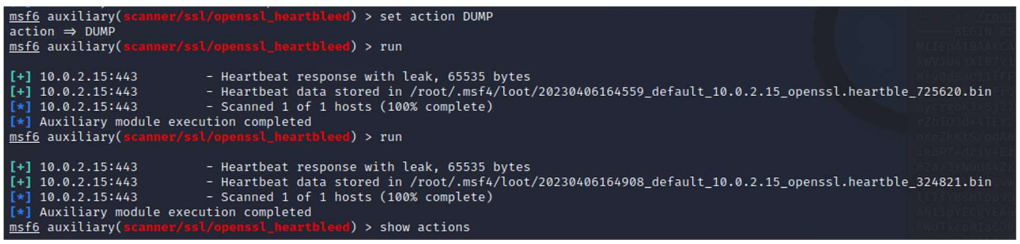

Figure 5 Configuring Payload

As shown, the dumps are stored within binaries, that then were examined as follows:

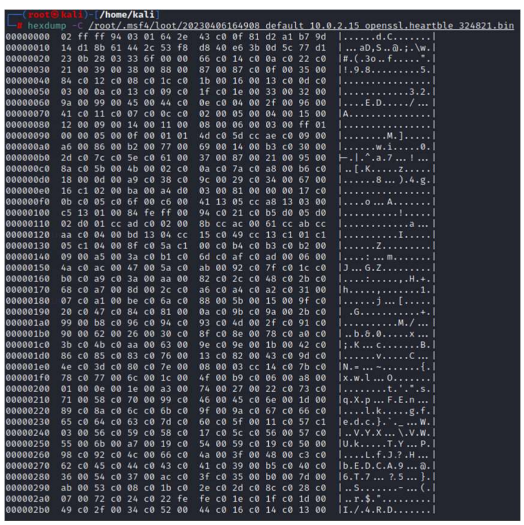

Figure 6 Dump of the Retrieved Content

Then It is possible to drill down, manually or by means of commands such as Grep, through the output of the Hexdump command. For instance below, is shown a .php file used as a handler within the Apache server. This could be useful in performing a PHP injection attack.

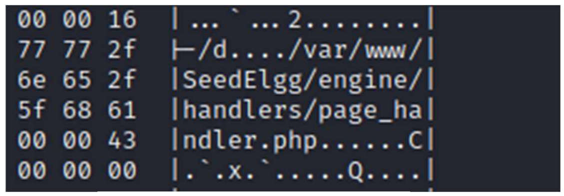

Figure 7 Examining the file dump

Moreover, a registration on the services hosted on the server was performed, and after this registration and performing again the exploit, information related to such request was found in the memory dumps, as shown in the screenshot below:

Figure 8 Examining the file dump

In particular, critical is the field Cookie, in which the Cookie value for this session is stored.

Performing the second action specified, the RSA private key used by the server within the SSL/TLS handshake was retrieved. It is clear that this attack is powerful and dangerous since highly sensitive information can be stolen and deployed in different ways. Following it is shown the output of the Msfconsole tool, in which the stolen RSA private key of the server is displayed:

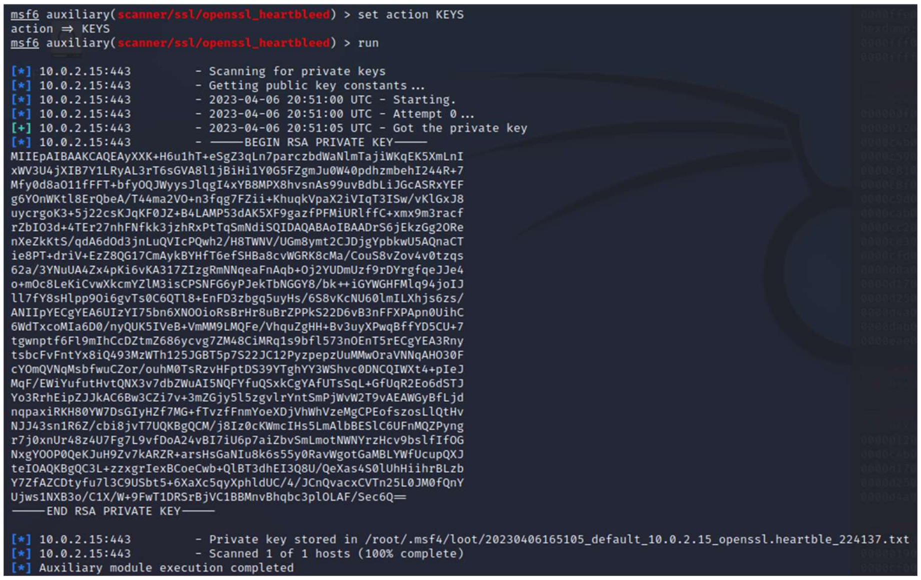

Figure 9 Retrieving Server's RSA Private Key

Conclusions

Different mitigations can be put in place. First of all, it is fundamental to ensure that the deployed systems are employing the latest OpenSSL version available, checking for updates. From a programming point of view, bounds checking before the memcpy function should be executed. Moreover, a check between the payload length passed in the field by the client and the actual length of the payload sent must be performed.

In conclusion, it has been estimated that about 17% of the SSL web servers exposed to the Internet[11] were affected. The consequences of this bug could be terrifying, as shown within the paper, since the exposure of sensitive information like passwords, session keys, pre-master secrets, and private keys would lead to the compromise of whole infrastructures, due to the large deployment of this protocol in many IT infrastructure among different sectors.

References

[1] Dierks, T., & Allen, C. (1999) *The TLS Protocol Version 1.0.* **https://www.rfc-editor.org/rfc/rfc2246**

[2] Rescorla, E. (2018). *The Transport Layer Security (TLS) Protocol Version 1.3.* **https://www.rfc-editor.org/rfc/rfc8446**

[3] Dierks, T., & Rescorla, E. (2008). *The Transport Layer Security (TLS) Protocol Version 1.2 Section 2.* **https://www.rfc-editor.org/rfc/rfc5246#section-2**

[4] OpenSSL Foundation, Inc. (2019). *Openssl.org.* **https://www.openssl.org/**

[5] *OpenSSL FIPS 140-2 Validation Certificate Issued - OpenSSL Blog.* (n.d.). www.openssl.org **https://www.openssl.org/blog/blog/2022/08/24/FIPS-validation-certificate-issued/**

[6] (2014). Openssl.org **https://www.openssl.org/news/secadv/20140407.txt**

[7] Seggelmann, R., Tuexen, M., & Williams, M. (2012). Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension. *www.rfc-Editor.org* **https://www.rfc-editor.org/rfc/rfc6520#section-1**

[8] *NVD - CVE-2014-0160.* (2014, April 7). Nist.gov. **https://nvd.nist.gov/vuln/detail/CVE-2014-0160**

[9] *SEED Project.* (n.d.). Seedsecuritylabs.org **https://seedsecuritylabs.org/index.html**

[10] Alston, A. (2022, September 17). *Heartbleed.* Github. **https://github.com/adamalston/Heartbleed**

[11] *Netcraft. Half a million widely trusted websites vulnerable to Heartbleed bug.* (2014). Netcraft.com **https://news.netcraft.com/archives/2014/04/08/half-a-million-widely-trusted-websites-vulnerable-to-heartbleed-bug.html**