Universidad Nacional Autónoma de México

Facultad de Ingeniería

Trabajo escrito: Heartbleed

Carrillo Sanchez Ricardo

De la Cruz Diaz Lucero Michelle

Gunnar Wolf

Sistemas Operativos

Grupo Teoría:

Semestre: 2022-1

14 de noviembre del 2021

Índice:

Heartbleed

¿Cómo comenzó Heartbleed?

Robin Seggelmann lanzó una actualización del protocolo SSL en el año nuevo del 2011 , en dicha actualización este protocolo agregaba una función llamada Heartbeat. Dicha función permite a los servidores recibir una señal por parte de las computadoras que hacen una solicitud por medio de los navegadores con el fin de saber si “no se ha quedado dormida” ya que el protocolo hace una encriptación de algunos datos confidenciales como contraseñas, es por ello que cuando no hay transferencia de datos confidenciales cabe la posibilidad de que se “quede dormida”.

No fue hasta el año del 2014 cuando un equipo de seguridad de Google descubrió el “bug” dentro del protocolo, este grupo informó a Open SSL de la vulnerabilidad pero ya para entonces una buena cantidad de los servidores a nivel mundial había sido afectado.

Conceptos generales

“Heartbleed Bug" es una vulnerabilidad grave en la popular biblioteca de software criptográfico OpenSSL. Esta debilidad permite robar la información protegida, en condiciones normales, por el cifrado SSL / TLS utilizado para asegurar Internet. SSL / TLS proporciona seguridad y privacidad en las comunicaciones a través de Internet para aplicaciones como web, correo electrónico, mensajería instantánea (IM) y algunas redes privadas virtuales (VPN).

El error Heartbleed permite que cualquier persona en Internet lea la memoria de los sistemas protegidos por las versiones vulnerables del software OpenSSL. Esto compromete las claves secretas utilizadas para identificar a los proveedores de servicios y para cifrar el tráfico, los nombres y contraseñas de los usuarios y el contenido real. Esto permite a los atacantes espiar las comunicaciones, robar datos directamente de los servicios y usuarios y hacerse pasar por servicios y usuarios.”

¿Qué hay de SSL?

Otra pieza fundamental sobre este ataque es el software OpenSSL. Este conjunto de librerías orientadas a la criptografía, estas ofrecen una aplicación open-source del protocolo TLS. Aún podemos especificar un poco la definición de Open SSL, ya que de por medio tenemos al protocolo SSL (Secure Sockets Layer). SSL se encargan de establecer enlaces autentificados y seguros entre redes de computadoras. Fue desarrollado en el año de 1995 por el equipo de Netscape con el fin de asegurar privacidad, autenticación e integridad de datos. Es de mencionar que el protocolo SSL ya no es el utilizado en la actualidad, sino que el protocolo TLS es la opción que se implementa actualmente.

La forma en la que podemos identificar que un sitio web implementa el protocolo SSL/TLS es a través del prefijo HTTPS en lugar del HTTP.

Open SSL por otro lado implementa tanto a SSL como a TLS, además de que está escrito en C y su diseño hace que varias distribuciones de Linux, Windows y MacOS sean compatibles. Eso trae consigo que alrededor del 66% de los servidores web usan Open SSL.

¿Y TLS?

Al igual que SSL, TLS se encarga de la encriptación de la comunicación entre aplicaciones web y servidores, ejemplo un navegador web cargando un sitio al cual queremos ingresar, aunque también podemos ver a este protocolo implementado en la encriptación de email, mensajería y otros servicios de comunicación como voz ip.

Para ver el funcionamiento del protocolo TLS, se necesita un certificado el cual debe ser emitido por una autoridad hacia la persona que es dueña del dominio. Cuando un cliente desea ingresar al sitio web que implementa TLS comienza un proceso que se le conoce como “handshake” entre el cliente y el servidor. Durante este proceso el dispositivo del cliente y el servidor:

1) Especifican la versión de TLS que se va a usar.
2) Se decide la suite de cifrado que se va a utilizar.
3) Autentifica la identidad del servidor utilizando el certificado TLS.
4) Genera llaves de sesión para encriptar los mensajes entre ambos una vez que el apretón de manos haya concluido.

¿Cómo es que funciona el ataque heartbleed?

Como lo mencionamos anteriormente, Seggelmann había implementado una función llamada heartbeat, la cual es una función para mantener en vivo en la que el dispositivo del cliente envía una carga de datos útil arbitrarios a otro extremo (servidor), y este envía una copia exacta de esos datos para demostrar que todo marcha bien. Acorde al estándar oficial, este mensaje se vería definido de la siguiente forma.

```
struct {
  HeartbeatMessageType type;
  uint16 payload_length;
  payload [HeartbeatMessage.payload_length];
  padding [padding_length];
} HeartbeatMessage;
```

El HeartbeatMessage llega a una estructura **SSL3_RECORD**, el cual es un bloque básico para las comunicaciones SSL/TLS. Algunas de las llaves relevantes que se utilizan para explicar el ataque son:

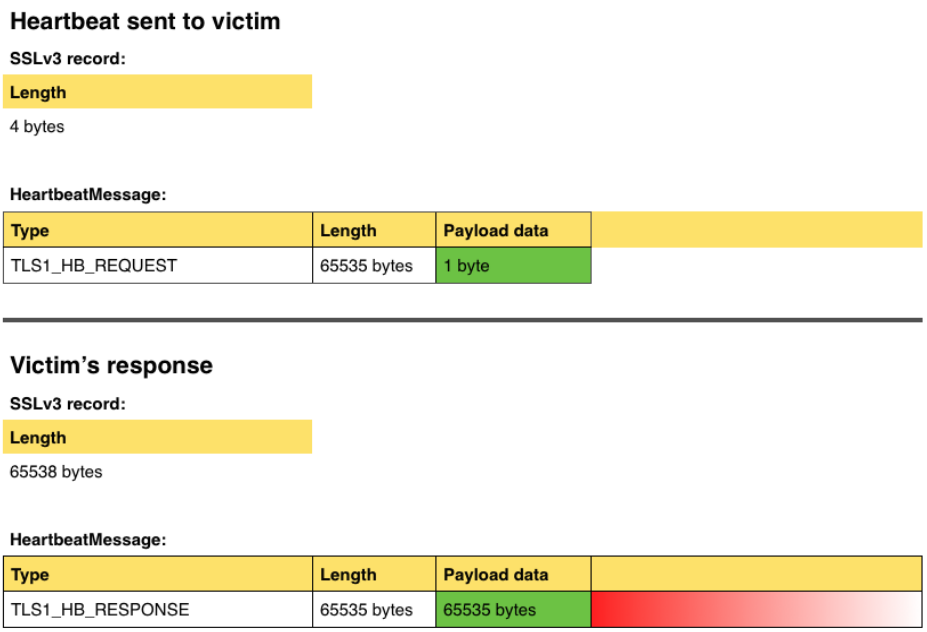
- ***length:*** representa la cantidad de bytes disponibles.
- ***data:*** que es un apuntador al mensaje *HeartbeatMessage* (donde comienza en memoria).
- ***Payload_length:*** es el número de bytes en la carga arbitraria (*HeartbeatMessage* "de regreso") que tiene que ser enviada de vuelta.

A continuación podemos ver la estructura definida dentro del código de Open SSL.

```
typedef struct ssl3_record_st

349     {
350 /*r */  int type;               /* type of record */
351 /*rw*/  unsigned int length;    /* How many bytes available */
352 /*r */  unsigned int off;       /* read/write offset into 'buf' */
353 /*rw*/  unsigned char *data;    /* pointer to the record data */
354 /*rw*/  unsigned char *input;   /* where the decode bytes are */
355  /*r */   unsigned char *comp;      /* only used with decompression -
     malloc()ed */
356 /*r */  unsigned long epoch;    /* epoch number, needed by DTLS1 */
357 /*r */  unsigned char seq_num[8]; /* sequence number, needed by DTLS1 */
358     } SSL3_RECORD;
```

Cualquiera que envíe un *HeartbeatMessage* controla la longitud de la carga, sin embargo, este bug sucede por que esta "carga" no es verificada respecto a la de su padre de (SSL3_RECORD), ocasionando que el atacante invada regiones de la memoria que no debería conocer.



Ejemplo del funcionamiento del ataque por: Jeff Howell

El ejemplo anterior, un atacante envía un mensaje (*HeartbeatMessage*) de 4 bytes y una carga útil de 1 byte que el servidor reconoce como la longitud a regresar. El atacante por

otro lado, modifica el payload_length y afirma que esta tiene un tamaño de 64KB cuando debería ser de 4 bytes. Nuestro servidor víctima ignora la estructura SSL3_RECORD y lee los 64KB de su propia memoria empezando de donde comienza el mensaje que se recibió y copia en un buffer de tamaño adecuado para enviarlo de vuelta al atacante. En consecuencia, acumulando demasiados bytes y filtrando información potencialmente peligrosa.

El código de Open SSL que procesaba el mensaje que se recibía de Heartbleed message se vería de la siguiente forma:

```
/ * Leer primero el tipo y la longitud de la carga útil * /
hbtype = * p ++;
n2s (p, carga útil);
pl = p;
```

En las hbtype se inserta el tipo de mensaje que se va a recibir, y el puntero aumenta en un byte. Dentro de ns2( ) se escribe la longitud de la carga del mensaje que se recibió a la carga del mensaje de regreso. Por último, p1 se convierte en un apuntador de la carga que se recibió (mensaje).

El anterior código únicamente hace válida la vulnerabilidad del diseño del protocolo, la longitud de la carga que habíamos definido en el ejemplo (64 KB) debe ser regresada de vuelta, pero no únicamente se regresan 64KB sino que también se regresan:

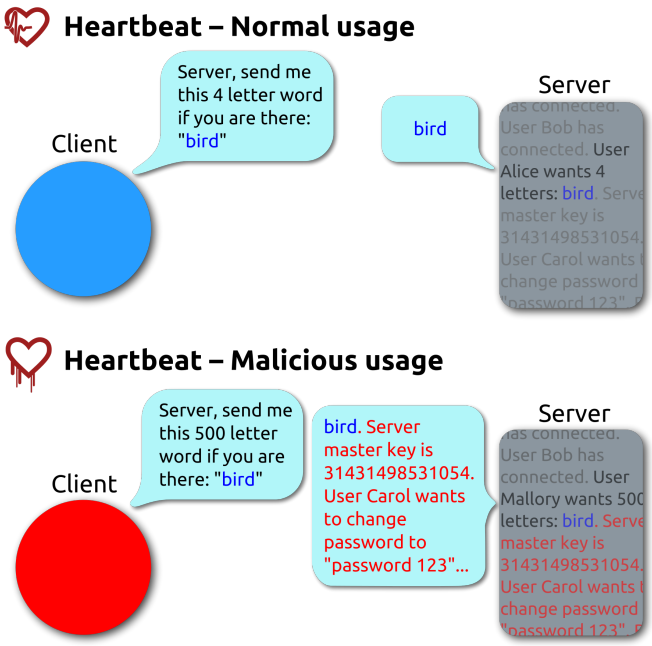
1) 1 byte extra para almacenar el tipo de mensaje
2) 2 bytes extra para almacenar la longitud de la carga y el padding.

Y para construir el mensaje que se regresara de vuelta, el código de OpenSSL haría algo como lo siguiente:

```
/ * Ingrese el tipo de respuesta, la longitud y la carga útil de la copia
* /
* bp ++ = TLS1_HB_RESPONSE;
s2n (carga útil, bp);
memcpy (bp, pl, payload);
```

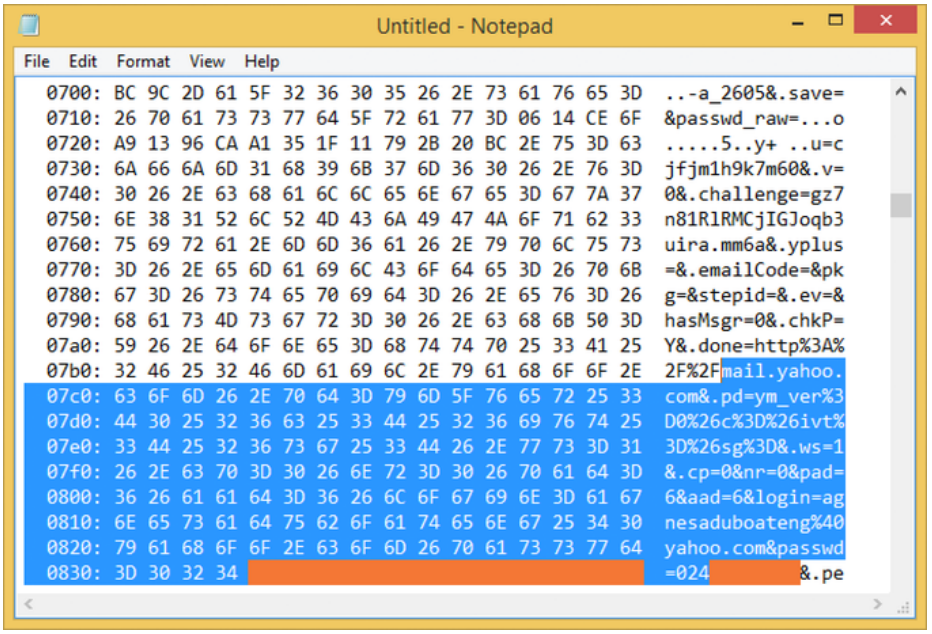
Aquí el código escribe el tipo de respuesta al inicio del buffer, por lo que debería incrementar el puntero, se usa s2n para escribir la longitud de la carga en la memoria y se vuelve a incrementar el puntero y por último se copia el mensaje o carga en una variable de respuesta.

Teniendo en cuenta lo anterior los tipos de respuesta que recibimos por parte del servidor se verían algo como el siguiente ejemplo:



Ejemplo del funcionamiento esperado de Heartbleed vs lo que pasó en realidad por: Malwarebytes

O de forma más concisa con un script para realizar el ataque:



Tweet de Mark Loman mostrando la respuesta de servidores de Yahoo al ataque Heartbleed

La solución:

Según (Chandra, 2014) para solucionar el bug se deben hacer dos tareas, que podemos ver en dos scripts:

a) Primeramente se debe saber si la longitud de la carga útil es cero o no. De ser así se debe descartar el mensaje.

```
/ * Leer primero el tipo y la longitud de la carga útil * /

If (1 + 2 + 16> s-> s3-> rrec.length)
       return 0; / * descartar
       hbtype = * p ++;
       n2s (p, carga útil);
```

b) Asegurar que el valor del campo de la longitud de la carga útil del mensaje de la carga coincide con el valor de la carga de la solicitud, de no ser así debemos descartar el mensaje.

```
If (1 + 2 + payload + 16> s-> s3-> rrec.length)
       devuelve 0; / * descartar por RFC 6520 seg. 4 * /
       pl = p;
```

¿Qué hacer para evitar un nuevo Heartbleed?

“Los proyectos deben asegurarse de que sean más fáciles de analizar. Por ejemplo, deben simplificar su código, simplificar su interfaz de programa de aplicación (API) y asignar y desasignar memoria normalmente.”

(Wheeler, 2017)

Otras formas en las que podemos reducir el impacto de las vulnerabilidades siempre tenemos que tomar en cuenta:

- Establecer defensas al momento de asignar memoria, especialmente hablando de Linux (malloc).
- Sobreescribir información crítica estableciendo mecanismos de encriptado en datos sensibles.
- Separación de privilegios para los secretos criptográficos
- Corregir la infraestructura de certificados SSL/TLS.
- Actualizaciones de Software

Bibliografía:

- Hern, A. (2017, 9 febrero). *Heartbleed: developer who introduced the error regrets «oversight»*. The Guardian. Recuperado 9 de noviembre de 2021, de https://www.theguardian.com/technology/2014/apr/11/heartbleed-developer-error-regrets-oversight

- Heartbleed. (2021, 19 octubre). En *Wikipedia*. https://en.wikipedia.org/wiki/Heartbleed#Discovery

- Contributor, T. (2014, 16 abril). *OpenSSL*. WhatIs.Com. Recuperado 11 de noviembre de 2021, de https://whatis.techtarget.com/definition/OpenSSL

- S. (2020, 5 noviembre). *What is OpenSSL and how it works?* SSL Dragon. Recuperado 11 de noviembre de 2021, de https://www.ssldragon.com/blog/what-is-openssl-and-how-it-works/

- Seggelmann, R. (2012, 21 febrero). *rfc6520*. Datatracker.Ietf.Org. Recuperado 12 de noviembre de 2021, de https://datatracker.ietf.org/doc/html/rfc6520

- Williams, C. (2014, 9 abril). *Anatomy of OpenSSL's Heartbleed: Just four bytes trigger horror bug*. The Register. Recuperado 13 de noviembre de 2021, de https://www.theregister.com/2014/04/09/heartbleed_explained/

- Leyden, J. (2014, 11 abril). *Revoke, reissue, invalidate: Stat! Security bods scramble to plug up Heartbleed*. The Register. Recuperado 13 de noviembre de 2021, de https://www.theregister.com/2014/04/09/heartbleed_vuln_analysis/

- Cloudflare. (s. f.). *Attention Required!* | *Cloudflare*. Recuperado 12 de noviembre de 2021, de https://www.cloudflare.com/learning/ssl/transport-layer-security-tls/

- Howell, J. (s. f.). *Heartbleed Details*. JEFF HOWELL. Recuperado 13 de noviembre de 2021, de https://www.jeffreyahowell.com/heartbleed-details.html

- Wheeler, D., January 2017, How to Prevent the next Heartbleed

- Chandra, B., (2014, 13 Mayo), A technical view of the OpenSSL ‘Heartbleed’ vulnerability, A look at the memory leak in the OpenSSL Heartbeat implementation. Versión 1.2.1. Recuperado 13 de noviembre de 2021

- https://github.com/ctfs/write-ups-2014/blob/master/plaid-ctf-2014/heartbleed/heartbleed.py