Covert Channel sobre SSL Heartbeat

Luis A. Catanzariti[†1] y Nestor Masnatta[†2]

[†]*Maestría en Seguridad Informática - Universidad de Buenos Aires*
Buenos Aires, Argentina
[1]lcatanzariti@gmail.com
[2]nmasna@dc.uba.ar

***Abstract*—This work shows a way to establish a covert channel over TLS, specifically using the sub protocol SSL Heartbeat. This sub protocol was wrongly implemented by openssl, and launched a worldwide alert when hundred of thousands of web sites, supposed to be secure, could expose their digital certificate's private keys.**

A covert channel is a communication channel that allows a process to transfer information in a manner that violates the system's security policy. In our work a covert channel was achieved by modifying the communicating agents in both sides: client and server. The modified client sends a special built HeartbeatRequest message containing the activation sequence. For a standard server, the protocol continues in the conventional way. The modified server instead, after receiving the activation sequence, sends the hidden information using a HeartbeatResponse message. When the server receives a standard HeartbeatRequest, it responds normally in order to hide its behavior and to avoid being detected.

The design and implementation of this covert channel allow sending data to a client from a server that, from the point of view of the TLS protocol, acts as usual managing its websites. These data are masked, so they become invisible to external agents.

***Resumen*— Se presenta una alternativa para establecer un canal encubierto de comunicación (covert channel) sobre el protocolo TLS, particularmente sobre el sub protocolo SSL Heartbeat. Este sub protocolo fue el que, mal implementado por el programa openssl, permitió que recientemente se provocara una alerta mundial ante la posibilidad de que cientos de miles de sitios web supuestamente seguros pudieran exponer las claves privadas de sus certificados digitales.**

Un covert channel es un canal de comunicación que permite a un proceso transferir información violando alguna política de seguridad. En nuestro trabajo, este se logra a través de la modificación de los agentes que actúan tanto del lado servidor como del lado cliente. El cliente modificado envía un paquete HeartbeatRequest especialmente construido con la correspondiente secuencia de activación. Para un servidor que sigue el protocolo en forma correcta, el mismo continúa de la manera habitual. El servidor modificado, ante la secuencia de activación, envía la información subrepticia a través del paquete HeartbeatResponse. Ante un paquete estándar HeartbeatRequest, la entidad comunicante responderá de manera tradicional para evitar o dificultar su detección.

El diseño e implementación del canal encubierto presentados permiten enviar datos hacia un cliente desde un servidor que, desde el punto de vista del protocolo TLS, actúa con normalidad manteniendo su correspondiente sitio web. Estos datos resultan indetectables por agentes externos.

I. Introducción

En este trabajo exponemos la posibilidad de transmitir información de manera encubierta utilizando como medio de transmisión la extensión Heartbeat del protocolo SSL/TLS.

SSL/TLS es el estándar que se utiliza en miles de millones de comunicaciones alrededor del mundo para la gestión de la seguridad de la información. Un ejemplo particularmente extendido lo conforman los millones de sitios web de e-commerce o entidades financieras que necesitan autenticación o confidencialidad. En este trabajo exponemos una forma de transmitir información de manera encubierta desde un servidor especialmente preparado que atiende las peticiones de conexión SSL/TLS regularmente, pero que ante la solicitud de un cliente que posee la clave de activación, inicializa un canal encubierto para la transferencia de información. Este canal de comunicación, resulta así indetectable, ya que la comunicación está solapada por una conexión SSL/TLS normal. No solamente el contenido de la comunicación podría ser cifrado y por lo tanto oculto, sino que lo indetectable aquí es el hecho mismo de la existencia de un canal de comunicación independiente del canal habitual SSL/TLS.

Para la creación de este canal sólo se requiere contar con privilegios de acceso para la modificación de una biblioteca de software en el servidor o para la instalación inadvertida del software ya adulterado.

A. El protocolo SSL/TLS

El protocolo SSL se ubica sobre la capa de transporte de la suite de protocolos TCP/IP y debajo de la capa de aplicación, proporcionándole servicios de transmisión segura a esta última. En este trabajo utilizamos los términos SSL y TLS en forma indistinta, realizando las aclaraciones del caso cuando sea necesario, aunque se considera hoy que SSL es inseguro debiendo ser utilizado solamente TLS. Particularmente SSL/TLS trabaja sobre TCP y DTLS sobre UDP.

SSL/TLS no está conformado por un único protocolo, sino que está definido como una suite de protocolos, que se compone de dos capas: en la capa más baja (que interactuará con el protocolo TCP) se encuentra el protocolo SSL (o TLS) Record Protocol y en la capa superior encontraremos un conjunto de protocolos de acuerdo mutuo (SSL/TLS Handshaking Protocols) compuesto por los protocolos: SSL (o TLS) Handshake Protocol, SSL (o TLS) Change Cipher Spec Protocol y SSL (o TLS) Alert Protocol. Una vez establecida la conexión entre ambos extremos, el Record Protocol será el encargado de brindar confidencialidad e integridad a las capas superiores (ver [1] y [2]).

B. El protocolo Heartbeat

A partir de febrero de 2012 se agregó una extensión implementando el Heartbeat Protocol, que trabaja sobre la capa SSL Record. Su propósito es evitar la renegociación de las conexiones para verificar si uno de los pares sigue conectado (keep-alive). En el caso de DTLS (Datagram Transport Layer Security) -un servicio que no está orientado a la conexión, ya que se implementa sobre UDP (User Datagram Protocol)- esta extensión sirve, además, para descubrir la unidad máxima de transferencia de la ruta en cuestión (PMTU) [RFC6520] [3]. En los siguientes apartados explicamos la extensión Heartbeat con mayor profundidad, así como también los detalles de implementación del covert channel.

Este nuevo protocolo utiliza dos tipos de mensajes idénticos: HeartbeatRequest y HeartbeatResponse. El mesaje HeartbeatRequest puede enviarse prácticamente en cualquier momento de la conexión (excepto durante el handshake) y debería contestarse con el correspondiente HeartbeatResponse. Si se recibe el Request durante el handshake simplemente debe descartarse. Mientras está en curso un request (se considera que un request permanece en curso hasta que llega el correspondiente response) no deben enviarse otros requests [RFC6520] [3].

C. Covert-Channel

El concepto de covert channel fue originalmente publicado por Lampson en 1973, quien planteaba la posibilidad de fugas de información en sistemas aparentemente seguros, debido a que los procesos podrían estar generando y enviando información de alguna forma no sospechada, utilizando canales no definidos específicamente para esa acción (P.Ej. manipulando la tasa fallos de página (*Page Fault Rate*), alterando el tiempo de utilización del procesador (*CPU Usage*), modificando el *File System*, etc). Entonces el autor planteaba como solución el aislamiento de los procesos y que estos quedaran confinados a entornos sin riesgo de fuga (security leak) [4].

En nuestro trabajo planteamos la posibilidad de enviar información utilizando como canal encubierto el campo payload de la extensión Heartbeat (Hartbeat Protocol) de SSL. Dicho campo solo debería transportar números de secuencia y bytes aleatorios. Otra aproximación a esta idea de canal encubierto puede encontrarse en [5].

II. Covert Channel sobre Heartbeat

El protocolo Heartbeat cuenta con un campo de 16 bits para indicar el tamaño del payload, dando un máximo teórico de 65536 bytes a enviar. Pero por restricciones en la especificación de SSL, el paquete Heartbeat, en su totalidad, no debe exceder los 2^{14} bytes de longitud. Debido a que la cabecera cuenta con 3 bytes (1 byte de tipo y 2 bytes para indicar la longitud) y que se deberá enviar obligatoriamente un padding de 16 bytes, obtenemos $2^{14} - 3 - 16 = 16365$ bytes como valor límite para el envío de información útil (payload).

Este trabajo se basó en una arquitectura x86 - IA32, utilizando como software de base GNU/Linux Debian 7.5 Wheezy, con kernel linux 3.2.0-4-686-pae. Sobre este sistema operativo, se ha instalado el package libssl1.0.0 (1.0.1e-2+deb7u9), el mismo contiene las librerías compartidas (shared libraries) libssl y libcrypto, las cuales son utilizadas por diversas aplicaciones (P. Ej: el servidor web apache2, el servicio de shell seguro openssh, telnet seguro telnet-ssl, entre otros) para la transmisión segura de información.

Bajo esta arquitectura, se procedió a revisar el código fuente de la librería, y se detectó que las funciones que están vinculadas al proceso Heartbeat son 4:

- Dos de ellas están relacionadas con TLS y se encuentran en el archivo ssl/t1_lib.c
  int tls1_heartbeat(SSL *)
  int tls1_process_heartbeat(SSL *)
- Las dos restantes están relacionadas con DTLS y se encuentran en el archivo ssl/d1_both.c:
  int dtls1_heartbeat(SSL *)
  int dtls1_process_heartbeat(SSL *)

En el código fuente se podrá apreciar que las funciones realizan tareas similares para ambos protocolos [6]. A fin de realizar las pruebas de concepto pertinentes, nos abocaremos a las funciones contenidas en el archivo t1_lib.c, para TLS. Pero las modificaciones serían idénticas si quisiéramos implementarlo también en d1_both.c para DTLS.

Para lograr enviar información encubierta utilizando el protocolo Heartbeat en TLS, deberemos introducir dicha información en el punto de ejecución donde se genera el paquete de HeartbeatRrequest (type = 0x01) o de HeartbeatResponse (type = 0x02), nosotros elegimos la segunda opción, es decir, que la información se enviará como respuesta a una solicitud Heartbeat. De esta forma, se podrá obtener información a demanda por un agente externo. Dicho punto de inserción se encuentra en la función: **tls1_process_heartbeat**.

Existen diversas formas posibles para hacer que la información a transmitir se encuentre disponible para las funciones del código involucradas con el proceso Heartbeat. Algunas de ellas las listamos a continuación:

1) **Áreas de memoria compartida (shared memory):** la librería define un área de memoria compartida para que otros procesos puedan escribir la información a enviar. La dificultad de esta aproximación, radica en las cuestiones de sincronización (IPC: Inter-Process Communication) de procesos con buffer finito.
2) **Utilizando *sockets*:** la librería se comunica con otros procesos a través de sockets para obtener la información. La dificultad aparece en la gestión de los sockets: jacketing para manejar el modo de recepción bloqueante (blocking receive). Además, podría detectarse el socket abierto.
3) **Utilizando *unnamed pipes*:** si bien es similar al caso de los sockets, exige que los procesos estén relaciona-

dos o dicho de otra forma, que tengan conocimiento uno del otro, a través de la jerarquía de procesos: solo los procesos hijos de aquel proceso que crea la tubería, utilizando la llamada al sistema *pipe*, podrán utilizarla [7]. En este caso, la gestión se torna complicada.

4) **Utilizando archivos regulares:** es una forma bastante cómoda de proporcionar la información a transmitir. Pero tiene la dificultad de que una vez leída, la información debe ser eliminada ya que si esto no se hiciese, se presentarían dos problemas: 1ro la biblioteca siempre enviaría los mismos datos, 2do el archivo crecería y podría hacerse notorio. Finalmente quedaría registrada en el sistema la información enviada.
5) **Utilizando named pipes:** se describe a continuación.

De todas las formas mencionadas elegimos la última, por ser la más simple (de hecho hemos conseguido hacer las pruebas de concepto agregando menos de 50 líneas de código C, ver detalle en el listado 1). El método consiste en utilizar un pipe con nombre (siendo su nombre correcto FIFO), el cual será leído en forma no bloqueante por la librería (proceso lector o más específicamente consumidor), para poder enviar la información en forma encubierta. Si en el mismo existe información (previamente introducida), esta se insertará en el paquete HeartbeatResponse. En cambio, si el FIFO está vacío, se responderá al HeartbeatRequest con un paquete HeartbeatResponse estándar, según el funcionamiento establecido en RFC 6520 [3].

Para crear estos archivos especiales, el kernel de los sistemas operativos Unix y derivados (como Linux) nos proporciona la llamada al sistema (*system call* o *supervisor call*) **mknod**, la cual puede invocarse empleando el comando homónimo [8].

Básicamente un FIFO es un pseudo archivo que funciona de manera similar a un *unnamed pipe*, pero se accede a él con las mismas llamadas al sistema utilizadas para acceder a un archivo regular (*open*, *read*, *write*, *close*, etc). Lo interesante de este método, es que permite la comunicación de procesos que no se conocen entre ellos, es decir que no poseen el identificador de proceso (PID: Process ID) del otro. Los datos se transfieren con la planificación primero en entrar primero en salir (*first in first out*, por su sigla **FIFO**). Además puede ser abierto por múltiples procesos para lectura y escritura [9], sin necesidad de que los procesos intervinientes tengan conocimiento acerca del o de los procesos que se encuentran en el otro extremo [7].

Una vez creado el FIFO, se procede a cargarlo con información utilizando una aplicación a la cual denominaremos productor hasta que el otro extremo, el proceso consumidor (que será la librería *libssl* modificada) la pueda leer. Es importante notar que si el proceso productor finalizara su ejecución antes de que la información se pueda leer, la misma se perderá.

Dentro de las limitaciones que tienen los FIFOs podemos mencionar que, debido a la forma de implementación, en la mayoría de los sistemas operativos tipo Unix se utiliza la estructura de i-nodos del sistema de archivos para almacenar la información. En esa estructura, y por cuestiones de rendimiento, solo se utilizan los bloques directos para almacenar la información (generalmente 10, dependiendo de cada implementación particular de la estructura de i-nodos). Para mayor detalle de la estructura del sistema de archivos, basadas en i-nodos, utilizada por sistemas tipo Unix consultar [7, capítulos 4 y 5], [10, capítulo 8], [11, capítulos 8 y 9] y [12].

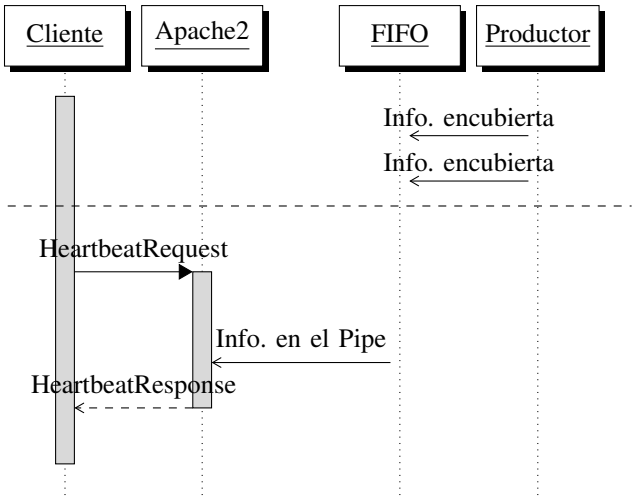

Fig. 1. Diagrama de secuencia

A. *Covert Channel de extremo a extremo*

En la figura 1 presentamos un diagrama de secuencia en el que se puede apreciar cuales son las interacciones entre el servidor, el pipe y el cliente. Siendo el disparador el envío del Heartbeat Request con la secuencia de activación (ver sección III) por parte del cliente.

B. *Código que se agregó a la función tls1_process_heartbeat para la demostración*

Listado 1. Código agregado a la biblioteca, en el archivo ssl/t1_lib.c

```
. . . CONTENIDO OMITIDO ...
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
. . . CONTENIDO OMITIDO ...
#define MAX_MSG_LEN 8192 // MAX: 16361
int leerFifo(char *, unsigned, char *);
. . . CONTENIDO OMITIDO ...
// COVERT CHANNEL
unsigned char *ptrAux = NULL;
if(payload >= 28 && memcmp(SHA1(pl,8,NULL),&pl
    [8],20) == 0)
  {
  int leidos, auxInt;
  char covertBuffer[MAX_MSG_LEN];

  leidos = leerFifo("/var/www/ssl", MAX_MSG_LEN,
      covertBuffer);
  if(leidos > 0)
    {
    if((leidos+sizeof(leidos)) > payload)
      {
      ptrAux = OPENSSL_malloc(leidos);
      pl = ptrAux;
```

```
      auxInt = leidos+sizeof(leidos);
      }
    else
      auxInt = payload;
    memcpy(pl,&leidos,sizeof(leidos));
    memcpy(&pl[sizeof(leidos)],covertBuffer,leidos
        );
    payload = auxInt;
    }
  }
 /* FIN COVERT CHANNEL */
. . . CONTENIDO OMITIDO ...
OPENSSL_free(ptrAux);
. . . CONTENIDO OMITIDO ...
int leerFifo(char *path, unsigned bytesALeer, char
     *pBuffer)
  {
  int fd;
  int bytesLeidos;
  if((fd = open(path, O_RDWR | O_NONBLOCK)) == -1)
    return -1;

  bytesLeidos = read(fd,pBuffer,bytesALeer);
  close(fd);
  return bytesLeidos;
  }
```

III. Covert-Channel en acción

El trabajo consistió en agregar una leve modificación a la librería **libssl1.0.0**, la cual implementa la funcionalidad del protocolo Heartbeat, y que esta envíe información a la entidad que realice el HeartbeatRequest.

Para que se envíe información solamente al cliente adecuado, se decidió aplicar una validación del payload en el mensaje HeartbeatRequest. Si el payload contiene una secuencia de bytes determinada, entonces se activará el código para el envío de información en forma encubierta.

Esta secuencia de activación consiste en 8 bytes arbitrarios (aleatorios), seguidos de su hashcode SHA1, de 20 bytes (para mayor detalle sobre funciones de hash se recomienda [13, capítulo 9]). Solamente si la petición contiene una secuencia válida, esto es: la concatenación de 8 bytes más los 20 bytes correspondientes al hash SHA1 de los primeros 8 bytes, entonces se procederá a leer el FIFO hasta en una cantidad máxima de bytes válida para el protocolo (siempre respetando los límites prescriptos en el RFC6520). Si el FIFO estuviera vacío entonces simplemente no se hará nada y el protocolo ejecutará en su forma habitual.

Si en el pipe existen datos, estos serán leídos (también quitándolos del mismo, dada la forma de funcionamiento de estos pseudo-archivos) y ensamblados en un paquete HeartbeatResponse, con una pequeña salvedad: los primeros 4 bytes contendrán la longitud del payload con la información encubierta (representada por un entero signado). Se debe tener en cuenta que en esta instancia no tendrá importancia la cantidad de información recibida en el HeartbeatRequest.

Una vez armado el paquete HeartbeatRresponse, se enviará el mismo al solicitante (que puede ser un cliente o un servidor). Y el solicitante en destino desensamblará el paquete y leerá la cantidad de información indicada.

En el caso de que se reciba un mensaje HeartbeatRequest que no contenga una secuencia de activación válida, la función ejecuta en su forma estándar habitual, es decir, contestará un mensaje HeartbeatResponse con el mismo payload que contiene el paquete HeartbeatRequest (de acuerdo al RFC6520 [3]).

El cálculo del hashcode SHA1 en la secuencia de activación, lo realizamos con la función homónima proporcionada por la misma biblioteca OpenSSL (**unsigned char * SHA1(const unsigned char *d, unsigned long n, unsigned char *md);**).

En la figura 2 se aprecia el resultado de las capturas de red, realizadas durante la prueba de concepto, mientras se efectúa el intercambio de mensajes Heartbeat. Se resalta en color rojo el mensaje HeartbeatRequest (paquete 8 de la secuencia) y en color azul el mensaje HeartbeatResponse (paquete 13 de la secuencia).

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.1.2	TCP	74	48375 > vop [SYN] Se
2	0.000025000	127.0.1.2	127.0.0.1	TCP	74	vop > 48375 [SYN, AC
3	0.000042000	127.0.0.1	127.0.1.2	TCP	66	48375 > vop [ACK] Se
4	0.000592000	127.0.0.1	127.0.1.2	TLSv1	291	Client Hello
5	0.000603000	127.0.1.2	127.0.0.1	TCP	66	vop > 48375 [ACK] Se
6	0.011874000	127.0.1.2	127.0.0.1	TLSv1	1159	Server Hello, Certi
7	0.011933000	127.0.0.1	127.0.1.2	TCP	66	48375 > vop [ACK] Se
8	0.014521000	127.0.0.1	127.0.1.2	TLSv1	4186	Heartbeat Request
9	0.015340000	127.0.1.2	127.0.0.1	TCP	1526	[TCP segment of a re
10	0.015410000	127.0.0.1	127.0.1.2	TCP	66	48375 > vop [ACK] Se
11	0.015432000	127.0.1.2	127.0.0.1	TCP	1526	[TCP segment of a re
12	0.051848000	127.0.0.1	127.0.1.2	TCP	66	48375 > vop [ACK] Se
13	0.051940000	127.0.1.2	127.0.0.1	TLSv1	1266	Heartbeat Response
14	0.051990000	127.0.0.1	127.0.1.2	TCP	66	48375 > vop [ACK] Se

Fig. 2. Capturas de red

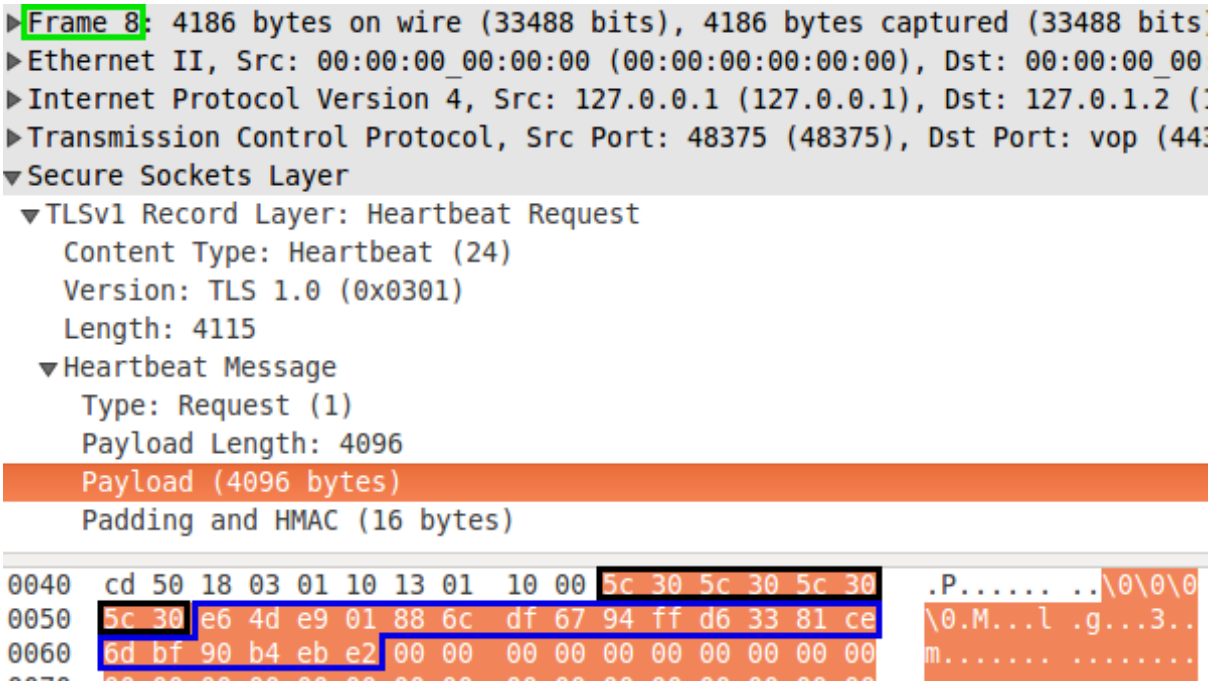

Fig. 3. HeartbeatRequest

En la figura 3 se puede apreciar el detalle del paquete HeartbeatRequest capturado. En verde se resalta el número de trama en la secuencia capturada (**8**). En color negro se resaltan los 8 bytes arbitrarios (en este caso: **5c:30:5c:30:5c:30:5c:30** hexadecimal) y en color azul se destacan los 20 bytes correspondientes al hashcode SHA1 los 8 bytes anteriores (en este caso **e6:4d:e9:01:88:6c:df:67:94:ff:d6:33:81:ce:6d:bf:90:b4:eb :e2** hexadecimal). Estos 28 bytes conforman la secuencia de activación para que el destinatario conteste con la información almacenada en el FIFO.

El detalle del paquete HartbeatResponse capturado se presenta en la figura 4. En la imagen se resalta en color verde el número de trama en la secuencia capturada (**13**) y en color azul la longitud de los datos extraídos del FIFO. Este dato se envía como un entero signado de 4 bytes (en este caso: **72:01:00:00** en hexadecimal y expresado en formato *Little Endian*. La descripción del ordenamiento de bytes se puede encontrar en [14, p.40]).

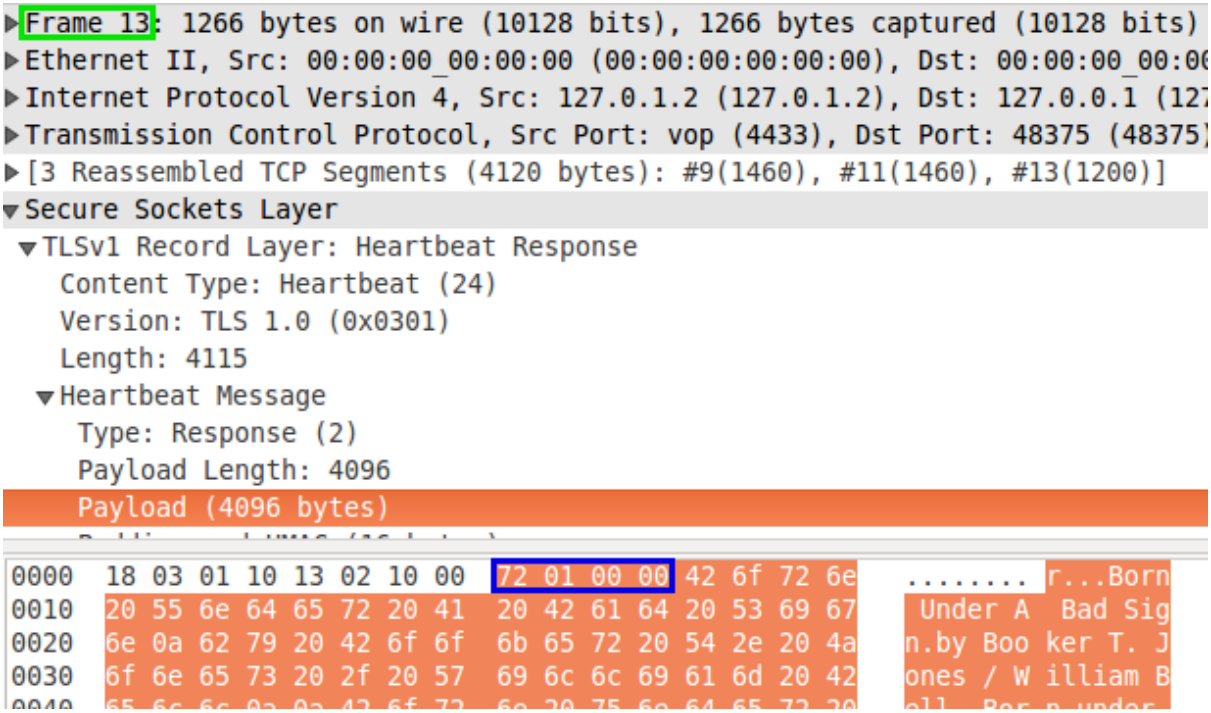

Fig. 4. HeartbeatResponse

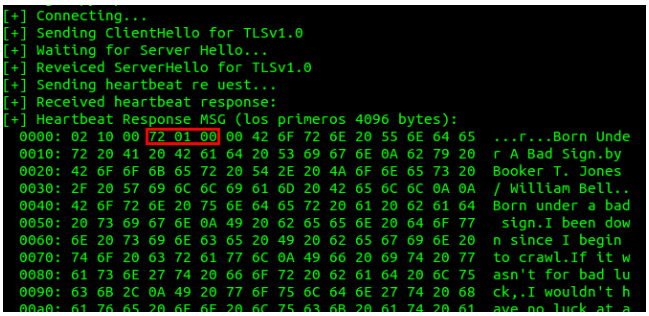

Fig. 5. Salida mostrada al cliente que envía la secuencia de activación correcta

En la figura 5 se puede observar el detalle de la respuesta mostrada por la aplicación python (hemos modificado el código de [15] para realizar la prueba de concepto) que tras enviar un mensaje HeartbeatRequest legal (todos sus campos son consistentes con la especificación), incluyendo la secuencia de activación de 28 bytes anteriormente mencionada, recibe el HeartbeatResponse transportando la información almacenada en el FIFO en ese momento. En rojo aparece resaltada la longitud de los datos (como ya se ha indicado **72:01:00:00** en hexadecimal) e inmediatamente a continuación los datos.

Para la prueba de concepto hemos cargado el FIFO a través de una sencilla aplicación escrita en C, utilizando la biblioteca GNU-ncurses, para simular el proceso productor mostrado el diagrama de secuencia anterior. En la figura 6 se muestra como se procede a cargar el FIFO.

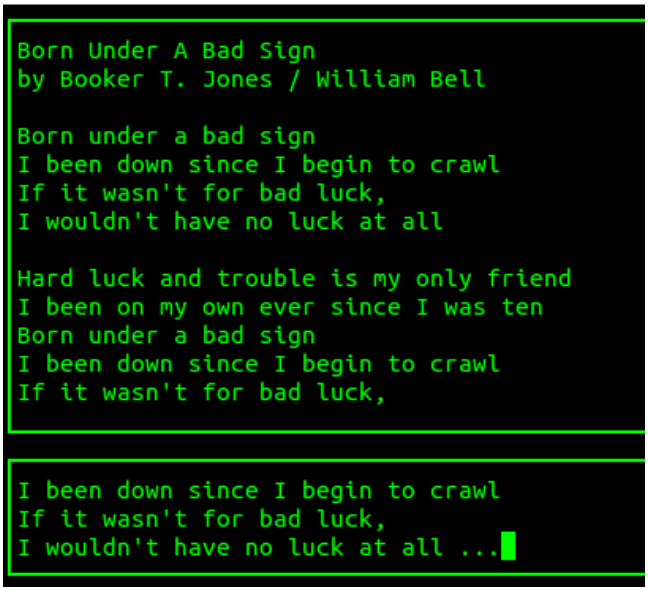

Fig. 6. Cargando el FIFO

IV. Conclusiones

Hemos demostrado que es posible construir un canal encubierto, enmascarado dentro de una comunicación que resulta indistinguible de otras utilizadas diariamente para miles de millones de transacciones por Internet. Hemos expuesto una implementación que lo hace posible, enviando los mensajes en claro y dejando abierta la factibilidad de cifrarlos. La implementación presentada permite enviar mensajes desde el servidor al cliente pero dejamos sentado que, con los mismos principios, puede construirse una comunicación bidireccional, gracias a que las peticiones Heatbeat pueden ser lanzadas desde ambos extremos.

Ha quedado por lo tanto probado el hecho de que en una organización, un empleado infiel puede filtrar información de manera prácticamente indetectable. Un administrador competente debería considerar las conclusiones de este trabajo para evaluar los riesgos subyacentes y tomar las medidas que considere pertinentes.

El estudio en profundidad de dichas medidas ha quedado fuera del alcance del presente trabajo, aunque consideramos de relevancia la realización de futuras investigaciones sobre el tema.

Agradecimientos

Queremos expresar nuestro agradecimiento al apoyo y supervisión que hemos recibido por parte de Hugo Pagola y Hernán Abbamonte, a cargo de la materia Seguridad en Redes I de la Maestría en Seguridad Informática de la Universidad de Buenos Aires, bajo cuyo ámbito se realizó el presente trabajo.

Referencias

[1] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," Internet Requests for Comments, RFC Editor, RFC 5246, August 2008, (Consultada en Febrero de 2016). [Online]. Available: https://tools.ietf.org/rfc/rfc5246.txt

[2] S. A. Thomas, *SSL and TLS Essentials*. John Wiley & Sons, Inc., 2000.

[3] R. Seggelmann, M. Tuexen, and M. Williams, "Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension," Internet Requests for Comments, RFC Editor, RFC 6520, February 2012, (Consultada en Febrero de 2016). [Online]. Available: https://tools.ietf.org/rfc/rfc6520.txt

[4] B. W. Lampson, "A note on the confinement problem," *Commun. ACM*, vol. 16, no. 10, pp. 613–615, Oct. 1973, (Consultada en Febrero de 2016). [Online]. Available: http://doi.acm.org/10.1145/362375.362389

[5] "CVS Log de OpenBSD. Disable Segglemann's RFC520 hearbeat." http://cvsweb.openbsd.org/cgi-bin/cvsweb/src/lib/libssl/ssl/Makefile?rev=1.29&content-type=text/x-cvsweb-markup, (Consultada en Mayo de 2016).

[6] "Openssl source code," https://www.openssl.org/source/, (Consultada en Febrero de 2016).

[7] M. J. Bach, *The Design of the UNIX Operating System*, 1st ed. Prentice Hall.

[8] D. MacKenzie, "Man page de mknod(1)," http://man7.org/linux/man-pages/man1/mknod.1.html, (Consultada en Febrero de 2016).

[9] "Man page de fifo(7)," http://man7.org/linux/man-pages/man7/fifo.7.html, (Consultada en Febrero de 2016).

[10] S. D. Pate, *UNIX Internals: A Practical Approach*, 1st ed. Addison Wesley Longman.

[11] U. Vahalia, *UNIX Internals: The New Frontiers*, 1st ed. Prentice Hall.

[12] S. D. Pate, *UNIX Filesystems: Evolution, Design, and Implementation*, 1st ed. Wiley.

[13] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1996.

[14] R. E. Bryant and D. R. O'Hallaron, *Computer Systems: A Programmer's Perspective*, 1st ed. Prentice Hall.

[15] "hb-test.py de takeshixx en github," https://gist.github.com/takeshixx/10107280, (Consultada en Febrero de 2016).