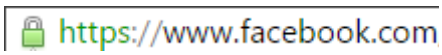
HDC


Secure Sockets Layer
SSL
Secure Sockets Layer
Secure Sockets

¿Qué es SSL?

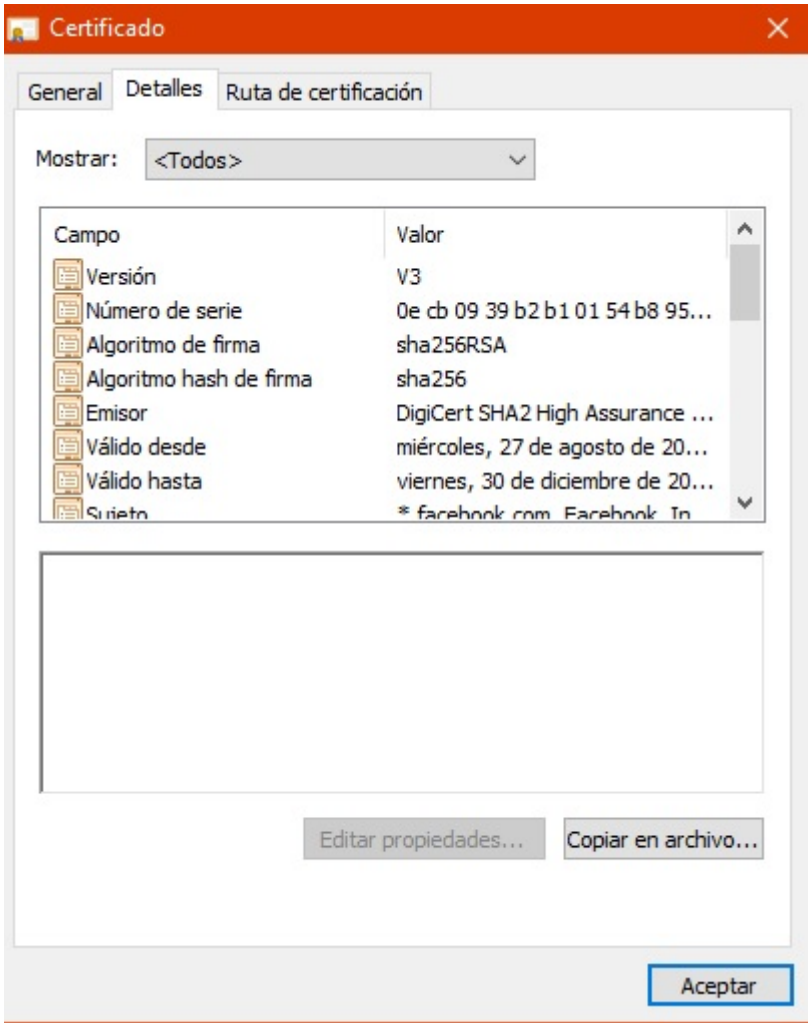
Es el **protocolo seguro de comunicación**, que permite **autenticación y privacidad** sobre Internet. Por cosas como éstas, podemos entrar a Facebook sin que nadie vea qué es lo que hacemos aunque esté intentando escuchar. Aunque sea así de segura, le corresponden fallas y agujeros de seguridad, por lo que se ha desarrollado **TLS** y hoy en día la actualización más nueva es la **1.2** (implementada desde 2008). Hablamos de que la 1.0 fue definida en el RFC a principios del '99, así que entiendan que los estándares duran varios años.

Algo que me gustaría ver, antes de adentrarnos en cuestiones técnicas, es esto. Abran sus navegadores y entren a una red social como Twitter o Facebook. Al lado de la dirección de la web, del lado izquierdo nos aparece un ***candadito***.



Ese ***candadito*** es lo que nos dice **"Ey, esta comunicación es segura"**, y además nos marca con verde el "**HTTPS**" porque estamos en un protocolo seguro para acceder a la web. Si le damos **click** al **candado** (ojo que cada navegador tiene su manera de visualizar los certificados. Estoy usando Chrome en este caso y se logra acceder con click derecho), nos encontramos con esto:

Bueno, tenemos **2 cosas que ver**. Primero que un supuesto **DigiCert SHA2** blablabla **emitió el certificado** y que lo **aprueba**. ¿Si nos metemos en los **datos del certificado**?

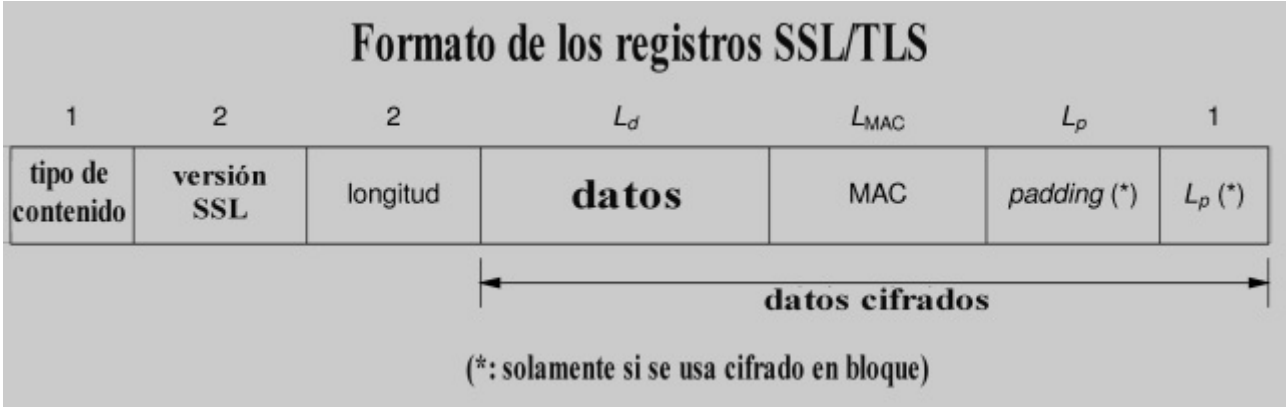


Nos aparece **todos los datos del mismo**. Es decir, el logaritmo usado para el hash de la firma, la versión, hasta que fecha es válido e incluso la clave pública. No se ustedes pero a mi siempre me pareció muy interesante que este tipo de información sea **pública**.

En la **segunda parte** nos dice **dos cosas** que nos importan. **La version de TLS (1.2)** y el **mecanismo de intercambio de clave simétrica**, que es el **ECDHE_ECDSA**. Esto corresponde a **“Curva elíptica de Diffie-Hellman”** - **“Curva elíptica de algoritmo de firma digital”**. Aunque creo que por ahora voy a dejar las curvas elípticas y la generación de claves para otro día porque merecen una clase entera para ellas solas. Además en la última clase profundicé RSA y me han llegado cartas de amenazas.(?)

REGISTROS SSL/TLS

Los registros se dividen de esta manera:



1) Campo que **indica qué es lo que viene.** Un error, datos, cambio en el protocolo o cambio de cifrado.

2) **Versión del protocolo.**

3) **Longitud del registro**, así verifican errores.

4) **Datos**. Desde aquí, todo lo que va en el registro está cifrado.

5) Código de autenticación **MAC**.

   "¿MAC? Le envía la dirección de la placa de red, otra vez. Eso no tiene sentido."

Aquí es distinto. Con **MAC**, quiero decir **Message Authentication Code**. No ahondaremos demasiado en eso. Simplemente es un **cálculo** que se realiza tomando varios datos del registro que estamos enviando y aplicando una **función hash**. Algo así como **un código de seguridad**.

6) Lo demás son **bytes de relleno** por si se usa alguna técnica de cifrado en bloque. Es decir, si cifra de a 8 bytes, pero usamos 30 bytes de datos, entonces necesitamos poner 2 bytes de relleno para llegar a los 32 y que sea múltiplo de 8. A esto se lo denomina **padding**.

7) El último campo nos dice **cuántos bytes de padding hay**.

PROTOCOLO DE NEGOCIACION

Como vimos en SSH, examinaremos el protocolo de negociación de una comunicación por SSL. Son **10 etapas importantes para la apertura de un canal nuevo**, o **4 si es un canal que alguna vez ya fue usado**. Voy a explicarlas paso por paso, de a poco.

1. **Hello Request:** Este mensaje es optativo. Generalmente no sucede, pero el server puede enviar hacia el cliente un **request de inicio de sesión**.

2. **Client Hello:** El cliente envía este mensaje que es el **saludo de inicio**, conteniendo: **versión del protocolo** que quiere usar, una cadena de **32 bytes aleatorios**, lista de **algoritmos criptográficos** preferentes, lista de **algoritmos** de **compresión** y (opcionalmente) **ID de una anterior sesión** para repetir parámetros. El ID es un número identificador.

3. **Server Hello:** El servidor recibe el “hola” inicial del cliente y manda su **respuesta con otro saludo**. La información contenida es: **versión de protocolo** a usar, otra cadena de **32 bytes aleatorios**, **ID de la sesión actual** (en caso de querer usar las mismas preferencias de un ID anterior enviado por el cliente, usa el mismo ID), **combinación de algoritmos criptográficos y de compresión**. En caso de que el server no envíe ningún ID es para no dejar que el cliente use el restablecimiento de la configuración en una conexión futura.

4. **Certificate o Server Key Exchange:** Para **autenticarse**, el **server** envía este mensaje **Certificate** con un certificado **X.509** (más adelante lo profundizaremos, pero piensen como si fuera una planilla de validación que certifica que es él mismo) o una cadena de certificados. **Si el servidor no tiene certificado, o acordó un método de intercambio de claves que no lo necesita**, debe mandar un **Server Key Exchange.** Desde este paso, sólo es necesario para una sesión nueva. Para reemprenderla, no es necesario tanto barullo.

5. **Certificate Request:** En caso de que el cliente deba decir quién es, entonces el server le manda este mensaje **exigiéndole** que lo **certifique**.

6. **Server Hello Done:** Para **terminar** el diálogo del handshake.

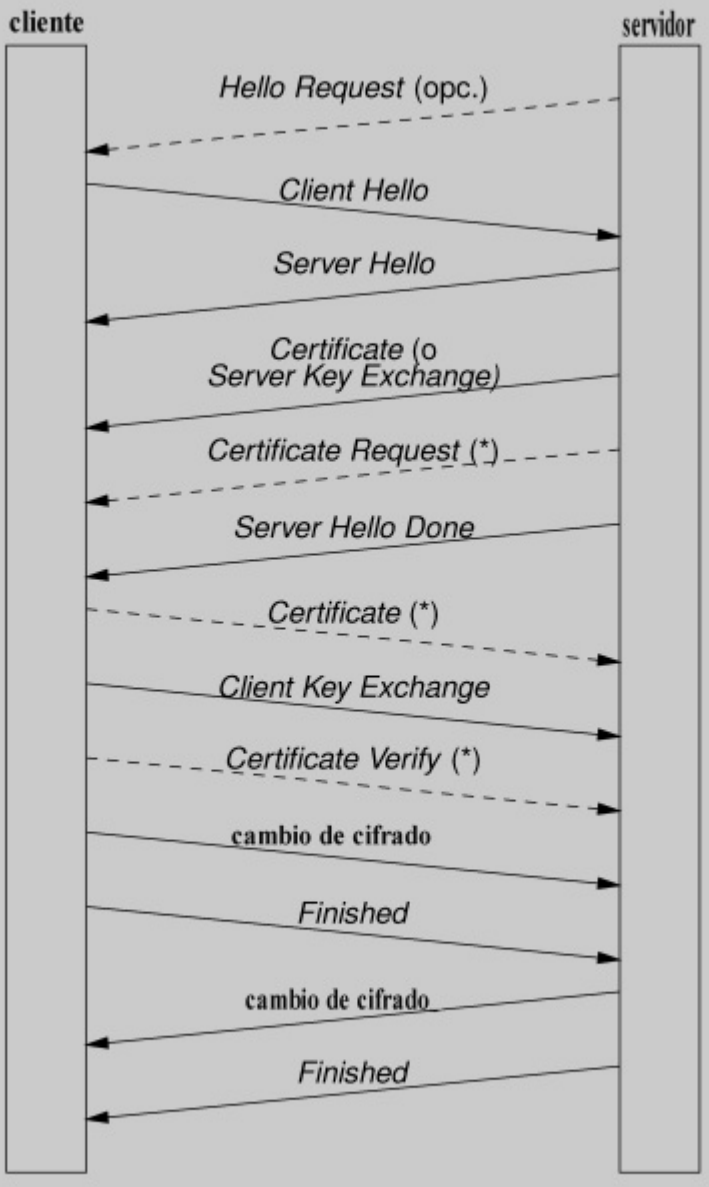
7. **Certificate:** Si el server le envío un mensaje de Certificate Request, entonces el cliente se lo envía.

8. **Client Key Exchange:** Aquí el cliente envía un contenido de claves como las **claves de cifrado simétrico previamente cifradas con la clave pública del servidor**. Como vimos en la clase anterior, es posible hacerlo con RSA.

9. **Certificate Verify:** La idea de este mensaje es que el **cliente envíe** este mensaje en conjunto con el Certificate para **demostrar** que fue él quien lo envío. ¿Cómo? Simple. **Firmándolo con su clave privada**. Recuerden que si le aplicamos la clave pública, entonces verificaremos que es él.

10. **Finished:** Con este mensaje se completa un handshake exitoso. Desde aquí se pueden enviar datos uno con el otro.

Negociación de una sesión SSL/TLS nueva



(*: solamente si se realiza autenticación de cliente)

¿Qué ~~corno~~ es x.509?

Es un **estándar**. Una manera de cómo están organizados los datos de los certificados para que todos lo hagan de una misma manera. En este caso están organizados así:

- **Versión**
- **Número de serie**
- **ID del algoritmo**
- **Emisor**
- **Fechas de validez**
- **Sujeto (a quién valida)**
- **Clave pública del Sujeto**
- **Datos opcionales**
- **Algoritmo usado para firmar el certificado**
- **Firma del certificado**

Vamos directamente a un ejemplo de un certificado:

Certificate:

Data:
Version: 1 (0x0)
Serial Number: 7829 (0x1e95)
Signature Algorithm: md5WithRSAEncryption
Issuer: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,
OU=Certification Services Division,
CN=Thawte Server CA/Email=server-certs@thawte.com
Validity
Not Before: Jul 9 16:04:02 1998 GMT
Not After : Jul 9 16:04:02 1999 GMT
Subject: C=US, ST=Maryland, L=Pasadena, O=Brent Baccala,
OU=FreeSoft, CN=www.freesoft.org/Email=baccala@freesoft.org
Subject Public Key Info:
Public Key Algorithm: rsaEncryption
RSA Public Key: (1024 bit)
Modulus (1024 bit):
00:b4:31:98:0a:c4:bc:62:c1:88:aa:dc:b0:c8:bb:
33:35:19:d5:0c:64:b9:3d:41:b2:96:fc:f3:31:e1:
66:36:d0:8e:56:12:44:ba:75:eb:e8:1c:9c:5b:66:
70:33:52:14:c9:ec:4f:91:51:70:39:de:53:85:17:
16:94:6e:ee:f4:d5:6f:d5:ca:b3:47:5e:1b:0c:7b:
c5:cc:2b:6b:c1:90:c3:16:31:0d:bf:7a:c7:47:77:
8f:a0:21:c7:4c:d0:16:65:00:c1:0f:d7:b8:80:e3:
d2:75:6b:c1:ea:9e:5c:5c:ea:7d:c1:a1:10:bc:b8:
e8:35:1c:9e:27:52:7e:41:8f
Exponent: 65537 (0x10001)
Signature Algorithm: md5WithRSAEncryption
93:5f:8f:5f:c5:af:bf:0a:ab:a5:6d:fb:24:5f:b6:59:5d:9d:
92:2e:4a:1b:8b:ac:7d:99:17:5d:cd:19:f6:ad:ef:63:2f:92:
ab:2f:4b:cf:0a:13:90:ee:2c:0e:43:03:be:f6:ea:8e:9c:67:
d0:a2:40:03:f7:ef:6a:15:09:79:a9:46:ed:b7:16:1b:41:72:
0d:19:aa:ad:dd:9a:df:ab:97:50:65:f5:5e:85:a6:ef:19:d1:
5a:de:9d:ea:63:cd:cb:cc:6d:5d:01:85:b5:6d:c8:f3:d9:f7:
8f:0e:fc:ba:1f:34:e9:96:6e:6c:cf:f2:ef:9b:bf:de:b5:22:
68:9f

Gracias Wikipedia por el certificate.

En negrita están remarcados los campos que deben ir. Pero no nos metamos más en esto.

Vulnerabilidades

Cada vez que SSL es vulnerado, la voz se pasa rápidamente y se aprovecha por muchos internautas curiosos. Pero generalmente el parche se libera al poco tiempo -relativamente hablando- así que no esperen poder aprovecharse de esto deliberadamente. Veremos el famoso bug que recorrió las noticias informáticas en Abril del 2014: **Heartbleed**.

La idea es que el **handshake** que vimos anteriormente es un proceso que no se quisiera hacer demasiadas veces, porque lleva **tiempo** y **recursos** computacionales. Entonces se creó el **HeartBeat** (latido del corazón), un mensaje que envía el cliente al servidor para hacerle saber que **aún está ahí**, comunicándose con él. El mensaje consta de una conversación parecida a ésta:

> **Cliente**: -Ey, server. ¿Estás activo todavía? Si es así, responde con la palabra "auto". Son 4 letras.
>
> **Servidor**: -auto

Si se fijan bien, el cliente dice **cuantas letras tiene el mensaje**. Claro, en el intercambio de datos normal estaríamos hablando de una **longitud en bytes**.

> ***"Es necesario saber eso?"***

Bueno, sí. Es justamente el fallo. Cuando el cliente dice que da una cantidad de letras **mayor a lo que es la palabra**, el server se confunde porque además de mandar la

palabra de respuesta, **rellena esa cantidad de "letras" con las que tiene en memoria en ese momento**. Entonces la conversación pasaría a ser algo así:

> **Cliente**: -Ey, server. ¿Estás activo todavía? Si es así, responde con la palabra "auto". Son **50** letras.
>
> **Servidor**: -auto. Perro, caniche. Admin. User. Passwords.....

Claro. Aquí en nuestro ejemplo, el server tiene palabras sueltas pero en la vida real, **en memoria, el server podría tener credenciales, contraseñas, logs, información de sistema requerida, acciones de los usuarios, etc.**

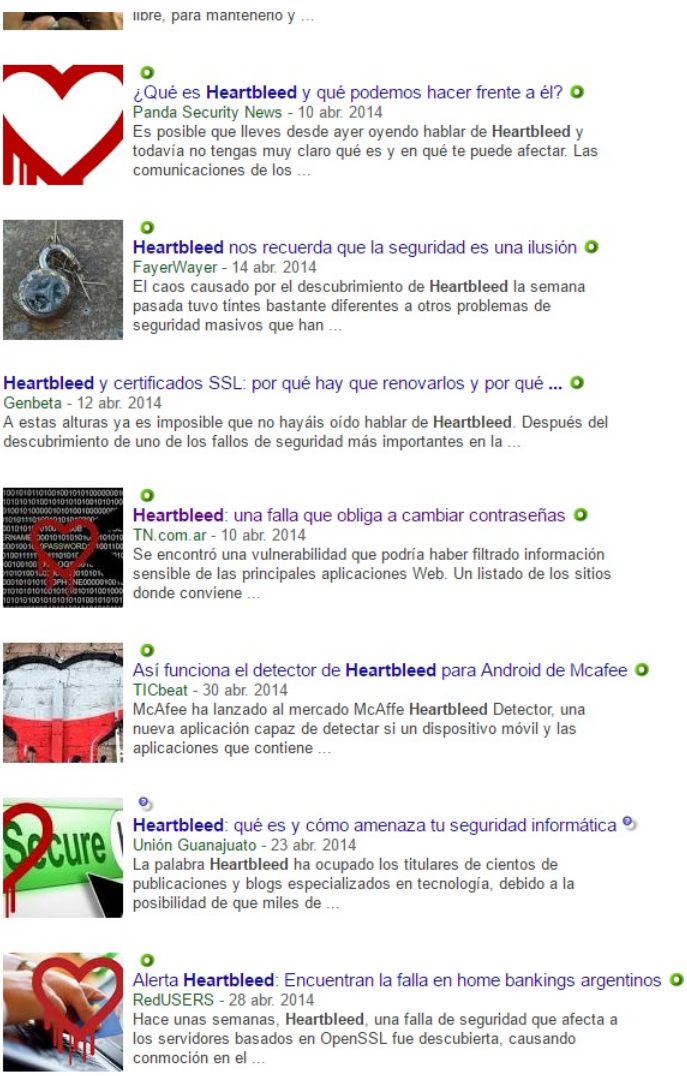
> ***"Me imagino que el registro sólo puede mandar una cantidad de datos limitados."***

Y claro. Tiene un limite, de 64KB.

> ***"Entonces el fallo es grave pero es difícil de recolectar muchos datos."***

En realidad, si fuesen siempre el mismo segmento de memoria no sería una inmensidad. Pero la verdad es que siempre que se hace un pedido para que el servidor responda, **la dirección de memoria cambia**. Esto es igual a **poder descargar GB de memoria en poco tiempo**. Imaginen la catástrofe que fue cuando se descubrió.

Lo importante es siempre mantener actualizado el sistema con el cual tenemos salida a internet :), leer las noticias e intentar de usar sistemas seguros.



En fin. Eso es todo por hoy :). Espero que lo hayan disfrutado (y aprendido sobre todo) y nos vemos en la siguiente clase. :D

Pueden seguirme en Twitter: @RoaddHDC

Contactarse por cualquier duda a: r0add@hotmail.com

Para donaciones, pueden hacerlo en bitcoin en la dirección siguiente:

1HqpPJbbWJ9H2hAZTmpXnVuoLKkP7RFSvw

Roadd.