Proyecto de Fin de Posgrado

Estudio del protocolo TLS (Transport Layer Security)

Leonor Priego García

Posgrado de Seguridad en Redes y Sistemas

Consultora: Cristina Pérez Solà

3 de junio de 2018

Agradecimientos

A mi familia y a Marcos por estar siempre ahí, apoyándome.

Licencia

Resumen

El objetivo de este proyecto ha sido la familiarización con el protocolo TLS (Transport Layer Security) a través del estudio de los conceptos criptográficos sobre los que se basa, su evolución y las fases que llevan a cabo este protocolo para cifrar y descifrar los datos en las comunicaciones en red. Así mismo, se explican los diferentes ataques que han conseguido vulnerar tanto el protocolo TLS como su predecesor SSL (Secure Sockets Layer) y qué vulnerabilidades han conseguido aprovechar.

Por otro lado, se detallan las mejoras tanto en velocidad como en seguridad que introduce la versión TLS 1.3, la cual se encuentra en fase de último borrador y está destinada a convertirse en el nuevo estándar de TLS.

Por último, se explica cómo se combinan los protocolos HTTP (Hypertext Transfer Protocol) y TLS para establecer un túnel seguro entre ambos intervinientes en la comunicación. Así como exponer las deficiencias que tiene el binomio HTTP+TLS y detallar y comparar éste con protocolos alternativos como QUIC, HTTP/2, DTLS y SPDY.

Este proyecto será puesto a disposición de todo aquel que quiera adentrarse en el protocolo TLS y desee continuar el proyecto, ya sea siguiendo las líneas futuras del mismo o cualquier otra, pues en el contexto de esta línea de investigación, esta memoria supone una base sólida de conocimiento sobre transferencia segura de datos en red y un punto de partida para investigaciones en profundidad de cualquiera de los aspectos recogidos en ella.

Palabras clave: TLS, SSL, cifrado, descifrado, encriptación, desencriptación, túnel seguro, protocolo, HTTPS, comunicaciones seguras.

Abstract

The objective of this project has been the familiarization with the TLS protocol (Transport Layer Security) through the study of the cryptographic concepts on which it is based, its evolution and the phases that needs this protocol to encrypt and decrypt the data in the network communications. Likewise, the different attacks that have achieved to infringe the TLS protocol and its predecessor SSL (Secure Sockets Layer) and which vulnerabilities have been exploited are explained.

On the other hand, the improvements in speed and safety introduced by the TLS 1.3 version are detailed, TLS 1.3 is in the final draft phase and is destined to become the new TLS standard.

Finally, it explains how the HTTP (Hypertext Transfer Protocol) and TLS protocols are combined to establish a secure tunnel between both parties in the communication. As well as exposing the deficiencies of the HTTP + TLS and detailing and comparing they with alternative protocols such as QUIC, HTTP/2, DTLS and SPDY.

This project will be made available to anyone who wants to discover the TLS protocol and wish to continue the project, either following the future lines of the same or any other, because in the context of this line of research, this memory is a solid base of knowledge about secure data transfer in the network and a starting point for in-depth investigations of any of the aspects included in it.

Key words: TLS, SSL, encryption, decryption, secure tunnel, protocol, HTTPS, secure communications.

Índice general

Índice de tablas y figuras

Figuras

Tablas

Capítulo 1. Introducción

1.1 Descripción del problema y motivación

TLS (Transport Layer Security) es un protocolo criptográfico que proporciona mecanismos de encriptación y autenticación entre máquinas que se comunican mediante la red. Se publicó en enero de 1999 como una versión nueva de su protocolo predecesor SSL (Secure Sockets Layer) que se introdujo en 1995 con la versión SSLv2, y que en cuestión de un año, fue renovado y reforzado por una nueva versión, SSLv3. TLS ha pasado por tres versiones tras encontrarse con diferentes vulnerabilidades y mejoras que permitían crear un protocolo más seguro y robusto, actualmente se utiliza la versión TLS 1.2, sin embargo, en marzo de 2018 se publicó el último borrador de la versión TLS 1.3.

Así mismo, el protocolo HTTPS (Hypertext Transfer Protocol Secure) se basa en el protocolo HTTP y añade un cifrado SSL/TLS que permite la transferencia segura de datos en las comunicaciones en red. Este protocolo cuenta con algunas carencias a la hora de operar y que se ven resueltas en protocolos alternativos como SPDY, HTTP2, DTLS y QUIC.

Con la realización de este proyecto se pretende llevar a cabo el estudio, recopilación de información y desarrollo de una documentación unificada del protocolo criptográfico TLS, a partir de diferentes fuentes, que cubra aspectos relacionados con la comparación de este protocolo con la nueva versión TLS 1.3 y con alternativas que pretenden cubrir diferentes carencias o problemas de la utilización de TLS junto al protocolo de comunicación HTTP.

Por otro lado, también se pretende reunir documentación tanto de los ataques que se han producido contra SSL/TLS como las vulnerabilidades que aprovecharon estos ataques, haciendo hincapié en cómo los malware actuales son capaces de conseguir adoptar este protocolo para poder asegurar sus comunicaciones y viajar sobre un canal seguro.

La motivación principal para la realización de este Proyecto de Fin de Posgrado, ha sido la idea estudiar cómo viajan los datos ocultos a través de Internet, es decir, de qué manera se cifran, qué debilidades se encuentran y de qué manera se pueden aprovechar. Esta idea siempre ha estado rondando en mi cabeza, puesto que muchas veces enviamos datos confidenciales por las redes, desde fotocopias del DNI hasta los datos de la tarjeta de crédito y realmente no solemos ser conscientes de la protección que llevan esos datos y si es fácil llevar a cabo ataques que puedan recuperarlos.

Como estoy estudiando el Posgrado de seguridad en redes y sistemas, considero la ejecución de este proyecto bastante adecuada y útil para tener una base sólida de la seguridad en la transmisión de los datos mediante la red de Internet.

1.2 Estado del arte

La versión del protocolo criptográfico TLS 1.0 fue definida en el RFC 2246[2] en enero de 1999, no obstante, pese a ser un protocolo clave en las comunicaciones seguras, no se ha encontrado ninguna guía o documentación unificada de todos los aspectos que se describen en este proyecto.

Se han encontrado diferentes guías sobre TLS[1][2] y SSL y comparativas entre ambos[3] donde se especifican qué carencias o vulnerabilidades mejora cada una de las nuevas versiones. No obstante, como ya existen diferentes artículos donde se explican los conceptos de TLS y SSL, no centraremos en ello el objetivo del proyecto sino que se hará hincapié en la explicación de ambos, para focalizarlo más a aspectos más prácticos como comparativas o ataques a ambos.

En marzo de 2018 se publicó el último borrador del protocolo TLS 1.3, que está determinado a convertirse en el nuevo protocolo de seguridad estándar de Internet. Hasta ahora lo más sólido y fiable que se puede encontrar es el borrador del IETF (Internet Engineering Task Force), que se tomará como referencia para este proyecto.

Por otro lado, encontramos ataques al protocolo TLS/SSL que explotan el algoritmo de cifrado RSA[4][5] y resulta interesante porque explica cómo los atacantes pueden invertir la encriptación RSA. También, se encuentran indicaciones de diferentes vulnerabilidades tanto en TLS [6][7] como en SSL pero no se llega a encontrar información unificada y detallada sobre cómo los diferentes malwares que consiguen aprovechar la encriptación de este protocolo para que sus datos viajen sobre canales seguros.

Finalmente, en diferentes artículos se indica que el protocolo QUIC[8] o HTTP/2[9] resuelven problemas que aparecen en el protocolo HTTPS, no obstante, no hay presencia de documentación detallada ni estudios sobre los problemas del binomio HTTP y TLS.

Por ello, se recaba la información más importante de todos los estudios e información encontrada y se detalla todo en un único documento. De manera que la documentación recogida en este proyecto suponga un punto de partida para

diferentes investigaciones en profundidad sobre el protocolo TLS e información importante y descentralizada en Internet.

1.3 Objetivos

El objetivo general del proyecto es el estudio del protocolo TLS. Para abordar este objetivo principal, se han fijado los siguientes subobjetivos:

1.- Describir el funcionamiento del protocolo TLS.

2.- Estudiar y detallar los ataques a este protocolo y sus respectivas vulnerabilidades que han permitido que éstos hayan podido llegar a producirse.

3.- Estudiar los malwares que han conseguido adoptar este protocolo para asegurar sus comunicaciones.

4.- Realizar una comparativa del protocolo TLS con su predecesor SSL.

5.- Estudiar y describir los problemas detectados en el binomio HTTP y TLS.

6.- Comparar HTTPS con protocolos SPDY, HTTP/2, DTLS y QUIC.

7.- Estudiar y detallar en qué consiste la nueva y reciente versión de TLS, TLS 1.3, así como las contramedidas y mejoras que propone para mitigar los ataques.

1.4 Metodología

La metodología seguida durante el desarrollo del proyecto ha estado compuesta por dos bloques fundamentales:

Los objetivos han sido llevados a cabo en orden. En primer lugar, se empezó por analizar en profundidad el protocolo TLS, su base criptográfica y su funcionamiento, para posteriormente, analizar sus debilidades o vulnerabilidades y qué ataques las han aprovechado.

Una vez se tuvo una base sólida sobre el protocolo TLS, se identificaron e investigaron si existen malwares que consiguen adoptar este protocolo para viajar sobre canales seguros, para posteriormente describir cómo lo consiguen. Así mismo, se comentan qué medidas propone el protocolo TLS 1.3 para corregir las

vulnerabilidades que han propiciado que el protocolo TLS haya sido tan atacado a lo largo de los años.

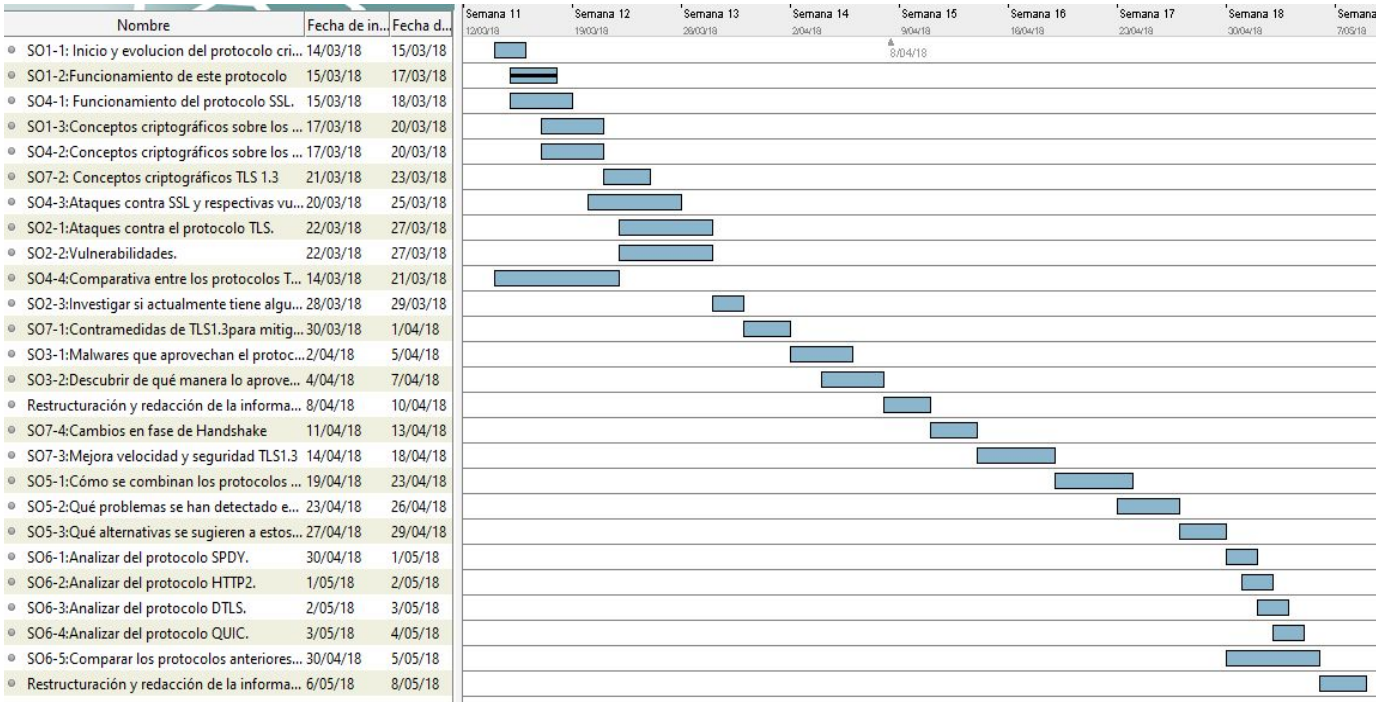
En el siguiente bloque, se compara cómo ha cambiado la negociación Handshake, en qué consiste el modo 0-RTT que permite ahorrar un viaje de ida y vuelta en la configuración de la conexión para algunos datos de la aplicación, a costa de ciertas propiedades de seguridad y aspectos que consideremos relevantes de esta nueva versión TLS 1.3.

Así mismo, se estudiarán los problemas que tiene la combinación de HTTP y TLS (como la lentitud en el establecimiento de la conexión, no enfocado a sesión, etc) y se comparará con protocolos alternativos como HTTP/2, DTLS, QUIC, SPDY.

Toda la investigación y estudio se documenta en la memoria de este proyecto. Así mismo, se referencia toda la bibliografía consultada.

1.5 Planificación

La planificación temporal detallada de estas tareas y sus dependencias se representa en la siguiente figura mediante un diagrama de Gantt.



Nombre	Fecha de in...	Fecha d...
SO1-1: Inicio y evolucion del protocolo cri...	14/03/18	15/03/18
SO1-2:Funcionamiento de este protocolo	15/03/18	17/03/18
SO4-1: Funcionamiento del protocolo SSL.	15/03/18	18/03/18
SO1-3:Conceptos criptográficos sobre los ...	17/03/18	20/03/18
SO4-2:Conceptos criptográficos sobre los ...	17/03/18	20/03/18
SO7-2: Conceptos criptográficos TLS 1.3	21/03/18	23/03/18
SO4-3:Ataques contra SSL y respectivas vu...	20/03/18	25/03/18
SO2-1:Ataques contra el protocolo TLS.	22/03/18	27/03/18
SO2-2:Vulnerabilidades.	22/03/18	27/03/18
SO4-4:Comparativa entre los protocolos T...	14/03/18	21/03/18
SO2-3:Investigar si actualmente tiene algu...	28/03/18	29/03/18
SO7-1:Contramedidas de TLS1.3para mitig...	30/03/18	1/04/18
SO3-1:Malwares que aprovechan el protoc...	2/04/18	5/04/18
SO3-2:Descubrir de qué manera lo aprove...	4/04/18	7/04/18
Restructuración y redacción de la informa...	8/04/18	10/04/18
SO7-4:Cambios en fase de Handshake	11/04/18	13/04/18
SO7-3:Mejora velocidad y seguridad TLS1.3	14/04/18	18/04/18
SO5-1:Cómo se combinan los protocolos ...	19/04/18	23/04/18
SO5-2:Qué problemas se han detectado e...	23/04/18	26/04/18
SO5-3:Qué alternativas se sugieren a estos...	27/04/18	29/04/18
SO6-1:Analizar del protocolo SPDY.	30/04/18	1/05/18
SO6-2:Analizar del protocolo HTTP2.	1/05/18	2/05/18
SO6-3:Analizar del protocolo DTLS.	2/05/18	3/05/18
SO6-4:Analizar del protocolo QUIC.	3/05/18	4/05/18
SO6-5:Comparar los protocolos anteriores...	30/04/18	5/05/18
Restructuración y redacción de la informa...	6/05/18	8/05/18

Figura 1: Planificación temporal en Diagrama de Gantt

1.6 Estructura del documento

El presente documento está estructurado en 6 capítulos en los que se abordan los distintos aspectos contemplados para la elaboración de este Proyecto de fin de posgrado.

El capítulo 1 está dedicado introducir del proyecto, exponiendo el problema a resolver, los motivos que han llevado a realizarlo, los objetivos marcados a alcanzar con su realización y la metodología y planificación llevada a cabo.

El capítulo 2 abarca el estudio del protocolo TLS, cuáles fueron sus inicios y cómo ha ido evolucionando. Se definen una serie conceptos criptográficos fundamentales para poder explicar el funcionamiento de este protocolo. También se hace hincapié en explicar los ataques más importantes que ha sufrido SSL/TLS y qué vulnerabilidades han aprovechado.

El capítulo 3 reúne las mejoras en seguridad y velocidad que introduce el protocolo TLS 1.3 con respecto a la versión anterior y actualmente vigente. Se presentan también los mecanismos teóricos para evitar ataques por repetición cuando se utiliza la técnica 0-RTT y qué medidas se llevan a cabo en la práctica.

El capítulo 4 se encarga de explicar cómo se combinan los protocolos HTTP y TLS para asegurar las comunicaciones en la red, el uso de certificados SSL/TLS para cifrar los datos y se recogen los problemas detectados en el protocolo HTTP en sus primeras versiones y qué alternativas existen para mejorar la eficiencia y eficacia del binomio HTTP+TLS.

En el capítulo 5 se explican los protocolos alternativos a HTTPS y se comparan con éste.

Por último, el capítulo 6 se encarga de recoger las principales conclusiones derivadas de este proyecto de fin de posgrado, así como las posibles líneas abiertas de investigación que se pudieran desarrollar en un futuro.

Capítulo 2. Protocolo criptográfico TLS. Conceptos, ataques y vulnerabilidades

2.1 Inicio y evolución

El protocolo TLS (Transport Layer Security) fue definido en enero de 1999 en el estándar RFC 2246[2] por Internet Engineering Task Force (IETF). TLS es una actualización del protocolo SSL (Secure Sockets Layer), el cual fue desarrollado por Netscape para asegurar la transferencia de datos entre máquinas de la red, aunque principalmente surgió para asegurar las transacciones en el comercio electrónico y permitir de esta manera poder proteger y encriptar los datos de los clientes que viajasen por la red.

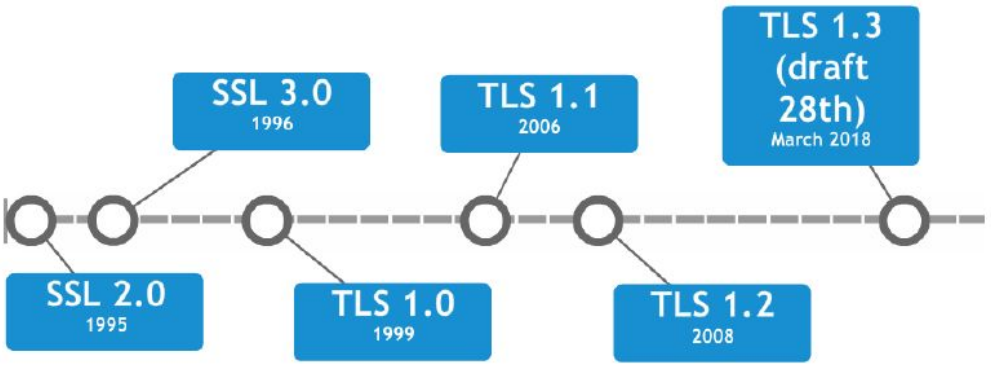


Figura 2: Eje cronológico de la evolución de SSL/TLS

Siguiendo la línea temporal de la figura 2, la primera versión publicada del protocolo SSL fue la versión 2.0 en 1995, no obstante, al cabo de un año fue reemplazada por la versión 3.0 debido a diferentes fallos de seguridad y ataques que había tenido que soportar. Actualmente la versión SSL 2.0 está desactivada por defecto en los navegadores por ser considerada insegura.

La versión de SSL 3.0 mejoró el cifrado y la autenticación con respecto a la versión 2.0, pero como este protocolo era propiedad de Netscape, el IETF lo versionó y estandarizó en 1999 en el RFC 2246[2] y le dio el nombre de TLS 1.0. Según especifica el RFC, “las diferencias entre este protocolo y SSL 3.0 no son dramáticas, pero son suficientemente significativas como para impedir la interoperabilidad entre TLS 1.0 y SSL 3.0”.

Si bien, el IETF ha seguido trabajando y mejorando este protocolo corrigiendo vulnerabilidades y carencias. En abril de 2006 se definió en el estándar RFC 4346[10] la versión TLS 1.1 la cual agregaba protección contra ataques CBC (

cipher-block chaining) y modificaba el vector de inicialización. Siguiendo esta línea, en agosto del 2008 se definió en el estándar RFC 5246[11] la versión de TLS 1.2, que mejoraba los algoritmos de cifrado la habilidad de clientes y servidores para especificar qué algoritmos de hash y de firma van a aceptar. No obstante, en marzo de 2011 se redefinió para que este protocolo no permitiese la retrocompatibilidad con la versión 2.0 de SSL.

Desde 2014 y tras 28 borradores, la IETF publicó en marzo de 2018 el último borrador de la versión de TLS 1.3[12], cuyo principal objetivo es permitir establecer conexiones más rápidas y seguras a través de Internet. Esta versión de TLS está destinada a convertirse en el nuevo estándar de Internet.

2.2 Conceptos criptográficos sobre los que se basa el protocolo TLS

Como veremos en el apartado 2.3, antes de establecer una conexión entre cliente y servidor, se deben intercambiar las claves públicas y acordar, entre ambos, una clave secreta con la que cifrar la información a transmitir. Este primer paso de negociación y autenticación se realiza mediante criptografía asimétrica o de clave pública (por la que no hace falta que exista un canal seguro). Una vez acordada la clave secreta, ambos utilizan dicha clave para cifrar y descifrar los mensajes, estableciendo una conexión donde los datos viajan cifrados mediante criptografía simétrica.

Para entender bien el proceso de negociación y cifrado de datos, este apartado se centra en explicar los conceptos criptográficos en los que se basa el protocolo TLS, como la confidencialidad, integridad, autenticación, tipos de cifrados, funciones hash y algoritmos de cifrado que emplea en las diferentes fases del establecimiento de la comunicación y el intercambio de mensajes.

2.2.1 Confidencialidad, integridad y autenticación

El protocolo TLS proporciona cifrado, autenticación e integridad de la información entre ambos extremos de la conexión.

Confidencialidad: Es la capacidad de garantizar que la información será protegida y solamente podrá acceder a ella la persona a quien va dirigida. TLS utiliza una combinación de cifrado simétrico y asimétrico para asegurar la confidencialidad de los mensajes. En la fase de handshake, el cliente y servidor acuerdan un algoritmo de cifrado y una clave compartida con la que cifrarán todos los mensajes

transmitidos entre ambos, garantizando confidencialidad del mensaje incluso si es interceptado. Además, para la distribución de la clave compartida utilizan cifrado asimétrica, que se explica en el apartado siguiente.

Integridad: Es la capacidad de garantizar que los datos no se modifican desde su envío hasta su recepción. TLS proporciona integridad de los mensajes enviados calculando un resumen o hash del mensaje. Este algoritmo de hash es acordado entre ambos ambos interlocutores.

Autenticación: Es la capacidad de garantizar que el interlocutor es quien realmente dice ser. Para la autenticación del servidor, el cliente utiliza la clave pública del certificado del servidor para cifrar la clave compartida secreta. El servidor puede reproducir esta clave secreta solamente si la puede descifrar con su clave privada.

Para la autenticación del cliente, el servidor utiliza la clave pública del certificado de cliente para descifrar los datos que el cliente envía. El intercambio de mensajes cifrados con la clave compartida confirma que se ha completado la autenticación.

Si la autenticación no se ejecuta correctamente, el reconocimiento falla y se finaliza la sesión.

Habitualmente, solamente se suele verificar la autenticidad o identidad del servidor, no siempre es necesario verificar al cliente.

2.2.2 Cifrado simétrico y asimétrico

En el **cifrado simétrico**, conocido como “Shared Key” o “Shared Secret”, tan sólo se utiliza una clave para cifrar y descifrar la información. Es mucho más rápido y fácil de implementar que el cifrado asimétrico.

En el **cifrado asimétrico**, conocido como “Criptografía de clave pública”, se emplean **dos claves** para cifrado y descifrado de la información. Ambos interlocutores poseen una clave pública y una clave privada, generando su clave pública en base a la clave privada.

El emisor cifra el mensaje con la clave la clave pública del destinatario, de tal manera que el destinatario puede descifrar el mensaje con su clave privada.

Las funciones de cifrado permiten garantizar la confidencialidad de la información.

2.2.3 Funciones hash

Las **funciones hash** son funciones matemáticas que crean a partir de una entrada (ya sea un texto, una contraseña o un archivo, por ejemplo) una salida alfanumérica

de longitud normalmente fija que representa un resumen de toda la información de entrada.

Esta cadena representa de manera unívoca a ese conjunto de datos y sólo puede volverse a crear con los mismos datos. El contenido generado es ilegible y son funciones que no tienen inversa, por lo que no se puede establecer el contenido del mensaje a partir del hash.

Las funciones hash permiten garantizar **la integridad** de la información.

2.2.4 Algoritmos criptográficos empleados

Para la criptografía pública, se ha utilizado RSA, Diffie-Hellman, Diffie-Hellman de Curva elíptica y Diffie Hellman de campos finitos. No obstante, la versión del protocolo TLS 1.3, deja obsoletos y elimina el uso de los algoritmos RSA y Diffie-Hellman, permitiendo solamente los algoritmos criptográficos Diffie-Hellman de Curva elíptica y Diffie Hellman de campos finitos, los cuales proporcionan Secreto Perfecto Hacia Adelante (Perfect Forward Secrecy).

Uno de los principales problemas de RSA y Diffie-Hellman es que se establece la clave de sesión secreta haciendo uso de las claves públicas y privadas de los interlocutores, siendo siempre la misma clave secreta para la comunicación entre ambos. El inconveniente que esto tiene es que las claves privadas pueden ser comprometidas en un futuro, dado que si un atacante adivina la clave secreta, puede descifrar todos los mensajes de todas las comunicaciones que hubiese grabado entre ambos extremos.

No obstante, ambos han quedado obsoletos por algoritmos que proveen Secreto Perfecto Hacia Adelante, que permite crear una clave secreta distinta para cada comunicación y que posteriormente es eliminada, volviendo imposible recuperar las claves en un futuro y descifrar todas nuestras comunicaciones.

Otro de los motivos por los que se sustituyen los algoritmos anteriores por algoritmos de curvas elípticas es porque estos sistemas utilizan curvas elípticas para crear claves criptográficamente más fuertes y de menor longitudes, lo que hace a estos algoritmos más eficientes y rápidos de implementar.

Por otro lado, para la criptografía simétrica se usaban algoritmos como RC2, RC4, IDEA, DES, Triple DES y AES con funciones hash MD5 o de la familia SHA. No obstante, en TLS 1.3, se elimina de la lista de algoritmos simétricos los algoritmos considerados heredados. Todos los algoritmos soportados utilizan encriptación autenticada con datos asociados (AEAD), que permite que el cifrado y la

autenticación sucedan concurrentemente, haciéndolo más sencillo de usar y optimizar que los modos antiguos modos como CBC (cipher block chaining).

En la fase de Handshake, el cliente envía un mensaje Client Hello como el de la figura siguiente, en el que le indica al servidor la versión el protocolo TLS, los algoritmos de cifrado, los métodos de compresión que soporta, etc.

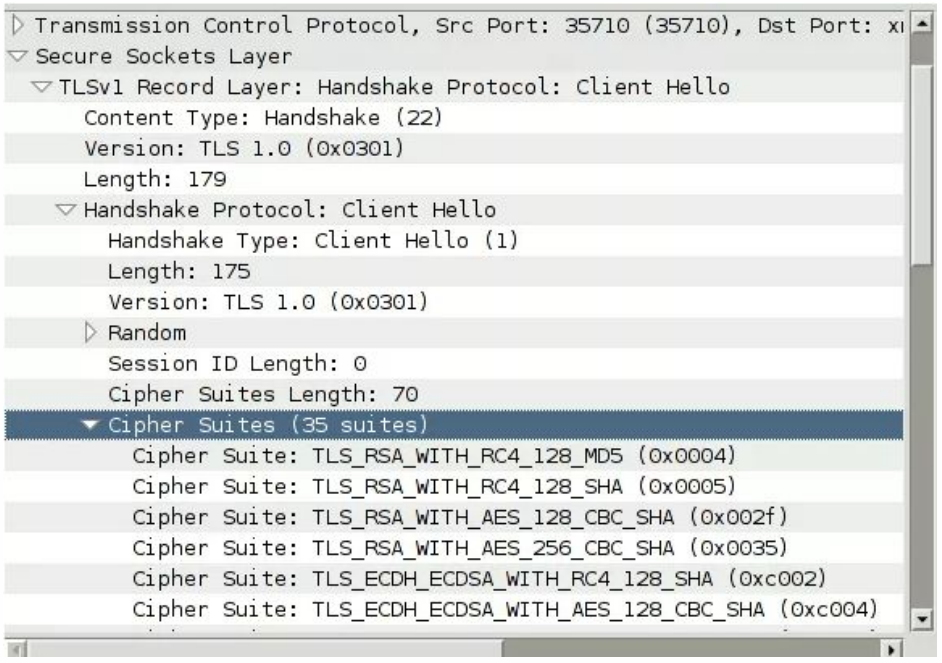


Figura 3: Captura Wireshark de un mensaje Client Hello

El servidor lo compara con su listado y entre ambos negocian los algoritmos que utilizarán durante la comunicación.

Cipher Suite

Una vez analizamos el mensaje Client Hello, observamos que en el apartado Cipher Suites se indica la lista de protocolos soportados por el cliente. En la siguiente figura se identifican los valores de un ejemplo de Cipher Suite.

Figura 4: Identificación de los valores de un mensaje Cipher Suite

Protocolo

Pueden ser SSL1, SSL2, SSL3, TLS1.0, TLS1.1, TLS1.2, TLS1.3, DTLS1.0.

Los algoritmos SSL1, SSL2, SSL3, TLS1.0 y TLS1.1 han sido catalogados como no seguros, por lo que se recomienda no usarlos.

Así mismo, a principios de 2017 se lanzó la nueva versión del estándar PCI[13] (Consejo de Normas de Seguridad de la Industria de Tarjetas de Pago), para mejorar la seguridad de las transacciones en línea. Por lo que todos los comercios y proveedores de servicios, estarán obligados a migrar a la nueva versión de TLS 1.2 antes del 30 de junio de 2018, de manera que si no se cumplen los requisitos mínimos recomendados, no se podrán realizar transacciones en línea (por ejemplo, pagos por internet), ni operar aplicaciones locales que utilicen cifrado TLS, como pueden ser aplicaciones de correo, de proveedores, etc.

Intercambio de claves (Key Exchange). Indica el algoritmo a emplear para compartir la clave simétrica secreta con la que se cifran los mensajes.

Pueden ser DH, DHE, ECDH, ECHE, RSA, PKCS, DSA, CK.

No obstante, en TLS1.3 solo se contemplan ECDH, ECDHE.

Firma digital (Digital Signature). Garantiza que los datos no han sido alterados.

Pueden ser RSA, ECDSA, DSS.

TLS1.3 propone dos nuevos algoritmos: ed25519 y ed448.

Algoritmo de encriptación simétrico

Pueden ser AES, RC2, RC4, RC5, DES, 3DES, IDEA, FORTEZZA.

En TLS 1.2 se consideran inseguros RC2, RC4 y DES. TLS1.3 tan sólo soporta algoritmos que utilizan encriptación autenticada con datos asociados (AEAD).

Modo de encriptación. Dependiendo del tipo de algoritmo de encriptación elegido.

Pueden ser CBC, GCM, ECB, PCBC, CFB, OFB, CTR.

Algoritmo de hash. Validación de integridad del mensaje.

Pueden ser SHA, SHA256, SHA384, AEAD, HMAC, MD2, MD4, MD5, MDC2, RIPEMD-160.

En TLS1.2 se consideran inseguros MD2, MD4 y MD5.

TLS 1.3 soporta HMAC y AEAD, añade algoritmos seguros e imposibles de romper actualmente como ChaCha20, Poly1305, Ed25519, x25519 y x448 y elimina algoritmos inseguros como MD5 y SHA-224.

Tamaño de curva elíptica. Dependiendo de si se escogió un algoritmo de curva elíptica en el intercambio de claves, la curva puede ser de dos tipos: P256 y P384.

2.3 Funcionamiento y fases

Antes de entender cómo funciona TLS [14], es necesario comprender ciertos conceptos importantes y que forman parte del funcionamiento interno de SSL/TLS.

La capa donde opera el protocolo TLS se encuentra entre la capa de aplicación (donde operan los protocolos como HTTP, SMTP, etc) y entre la capa de transporte. TLS se suele implementar sobre el protocolo de transporte TCP, aunque actualmente también se puede implementar sobre UDP.

Para operar, el protocolo TLS encripta la información para protegerla al emitirla por la red. Para encriptar esta información, el protocolo TLS se divide en dos fases diferentes, por un lado se utiliza el protocolo de de mutuo acuerdo (TLS Handshake Protocol) que negocia los parámetros de seguridad mediante los cuales se van a comunicar ambos extremos, y por otro lado el protocolo de registro (TLS Record Protocol) que especifica la forma de encapsular los datos a emitir (incluyendo los parámetros de configuración acordados).

La capa donde actúa el protocolo handshake se encuentra ubicada en medio de la capa donde opera el protocolo de registro y la capa aplicación. La capa donde opera el protocolo de registro está ubicada sobre la capa de transporte y acondiciona los mensajes que recibe del handshake y los envía.

2.3.1 Protocolo Handshake TLS

Este protocolo permite que emisor y receptor negocien la versión del protocolo TLS a utilizar, elijan el algoritmo de cifrado y se autentiquen mutuamente garantizando la integridad de ambos. Una vez negociados ambos conceptos, se procede a establecer la conexión.

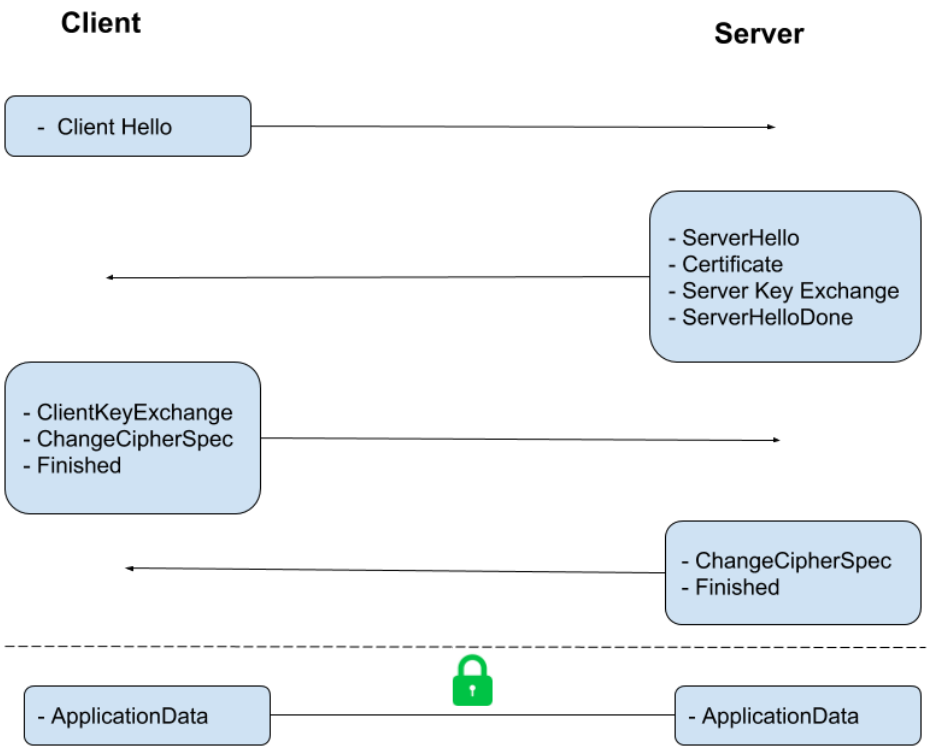


Figura 5: Intercambio de mensajes en el protocolo Handshake TLS 1.2

Para poder iniciar una sesión de conexión, como se puede ver en la figura 5, el protocolo handshake o de negociación inicia una fase de negociación siguiendo el siguiente orden:

- El cliente envía un mensaje **ClientHello** especificando tanto la información de cifrado como la versión del protocolo TLS/SSL, la lista de algoritmos de cifrado y los métodos de compresión que soporta. En el caso en el que se intente restablecer a una sesión anterior (se verá más adelante), se puede enviar el identificador de sesión, de esta manera, el proceso de negociación es más corto puesto que el proceso de negociación ya se ha hecho con anterioridad.
- El servidor responde al cliente con un mensaje **ServerHello** indicando el cifrado, método de compresión y protocolo TLS/SSL elegidos y una cadena de 32 bytes aleatorios. El protocolo TLS/SSL escogido debe ser el más actual que soporten ambos (tanto cliente como servidor).
- El servidor ofrece su mensaje de certificado para ser verificado por el cliente. De esta manera el cliente puede verificar la integridad del servidor, es decir, que es quien realmente dice ser.
- El servidor envía su mensaje **ServerKeyExchange**, el cual incluye la clave pública y la clave del certificado.
- El servidor solicita mediante el mensaje **Certificate Request** el certificado del cliente, si lo tiene, para que la conexión pueda ser mutuamente autenticada.

- El servidor envía un mensaje **ServerHelloDone**, dando por concluida la fase de negociación asimétrica.
- El cliente responde con un mensaje con su certificado.
- El cliente una vez ha comprobado y validado el certificado responde con un mensaje **ClientKeyExchange** que contiene la **PreMasterSecret** cifrada con la clave pública del servidor.
- El cliente y servidor usan las cadenas de números aleatorios y la **PreMasterSecret** para generar la clave de sesión secreta con la que ambos cifrarán y descifrarán la información enviada.

Una vez ha terminado la negociación, el cliente envía al servidor un registro **ChangeCipherSpec** indicando que todo los mensajes enviados a partir de ahora estarán cifrados con el cifrado acordado. El cliente envía un mensaje cifrado Finished, que da por finalizada la fase de negociación asimétrica. Este mensaje incluye un hash y un MAC [15] (código de autenticación del mensaje) de todos los mensajes handshake anteriores, para garantizar la integridad de la comunicación.

Tras esto, el servidor envía un registro **ChangeCipherSpec** indicando que todo lo que éste envíe a partir de ahora estará cifrado. El servidor envía el mensaje cifrado Finished para y el cliente lo descifra y verifica.

Con el protocolo de negociación o handshake terminado, se consigue crear un canal entre ambos bajo el protocolo TLS/SSL escogido, garantizando que todos los datos que viajen por este canal irán cifrados.

Lo bueno que tiene este protocolo de negociación y una de las razones por las que TLS es tan extendido y fiable, es porque utiliza la criptografía asimétrica de clave pública que permite a ambos extremos de la comunicación poder negociar una clave secreta compartida sin tener que hacerlo a través de un canal encriptado ni de conocerse previamente entre ambos. Una vez se acuerda esta clave secreta compartida se utiliza la criptografía simétrica para cifrar los datos a emitir, que es más rápida y requiere menos recursos hardware.

Sin embargo, la desventaja de esta negociación es que son necesarios varios mensajes en cada sentido de la comunicación y esto agrega mayor latencia a las conexiones TLS. Aunque uno de los objetivos principales de TLS 1.3 es reducir la latencia en las sesiones de conexión segura.

2.3.2 Protocolo de registro TLS

En la capa de registro se crean y se intercambian registros, los cuales encapsulan los datos a enviar.
Los datos se reciben de la capa de aplicación, se dividen en bloques, se les establece un código MAC, se cifran con el cifrado negociado y se distribuyen a la capa transporte para enviarlos. Gracias a este código MAC, el receptor puede garantizar la integridad y autenticidad del mensaje calculando y verificando este valor.

En el receptor, el procedimiento es el mismo pero a la inversa, es decir, recibe los datos encriptados, los desencripta, verifica el código MAC y los envía a la capa de aplicación. Lo bueno de este procedimiento es que todo se maneja en la capa TLS y es prácticamente transparente para las aplicaciones.

El protocolo TLS Record también se encarga de identificar los diferentes tipos de mensajes (protocolo de enlace, alerta o datos) y comprobar la integridad de los mensajes.

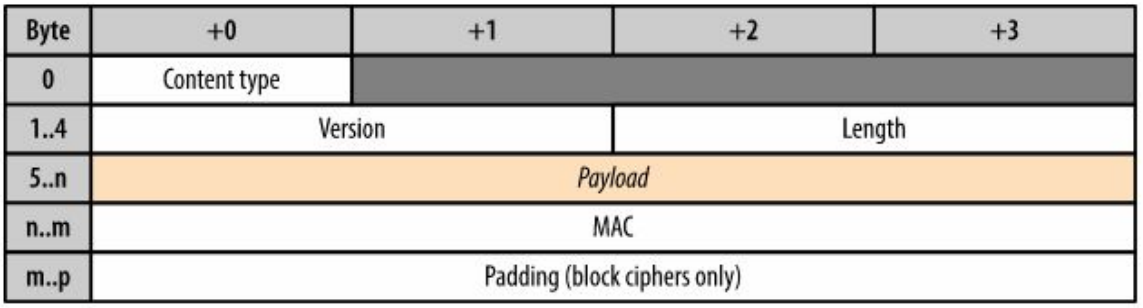


Figura 6: Registro genérico del protocolo TLS Record

Como se observa en la figura anterior, el registro genérico define 4 valores de content_type posibles.

- Content_type = 22 identifica un registro de tipo Handshake usado para el establecimiento de conexión.
- Content_type = 20 es de tipo ChangeCipherSpec son los enviados por el cliente y servidor para indicar que todo los mensajes siguientes irán cifrados.
- Content_type = 21 es de tipo Alert que sirve para el manejo de errores, avisos, etc que impiden el establecimiento o envío normal en la conexión. Ejemplos de ello pueden ser MAC incorrectos, errores de negociación, etc.
- Content_type = 23 es de tipo Application y contiene datos recibidos de la capa de aplicación, serán cifrados y enviados.

El campo Version identifica la versión utilizada de protocolo TLS/SSL.

El campo Length indica la longitud del mensaje, el MAC y relleno. No debe exceder de 16 Kb.

El campo Payload contiene el mensaje enviado por el protocolo correspondiente.

El campo MAC indica el código de autenticación calculado con el mensaje del protocolo, se utiliza para garantizar la integridad del mensaje.

El campo Padding contiene los datos cifrados siempre y cuando se utilice un cifrado de bloques.

2.4 Ataques y vulnerabilidades

2.4.1 POODLE

En Octubre de 2014, se publicó el ataque Padding Oracle On Downgraded Legacy Encryption (POODLE)[16] que ataque afectó a la versión SSL 3.0 aprovechando las vulnerabilidades CVE-2014-8730[17] y CVE-2014-3566[18]. Este ataque aprovechaba dos factores, por un lado, que todavía muchos servidores seguían permitiendo SSL 3.0 para la interoperabilidad, y por otro lado, la vulnerabilidad conocida relacionada con el padding o relleno en los bloques al utilizar un cifrado de bloques CBC.

Este ataque requiere que exista previamente un ataque Man-in-the-Middle entre el cliente y el servidor. En este caso, al iniciarse la fase de Handshake, el cliente le envía al servidor la lista de versiones TLS/SSL soportadas, no obstante, el atacante captura esa información y se hace pasar por el servidor haciendo creer al cliente que la versión más estable o más sólida que soporta es SSL 3.0, haciendo que la conexión entre ambos se establezca por SSL 3.0.

Ahora que la conexión entre cliente y servidor es vulnerable, el atacante realiza el ataque POODLE. SSLv3 tiene dos modos de funcionamiento, utilizando un cifrado de flujo con RC4 (vulnerable), o un cifrado por bloques en modo CBC. Este ataque consiste en explotar el modo de encadenamiento de cifrado de bloques (Cipher Block Chaining) que divide los datos a cifrar en bloques de longitud fija, se cifra cada uno y se combina con el texto sin cifrar del bloque anterior (utilizando la operación XOR). Si el tamaño del último bloque no es múltiplo de la longitud total, se añade un padding o relleno. El código MAC no cubre al relleno, por lo que el servidor no puede saber si el padding ha sido modificado.

Por ello, si el atacante quiere descubrir una cookie de sesión, lo primero que tiene que hacer es identificar el primer byte de la cookie. Para ello, el atacante puede

hacer peticiones desde el navegador al servidor y modificar los bytes forzando que el padding ocupe un bloque entero de 8 bytes, y que el primer carácter de la cookie caiga en la posición final de un bloque, al que llamaremos el bloque **X**. El atacante copia este bloque **X** en el último bloque, que contiene el relleno encriptado. El atacante puede ir modificando bytes e iterando la petición; si en una de estas iteraciones el servidor acepta la petición, el atacante habrá obtenido una información muy valiosa: el primer byte de la cookie.

Como la función XOR es reversible, podemos obtener el texto en claro del primer byte de la cookie, de esta manera podemos reproducir el ataque con el resto de bytes que queramos adivinar hasta obtener todos los bytes de la cookie.

La probabilidad de que el servidor considere la petición como correcta y la acepte es de 1 entre 256 solicitudes SSL 3.0 del navegador de la víctima. Siempre que un intento termine en fracaso, podemos volverlo a intentar de nuevo sabiendo que lo que cambia es la clave de cifrado.

El ataque Poodle, pese a vulnerar el protocolo SSLv3, no es tan grave ni tan potente como los ataques HeartBleed, puesto que, se necesita un Man-in-the-Middle, que el servidor permita SSLv3 y que use el cifrado de bloques CBC. Además, es algo lento comparado con el resto de ataques.

2.4.2 Heatbleed

Heartbleed es una vulnerabilidad crítica en todas las implementaciones de OpenSSL liberadas entre marzo de 2012 y abril de 2014, posteriormente se corrigió y se liberó la versión 1.0.1g. Es conocida como CVE-2014-0160[19].

Este fallo de implementación en OpenSSL permitía que un atacante pudiese leer hasta 64 Kb de memoria por ataque en cualquiera de ambos extremos.

Heartbeat es una extensión para los protocolos TLS y DTLS especificada en el estándar RFC 6520[20] que incorpora **mensajes de heartbeat** para mantener activa la conexión aunque los interlocutores no se estén continuamente transmitiendo datos. Esta extensión actúa en la parte superior de la capa de registro y permite al emisor enviar un mensaje **HeartbeatRequest** y al receptor responder inmediatamente con un mensaje **HeartbeatResponse**.

Uno de los extremos envía un mensaje **HeartbeatRequest** con los datos o payload y tamaño de esos datos más el relleno. El otro extremo, reserva un buffer de datos con el tamaño recibido y copia el payload recibido, inmediatamente envía un mensaje **HeartbeatResponse** con el contenido de las posiciones del buffer

reservado. No obstante, esto puede ser fácilmente atacado si, por ejemplo, se envía un **HeartbeatRequest** con 5 bytes de contenido y tamaño de 5000 bytes, haciendo que el servidor guarde en el buffer los 5 bytes de contenido, pero a la hora de responder, envía un mensaje **HeatbeatResponse** con los 5 bytes de nuestro mensaje más los 4995 bytes restantes que contienen otros datos de la memoria.

Por ello, si hacemos ataques continuamente extrayendo 64 KB de datos cada vez, conseguiremos encontrar información importante de la memoria. Así mismo, ha habido exploits de prueba que han conseguido filtrar la clave privada del servidor, pudiendo con ello, descifrar todo el tráfico del servidor.

2.4.3 FREAK

El ataque FREAK (Factoring Attack on RSA-EXPORT Keys) permite, mediante un ataque Man-in-the-Middle, forzar a los clientes a usar métodos de cifrado débiles y antiguos. Se aprovechaba de las vulnerabilidades CVE-2015-0204[21], CVE-2015-2235[22], CVE-2015-1637[23].

Aunque este ataque se descubrió en el año 2015, la vulnerabilidad surge en la década de los 90, cuando Estados Unidos fue incorporando el cifrado a las comunicaciones consideradas críticas. Esos cifrados eran tan importantes que decidieron exportarlos fuera de Estados Unidos pero mediante una versión limitada que permitía utilizar los mismos métodos de cifrado RSA pero con claves de longitud inferior o igual a 512 bits. Estos conjuntos de cifrados destinados a ser exportados fuera de EEUU se les conoció como EXPORT.

Pronto empezaron a cambiar las restricciones y estos cifrados débiles fueron sustituidos por mejores algoritmos y claves de mayor longitud, pero nunca terminaron por desaparecer.

Este ataque se aprovecha de que muchas implementaciones de los cifrados TLS/SSL seguían soportando versiones antiguas como las EXPORT, aunque prácticamente nadie las utilizaba.

Al igual que en el ataque POODLE, es preciso de un ataque Man-in-the-Middle que fuerce a que cliente y servidor se comunicasen utilizando versiones EXPORT, cifrando los datos con claves RSA de 512 bits, que se pueden romper fácilmente en pocas horas y con hardware convencional.

Una exploración en más de 14 millones de sitios web que soportan los protocolos SSL/TLS resultó que más del 36% eran vulnerables a los ataques de descifrado que

admiten el uso de los certificados en modo RSA EXPORT (por ejemplo, TLS_RSA_EXPORT_WITH_DES40_CBC_SHA).

Al igual que el ataque POODLE, FREAK es poco probable que ocurra, pues se tienen que dar diversas condiciones como por ejemplo, que exista un atacante Man-in-the-Middle, que los navegadores permitan el uso del certificado en modo RSA EXPORT y que las implementaciones SSL/TLS utilicen el algoritmo de cifrado RSA.

2.4.4 Logjam

En 2016, un grupo de investigadores de seguridad y científicos informáticos descubrieron una vulnerabilidad en la forma en que se implementa un intercambio de claves Diffie-Hellman en la web. Es conocida como Logjam[6] y recogida en CVE-2015-4000[24].

El algoritmo Diffie-Hellman se utiliza para establecer la clave secreta entre ambos interlocutores. Así mismo, protocolos como SSH o TLS usan Diffie-Hellman para establecer claves de sesión y poder encriptar los datos para que viajen de forma segura.

Los ataques que aprovechan la vulnerabilidad de Logjam se basan en un ataque Man-in-the-Middle que puede degradar las conexiones vulnerables de TLS a una criptografía que usa grupos de claves de 512 bits, un nivel de cifrado bastante sencillo de romper. No obstante, posteriormente se demostró que también afecta a los grupos de claves Diffie-Hellman de 1024 bits.

Este ataque es muy similar al ataque FREAK, a diferencia de que Logjam aprovecha un defecto en el protocolo TLS en vez de un problema de implementación y ataca a un intercambio de clave Diffie-Hellman en vez de RSA.

Tras este ataque, optaron por dejar de utilizar Diffie-Hellman (DH) e incorporar a los servidores el algoritmo Diffie-Hellman de Curva elíptica efímero (ECDHE), considerado más seguro por su uso de curvas elípticas. Así mismo, usar claves de 2018 bits, en vez de 1024 o 512.

2.4.5 CRIME

CRIME[25] (Compression Ratio Info-leak Made Easy) es una vulnerabilidad recogida en CVE-2012-4929[26] que se encontró en la compresión de datos en modo

DEFLATE de TLS/SSL. El ataque se lleva a cabo mediante un ataque Man-in-the-Middle y le permite poder revelar la información cifrada que se transmite por el canal. Esta vulnerabilidad se solía aprovechar recuperar las cookies de sesión y suplantar a la víctima.

DEFLATE, como la mayoría de algoritmos de compresión emplean la técnica de reemplazar bytes o caracteres repetidos con un puntero a la primera instancia de ese byte, por lo que cuantos más caracteres repetidos haya, mejor es la relación de compresión.

Se explica este ataque mediante el siguiente ejemplo:

El atacante conoce una parte de la cookie **"Cookie: secret ="**. Para intentar adivinar el valor de **"secret"**, intercepta la solicitud HTTP que realiza el cliente al servidor, ataca el navegador de la víctima con código Javascript (por ejemplo) y le añade a la petición **"Cookie: secret=0"**. El atacante analiza la longitud de la información comprimida y lo intenta de nuevo, haciendo que el cliente envíe ahora **"Cookie: secret=1"** en el cuerpo de la solicitud, y así hasta que al analizar la información comprimida, observe que la compresión ha sido mejor. De esta manera, el atacante sabe que el valor que él ha enviado junto a la palabra secret se encuentra en la cookie. Tras ello, repite el proceso para obtener byte por byte, el valor de "secret" completo.

Para llevar a cabo un ataque CRIME, se requiere que tanto cliente como servidor utilicen compresión TLS/SSL (que se negocia en la fase de Handshake), que el atacante pueda ejecutar un agente Javascript en el navegador del cliente y que se produzca un ataque Man-in-the-Middle y éste tenga un sniffer para interceptar los datos transmitidos por cliente y servidor.

2.4.6 SLOTH

SLOTH[27] (security losses from obsolete and truncated transcript hashes) es una vulnerabilidad del protocolo TLS 1.2 recogida en CVE-2015-7575[28] y basada en las vulnerabilidades anteriormente comentadas como POODLE, FREAK y Logjam, las cuales ya han demostrado previamente la posibilidad de degradar a suites de cifrado y versiones de protocolo SSL/TLS más débiles. No obstante, a diferencia de estos ataques, los ataques SLOTH pueden obligar a ambas partes de la comunicación a utilizar un algoritmo hash más débil e inseguro como MD5 y SHA1, los cuales no necesitan máquinas computacionalmente costosas ni emplear demasiado tiempo para ser rotos.

En protocolos anteriores a TLS 1.2 no existía la opción de negociación del protocolo de firma y hash, si no que se utilizaba una concatenación de MD5 y SHA. Con la llegada de TLS 1.2 se añadió el campo **SignatureAndHashAlgorithm** en el mensaje **ServerKeyExchange**, que permite indicar qué algoritmos de firma y hash debe usar el cliente.

El cliente, envía el mensaje **ClientHello** indicando qué algoritmos de firma y de hash puede utilizar, no obstante, un atacante en medio de la comunicación puede interceptar este paquete y responder con un mensaje que sugiere algoritmos diferentes como RSA para la firma y MD5 para el hash. El cliente aceptará, permitiendo, de esta manera, al atacante suplantar al servidor y descifrar el tráfico una vez degradado el canal TLS.

De igual manera pasaría si se quisiese suplantar al cliente, en vez de la del servidor. El atacante interceptaría el mensaje **ClientHello** del cliente, lo modificaría indicando que el único algoritmo que soporta es RSA-MD5 y el servidor aceptará, permitiendo la degradación a este algoritmo débil y vulnerable.

2.4.7 ROBOT

En diciembre de 2017 se publicó un nuevo ataque que aprovecha vulnerabilidad en algunas implementaciones de TLS/SSL, conocida como ROBOT[29] (Return Of Bleichenbacher's Oracle Threat) y recogida en CVE-2017-17382[30]. Este nuevo ataque es una variante del ataque Bleichenbacher sobre el algoritmo RSA, que se descubrió en 1998.

En la fase de Handshake, el cliente cifra la **PreMasterSecret** con la clave pública RSA del servidor y se la envía, por lo que sólo puede ser descifrada por el servidor, que utiliza su clave privada para descifrarlo.

El tamaño de la **PreMasterSecret** suele ser más pequeño que el tamaño de la clave pública del servidor, no obstante, es una mala práctica a nivel de seguridad cifrar un valor más pequeño que el tamaño de la clave RSA, por lo que se le añaden bits adicionales como relleno al mensaje. Este relleno tiene que cumplir el formato PKCS#1 v1.5[23]. Quedando el mensaje original cifrado como se observa en la siguiente figura:

Figura 7: Mensaje cifrado con clave RSA y relleno PKCS#1 v1.5

Donde los valores en rojo son fijos, los valores en verde son aleatorios distintos de 0 y los valores en azul corresponden a la **PreMasterSecret**.

Por ello, el servidor recibe el **ClientKeyExchange** y lo descifra. Si coincide con que los dos primeros bytes son 0x00 0x02 o P y M son distintos de cero y los valores entre ambos son 0, el servidor continúa con la conexión; si no, envía una alerta y cierra la conexión. Esto implica que el servidor está proporcionando respuestas de "sí" y "no", que a medida que se vaya repitiendo el ataque, puede revelar información sobre el contenido del mensaje. Esto se conoce como el oráculo de Bleichenbacher.

Tras conocer el oráculo, vamos a comentar cómo un atacante puede aprovecharlo. El algoritmo RSA es un algoritmo homomórfico, es decir, podemos aplicar una serie de transformaciones a un mensaje cifrado quedando el texto en claro del nuevo mensaje relacionado de forma concreta con el texto en claro del mensaje original. El ataque consiste interceptar el paquete cifrado por el cliente y aplicarle transformaciones para ir conociendo la relación entre el mensaje aceptado y el original. De esta manera, el atacante puede ir acotando los posibles valores del PreMasterSecret y mediante fuerza bruta obtener su valor exacto.

Este fallo permite a los atacantes a poder descifrar mensajes cifrados con RSA sin tener que utilizar la clave privada del servidor, lo que permite romper la confidencialidad de SSL y poder descifrar todos los mensajes intercambiados entre cliente y servidor.

El ataque de Bleichenbacher se produjo en 1998, en aquella época el protocolo SSL estable era SSLv3. En 1999, se introdujo TLS 1.0, el cual presentaba contramedidas para evitar este ataque, proponiendo que el servidor generase una **PreMasterSecret** aleatoria cada vez que detectase un mensaje **ClientKeyExchange** mal formado. Así, el atacante ya no puede distinguir los valores incorrectos, pues el servidor se comporta como si todos los mensajes fueran válidos.

No obstante, se desarrolló una nueva versión del oráculo, el cual esta vez se basaba en el tiempo que tardaba el servidor en responder. Puesto que, como al generar una PreMasterSecret aleatoria tardaba más tiempo que en responder con la correcta, el atacante podía adivinar cuando el formato del mensaje era correcto y cuando no.

En la versión de TLS 1.1 se tuvo en cuenta este nuevo ataque y se adaptaron las contramedidas, no obstante, en TLS 1.2 se propusieron dos nuevos algoritmos que permitían evitar el ataque de Bleichenbacher, que básicamente se diferencian en cómo tratan los mensajes **ClientHello** incorrectos. El primero de ellos establece que

aunque los servidores reciban **ClientHello** incorrectos, los arreglen y calculen el mensaje finalizado con él, mientras que el segundo supone que un error en la PreMasterSecret será considerado como un error. El estándar TLS 1.2, recomienda utilizar el primero de ellos, pues arregla el oráculo de los tiempos comentado anteriormente.

Los estándares TLS indican que el protocolo OAEP (Optimal Asymmetric Encryption Padding) proporciona mayor seguridad, pero decidieron mantener el estándar PKCS#1 v1.5[31] para mantener la compatibilidad. Por ello, esta versión de implementación de TLS sigue siendo vulnerable. Muchos investigadores critican que en el estándar TLS 1.2 no arregla la vulnerabilidad, sino que simplemente aplica nuevas contramedidas.

Sin embargo, el problema actual, según investigadores, recae en que el estándar TLS es muy complejo y muchos vendedores de equipos de servidores no implementan correctamente la Sección 7.4.7.1 del estándar TLS (RFC 5246[11]), que define las medidas de ataque originales de Bleichenbacher.

El 12 de diciembre de 2017, se publicó un artículo detallando el ataque ROBOT y señalando que 27 de los 100 sitios web más visitados del mundo según el ranking Alexa son vulnerables a ROBOT, entre ellos se incluye PayPal y Facebook. También afecta a productos de seguridad de empresas como F5, Cisco, Citrix, Radware, etc. Esto es debido a que no han aplicado las nuevas contramedidas y siguen utilizando TLS con RSA.

2.4.8 Vulnerabilidades actuales que no se han corregido

Como se ha comentado anteriormente, los diseñadores del protocolo TLS, con la versión TLS 1.2 reaccionan al ataque de Bleichenbacher adoptando soluciones bastante complejas, las cuales no se implementan en una gran cantidad de equipos. Por ello, lo que se lleva a cabo para la versión TLS 1.3, es la desaprobación del cifrado RSA, lo que no significa que sea suficiente, porque el ataque se puede seguir llevando a cabo mediante las otras versiones del protocolo, pues habrá clientes que no soporten TLS 1.3 o ataques Man-in-the-Middle que degraden la versión del protocolo a utilizar.

Por ello, tras estudiar algunos de los mayores ataques que ha sufrido el protocolo TLS/SSL, observamos que la mayoría de ataques se basa atacar o aprovechar las debilidades del cifrado RSA, en consecuencia, lo correcto sería deshabilitar completamente el soporte para intercambios de claves cifradas con RSA. En el último borrador publicado por la IETF del protocolo 1.3, se elimina el soporte para los intercambios de clave RSA y Diffie-Hellman, indicando que todos los

intercambios de clave basados en clave pública proveerán secreto hacia adelante. Por lo que, TLS 1.3 sólo soportará (EC) DHE (Diffie-Hellman sobre campos finitos o sobre curvas elípticas), PSK y PSK con (EC) DHE.

Además, si el cliente no provee una extensión suficiente de **PreMasterSecret** como, por ejemplo, que incluya solo grupos DHE o ECDHE inaceptables o no admitidos por el servidor, el servidor manda un mensaje **HelloRetryRequest** y el cliente necesita reiniciar el proceso con una **PreMasterSecret** adecuada. En el caso de que no haya parámetros criptográficos comunes, el servidor debe abortar el handshake y emitir una alerta apropiada.

2.5 Malwares que aprovechan el protocolo TLS para asegurar sus comunicaciones

El cifrado se ha utilizado para proteger los datos confidenciales que viajan por la red, sin embargo, en los últimos años, el uso del cifrado se ha incrementado de tal manera que la mayoría del tráfico viaja cifrado hoy en día. No obstante, al igual que es efectivo para la protección de los datos, también permite que datos maliciosos viajen ocultos.

En 2017, un estudio de Zscaler[32], empresa de Cloud Security, detectó que el 60% de las amenazas detectadas en su nube venían cifradas con SSL/TLS, bloqueando alrededor de 800.000 actividades maliciosas cifradas cada día, incluyendo troyanos, con el ransomware siendo la segunda mayor amenaza con un 25% de ataques.

Los principales ataques han sido:

- Exploit kits
- Malware
- Adware
- Devolución de llamada de malware

2.5.1 De qué manera lo aprovechan.

Los delincuentes cibernéticos se aprovechan de los proveedores que emiten certificados SSL/TLS gratuitos, ya que permiten utilizar HTTPS en dominios maliciosos, eludiendo comprobaciones de identidad e integridad en los navegadores.
Estos certificados ofrecen confianza a los usuarios, sin embargo, no son conscientes de que pueden ser víctimas de malwares que pueden robarles cualquier tipo de información.

Otro de los ataques importantes se basa en publicar páginas phishing maliciosas en sitios con certificados legítimos que utilizan SSL/TLS. De esta manera, tras ver la palabra “segura” y el candado verde en el navegador, el usuario cree entrar a un sitio válido, sin embargo, esto no valida la legitimidad del sitio web ni si realmente es el sitio que dice ser, tan sólo indica que el certificado es válido y que las comunicaciones van encriptadas.

Según indicó Deepen Desai[32], director de investigación de seguridad de Zscaler, Microsoft, LinkedIn y Adobe se encuentran entre las marcas más comúnmente falsificadas. Otros sitios que están siendo abusados por equipos de phishing incluyen Amazon Seller, Google Drive, Outlook y DocuSign.

Los enfoques típicos para enfrentar este problema se basan en descifrar el tráfico aplicando la técnica Man-in-the-Middle y mecanismos clásicos de IDS / IPS sobre este tráfico. Sin embargo, este enfoque tiene varias desventajas, es difícil y costoso de implementar y también implica romper la confidencialidad.

Capítulo 3. Mejoras de seguridad y velocidad introducidas por TLS 1.3

TLS 1.3 [12] es la cuarta versión de la familia TLS. Con ella trae una serie de mejoras que corrigen las inseguridades presentes en TLS 1.2 y permiten aumentar la velocidad en el inicio de la comunicación entre ambas partes al reducir el número de conexiones en la fase de Handshake.

En cuanto a seguridad, como ya se ha comentado previamente, TLS 1.3 elimina el debate sobre qué algoritmo de encriptación usar y la consecuencia de permitir un ataque downgrade, al eliminar toda la posibilidad de utilizar cifrados y algoritmos inseguros. TLS 1.3 elimina los sistemas de cifrado obsoletos y los reemplaza con algoritmos actualmente seguros.

Por otro lado, TLS 1.3 ofrece una versión mejorada del protocolo de enlace en el que se lleva a cabo la fase de Handshake antes de iniciar una comunicación.

Las versiones anteriores a TLS 1.3 llevaban a cabo una negociación conocida como “2-RTT” en la cual se necesitaban dos viajes de ida y vuelta de datos entre cliente y servidor para negociar los términos mediante los que se llevaría a cabo la transferencia de datos. Esto supone la necesidad de emplear entre 0.25 y 0.5 segundos en ejecutar el protocolo de enlace. Lo cual, para una conexión rutinaria para navegar por Internet o para realizar consultas cotidianas quizá no sea demasiado significante esperar medio segundo para poder establecer la conexión, pero en áreas como, por ejemplo, el comercio de la inversión en acciones, la velocidad de conexión adquiere mayor importancia, pues medio segundo puede conllevar un impacto masivo.

Puesto que no es posible reducir la latencia de cada uno de estos mensajes de ida y vuelta, con TLS 1.3 redujeron el número de viajes necesarios para establecer la conexión, consiguiendo un Handshake bastante más rápido. De esta manera, se eliminan pasos innecesarios y reducen a tan solo un viaje de ida y vuelta (1-RTT) el proceso de negociación. En la figura 8 se muestra gráficamente el intercambio de mensajes llevado a cabo en TLS 1.3.

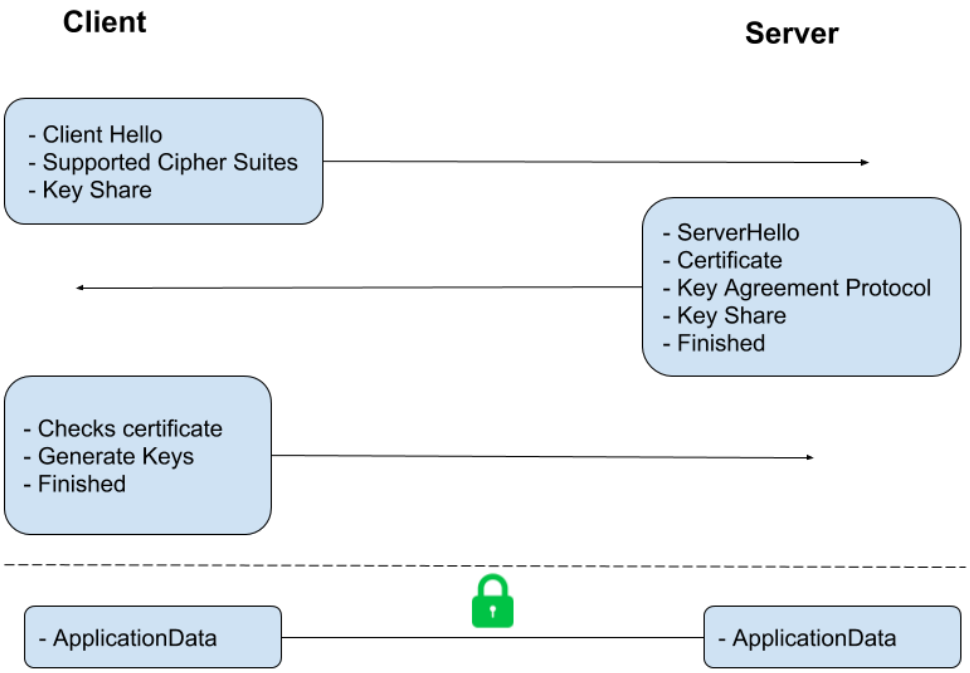


Figura 8: Intercambio de mensajes en el protocolo Handshake TLS 1.3

Paso 1: Es similar al Handshake de TLS 1.2. Comienza enviando el cliente al servidor el mensaje “Client Hello” que contiene la lista de cipher suites soportadas y su keyShare.

Paso 2: En respuesta al mensaje "ClientHello", el servidor responde con el mensaje "ServerHello" que contiene el cipher suite elegido, el keyShare del servidor, su certificado y un mensaje “Server Finished” indicando que por su parte ha terminado el proceso de handshake.

El mensaje "Server finished" se envía en el segundo mensaje de vuelta en TLS 1.2. De esta manera se puede ahorrar un viaje de ida y vuelta.

Paso 3: El cliente verifica el certificado del servidor, genera claves ya que tiene el recurso clave compartido del servidor y envía el mensaje "Cliente finalizado". A partir de aquí, comienza el cifrado de los datos.

3.1 TLS 1.3 + 0-RTT

TLS 1.3 proporciona ventajas para conexiones iniciadas o reanudadas. Como alrededor del 40% de las conexiones HTTPS son reanudadas, para ganar velocidad en el establecimiento de la comunicación, se utiliza el protocolo 0-RTT, una

tecnología que permite que solicitudes HTTPS sean tan rápidas como las solicitudes HTTP, las cuales no van cifradas.

Esta tecnología permite reanudar una comunicación con los parámetros ya negociados previamente en la conexión anterior, evitando volver a tener que iniciar el proceso de handshake desde el principio.

3.1.1 0-RTT Data

TLS 1.3 introduce el concepto 0-RTT Data, que permite que cuando cliente y servidor ya comparten una PSK (PreSharedKey) porque ya hayan iniciado una comunicación anteriormente, puedan volver a reanudar la comunicación pudiendo enviar datos en el primer viaje del proceso de handshake (lo que se conoce como "early_data"). El cliente, como ya tiene la clave compartida PSK, la puede utilizar para autenticar al servidor y para encriptar esos datos.

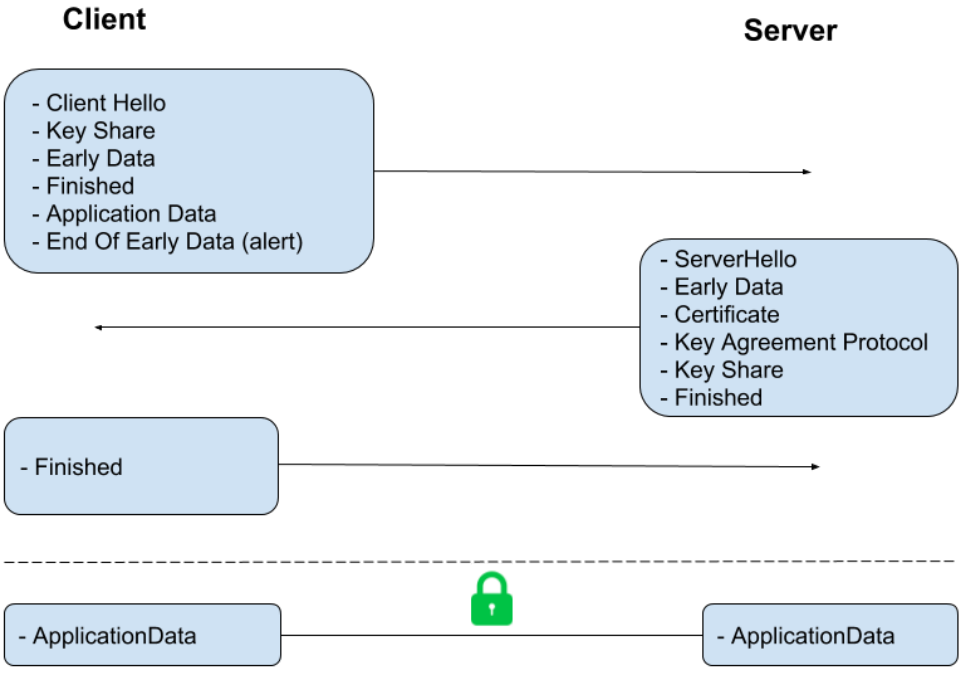


Figura 9: Intercambio de mensajes en el protocolo Handshake TLS 1.3 0-RTT Data

3.1.2 Pérdida en propiedades de seguridad

Las propiedades de seguridad para las solicitudes 0-RTT son más débiles que para el resto de datos enviados en una comunicación no reanudada. Especialmente porque:

- No hace uso del Perfect Forward Secrecy, sino que la información se encripta utilizando la PSK ya conocida, o que supone que si la clave está

comprometida, un atacante puede descifrar esos datos. Podemos evitarlo si se rotan las claves de sesión con frecuencia.

- No hay garantías de no repetición entre conexiones. El Random value del servidor proporciona protección contra la repetición en 0-RTT Data, pero estos datos no dependen del Server Hello sino del cliente, por lo que las garantías son más débiles. Esto es especialmente relevante si los datos se autentican con autenticación de cliente TLS o dentro de la aplicación.

Las solicitudes enviadas como parte de una reanudación 0-RTT son vulnerables a ataques de repetición. Si un atacante logra obtener acceso a la conexión cifrada entre cliente y servidor y consigue los datos de la primera solicitud, puede hacer una copia de los mismos y enviar la solicitud al servidor suplantando la identidad del cliente. El servidor verá solicitudes repetidas pero como no tiene forma de saber el origen de procedencia de los datos, el atacante realizará el ataque con éxito.

3.1.3 Mecanismos para evitar ataques por repetición

Estos ataques no son demasiado fáciles para llevarlos a la práctica pero, además de ello, en el capítulo 8 del borrador 28 de TLS 1.3 [12] figuran mecanismos para evitar estos ataques de repetición.

3.1.3.1 Single-Use Tickets

La forma más sencilla de evitar ataques de repetición al servidor es que éste tan sólo permita que el mismo ticket de sesión se use una vez. Por ejemplo, el servidor puede mantener una base de datos de todos los tickets válidos pendientes y eliminar los ya usados.

Si los tickets no son autocontenidos, sino que son claves de la base de datos, y los PSK correspondientes se eliminan con el uso, las conexiones establecidas mediante PSK disfrutan de confidencialidad y Forward Secrecy. Esto mejora la seguridad de todos los datos 0-RTT.

Puede conllevar problemas o ineficacia cuando este mecanismo requiera compartir esta base de datos con diferentes nodos de servidor en sistemas distribuidos, dado que puede ser complicado lograr altas tasas de conexiones PSK 0-RTT exitosas.

3.1.3.2 ClientHello Recording

Otro mecanismo para evitar estos ataques se basa en registrar un valor único derivado de ClientHello (generalmente el Random Value o el PSK) y rechazar

duplicados. Si se registrasen todos mensajes ClientHello, el registro crecería sin límite, por lo que en el borrador se propone el registro de mensajes ClientHello dentro ventanas de tiempo determinadas y hacer uso del parámetro "obfuscated_ticket_age" para garantizar que los tickets no se reutilicen fuera de esa ventana.

Para implementar esto, el servidor al recibir un mensaje ClientHello verifica el PSK, calcula el tiempo de llegada mediante el parámetro “expected_arrival_time” y comprueba si ese tiempo está dentro de la ventana de tiempo. Si no está dentro de la ventana rechaza el 0-RTT y efectúa el handshake de 1-RTT de TLS 1.3.

En caso de que el servidor verifique que se ha registrado un ClientHello coincidente, cancela el handshake y emite una alerta. Si no se encuentra ClientHello coincidente, entonces acepta 0-RTT y almacena el ClientHello siempre que el “expected_arrival_time” esté dentro de la ventana de tiempo.

El servidor debe guardar las claves solo de las secciones validadas de ClientHello. Si el ClientHello contiene múltiples PSK, un atacante puede crear múltiples mensajes ClientHello con diferentes valores de PSK en el supuesto de que el servidor no la verifique. Esto puede causar efectos negativos como la contaminación de la caché del registro, aunque no se podrá acceder a los datos puesto que al ser claves diferentes no se podrán descifrar.

Este mecanismo no consiste en almacenar todos los tickets pendientes, por lo que podría ser más fácil de implementar en sistemas distribuidos con altas tasas de reanudación y 0-RTT. En muchos de estos sistemas, no es práctico tener un almacenamiento global coherente de todos los ClientHello recibidos. En este caso, la mejor protección contra los ataques de repetición se consigue al tener solamente una única zona de almacenamiento autorizada para un ticket determinado y rechazar el 0-RTT para ese ticket en cualquier otra zona.

Puede darse el caso de que el reloj del cliente se ejecute de manera más rápida que el del servidor y se reciba un ClientHello que esté fuera de la ventana en el futuro, en cuyo caso podría aceptarse para 1-RTT, lo que provocaría un nuevo intento del cliente que posteriormente sería aceptado para 0-RTT. Esto podría ser otra forma de ataque.

3.1.3.3 Medidas llevadas a cabo en la práctica

Aunque el protocolo de enlace permita ataques por repetición, las aplicaciones web deberían estar diseñadas para resistir a este tipo de ataques. Sin embargo, no

siempre es así. La empresa Cloudflare [33] protege las aplicaciones web utilizando un enfoque conservador para elegir qué tipo de solicitudes se responden mediante 0-RTT o mediante un proceso de handshake iniciado desde cero.

Como las solicitudes GET son idempotentes y no cambian el estado del servidor, solamente las solicitudes GET sin parámetros se responden mediante 0-RTT, el resto realizarían la fase de handshake 1-RTT propia de TLS 1.3.

Además, implementan un número máximo de solicitudes de 0-RTT, limitan el tiempo en el que pueden reanudarse y permiten identificar de manera única los intentos de reanudación de una conexión, respondiendo al cliente añadiendo un encabezado a la solicitud que permite identificarla. De esta manera, si el encabezado se repite, el origen sabrá que se ha producido un ataque de repetición.

El uso de 0-RTT no es obligatorio con TLS, por lo que si alguno de los interlocutores de la comunicación desconfía o encuentra cosas extrañas en la aplicación web, puede desactivarlo.

Capítulo 4. HTTP + TLS

4.1 Combinación de los protocolos HTTP y TLS para asegurar las comunicaciones.

Hypertext Transfer Protocol (HTTP)[34] es un protocolo para la transmisión de documentos hipermedia, como HTML, reside en la capa de aplicación. Define cómo se formatean y se transmiten los mensajes en la comunicación entre los navegadores y servidores web.

Es un protocolo denominado “sin estado” dado que cada comando se ejecuta de manera independiente, sin tener conocimiento de los comandos anteriores y no se guarda ningún dato entre peticiones. Por esta razón, es difícil la implementación de sitios web que reaccionen de forma inteligente a la entrada del usuario, lo que ha conllevado a abordar esta deficiencia en tecnologías como ActiveX, Java, JavaScript o mediante cookies.

HTTP sigue un modelo cliente-servidor, en el que un cliente establece una conexión, realizando una petición a un servidor y espera una respuesta del mismo. Aunque en la mayoría de casos se basa en una conexión del tipo TCP/IP, puede ser usado sobre cualquier capa de transporte segura o de confianza.

HTTP fue el protocolo más utilizado en Internet, no obstante, el incremento de uso de este protocolo para aplicaciones sensibles requirió adoptar una serie de medidas de seguridad, como la implementación de HTTP sobre SSL/TLS en vez de sobre TCP, dando paso al protocolo HTTPS [35].

Con HTTPS, la conexión es previamente negociada en vez de enviar los datos directamente por TCP. El cliente HTTP realiza una solicitud de conexión al servidor y entre ambos se realiza un proceso de handshake SSL/TLS previamente al establecimiento de sesión, como se ha desarrollado en capítulos anteriores. Tras haber acordado la clave de cifrado, se cifran todos los datos de HTTP, es decir, la página web, la URL, las cookies, parámetros enviados, etc. Los únicos datos que no se cifran y quedan al descubierto son datos del paquete TCP: el servidor y el puerto al que nos queremos conectar.

En la práctica, es muy común ejecutar HTTP + TLS sobre un puerto separado con la finalidad de identificar qué protocolo se está utilizando. Si HTTPS se está ejecutando sobre TCP/IP, el puerto predeterminado es el 443. Esto no impide que

se pueda realizar en otro puerto TCP. TLS tan sólo proporciona a HTTP una conexión donde la transmisión de datos es segura.

Desde el año 2014, Google presiona para que HTTPS se use en todos los sitios de la red, incluso recompensa con mejor posicionamiento en la web a los sitios web seguros. A partir de julio de 2018, catalogará y marcará los sitios que no han sido migrados a HTTPS como “sitios no seguros” en el navegador Chrome.

Según se especifica en el Security Google Blog [36] *“La nueva interfaz de Chrome ayudará a los usuarios a comprender que todos los sitios HTTP no son seguros, y continúan moviendo la web hacia una red segura HTTPS de forma predeterminada. HTTPS es más fácil y más barato que nunca, y desbloquea mejoras de rendimiento y nuevas funciones potentes que son demasiado sensibles para HTTP.”*

4.2 Certificados SSL/TLS

Un argumento que suele aparecer en contra del uso de TLS junto con el protocolo HTTP es el uso de los certificados SSL/TLS de servidor y sobre todo su precio. Si bien existen algunas Autoridades Certificadoras que ofrecen muy baratos e incluso gratuitos como Let's Encrypt, los cuales son bastante sencillos de instalar.

Merece la pena dedicar un apartado a este tipo de certificados puesto que es una entidad certificadora que ha entrado con fuerza en el mercado de las entidades emisoras de certificados SSL/TLS y que en tan sólo dos años ha superado los 70 millones de certificados activos [37].

4.2.1 Certificados de Autoridad Certificadora Let’s Encrypt

Let’s Encrypt [37] es una autoridad de certificación (CA) automatizada que se puso en marcha el 12 de abril de 2016 y que proporciona certificados X.509 gratuitos para cifrar las comunicaciones al nivel de transporte. Nace de un esfuerzo conjunto del grupo Internet Security Research Group [37] el cual está compuesto por miembros de empresas como Akamai, Mozilla, Cisco, CoreOS, etc, y fue creado para el beneficio de la comunidad.

Let’s Encrypt usa el protocolo ACME (Automatic Certificate Management Environment)[37] para verificar que el dominio realmente pertenece al servidor desde el que se está solicitando o renovando el certificado digital. Existen dos mecanismos

para validar que se tiene el control sobre el dominio, el primero y más empleado por los clientes utiliza el protocolo HTTP.

Consiste en utilizar un par de claves pública-privada y una serie de challenges que el servidor tiene que realizar. En el servidor es necesario instalar el cliente Certbot de Let's Encrypt, el cual se conecta con la CA y genera el par de claves. El proceso entre el cliente de Let's Encrypt y la CA, según la página oficial [37], es el siguiente:

1. La CA le pide al cliente instalado en el servidor que aloje un fichero en una dirección específica del dominio, y le pasa un token para que lo firme.
2. El cliente guarda ese archivo y lo firma, comunicando a la CA que está listo.
3. La CA accede a ese fichero, verifica la firma del token con la clave pública y lo intenta descargar.
4. Si todo es correcto, el servidor queda autorizado para realizar peticiones y renovaciones de certificados.
5. El cliente tan sólo tiene que mandar la solicitud o revocación cifrada con la clave privada del servidor.
6. La CA lo descifra con la clave pública y pasa a realizar las tareas de gestión oportunas.

Por lo que se ha podido observar, la tarea de solicitar un certificado SSL/TLS a Let's Encrypt que proteja las conexiones es bastante sencilla y automática. Además permite avanzar en las mejores prácticas de seguridad de TLS.

Cabe tener en cuenta que estos certificados permiten cifrar las comunicaciones para evitar ataques Man-in-the-Middle exitosos, sin embargo, no garantizan la veracidad del sitio ni del dominio.

4.2.2 Desconfianza en los certificados Validados por Dominio

Los certificados SSL/TLS no solo cifran los datos, sino que también autentican sitios web. Son bastante importantes puesto que generan confianza, ya que muchas personas al ver que la conexión utiliza el protocolo HTTPS y el candado verde en el navegador, creen que el sitio web es auténtico. Sin embargo, existen Autoridades de Certificación como Let's Encrypt que no utilizan niveles de autenticación altos ni comprueban de manera exacta a los propietarios de sitios web a la hora de emitirles un certificado. Por ello, muchos sitios web falsos utilizan este tipo de certificados para dotarlos de confianza.

Para actividades fraudulentas como el phising o la suplantación de identidad, los certificados SSL de confianza resultan esenciales, pues es muy difícil identificar si un sitio web está validado correctamente.

Para los certificados Validados por Dominio (DV) generalmente las Autoridades de Certificación sólo exigen que se demuestre que es el propietario del dominio. Cuya comprobación se suele hacer simplemente enviando un correo al email registrado en la información de ese dominio. Esto no garantiza que el sitio web no sea fraudulento.

Para combatir este tipo de fraudes, algunas Autoridades de Certificación emiten certificados de validación extendida (Extended Validation) mediante niveles de autenticación más altos. Estos son bastante más caros pero sí garantizan propietariedad de la empresa u organización y lo identifican a sus visitantes poniendo el nombre en verde en la barra de direcciones, como se puede observar en la siguiente figura.

Figura 10: Identificación de la compañía mediante certificado EV

4.3 Problemas detectados

HTTP en sus primeras versiones como HTTP/0 o HTTP/1.1 era muy sensible a la latencia y en enlaces de alta latencia, como es el caso de las tecnologías móviles de hoy en día, es complicado conseguir una rápida y buena experiencia de usuario en los sitios web, incluso contando con buen ancho de banda. Esto se debe a que para cargar contenido web es necesario el uso de múltiples conexiones TCP concurrentes para poder descargar todos los elementos de la web. Es decir, tiene que establecer una conexión TCP por cada elemento de la web, lo que conlleva también a ofrecer lentitud en el establecimiento de la conexión.

Cuando utilizamos HTTP sobre TLS estamos obligando a que previamente a establecer la conexión TCP, es necesario realizar la fase de handshake para negociar los parámetros de seguridad y la clave de cifrado, lo cual proporciona mayor número de viajes de ida y vuelta de los paquetes y, por lo tanto, mayor lentitud sumado a la latencia pertinente. Esto sumado a la lentitud que ya prevé HTTP, hace que las conexiones cifradas no sean eficientes y no se usen nada más que para casos específicos.

Otro gran problema que se plantea a HTTP es que no utiliza prioridades al enviar varias solicitudes consecutivas sin haber recibido respuestas (HTTP Pipelining). Esto permite que ocurra el llamado “Bloqueo del primero de la fila” (head of line blocking). Por tanto, puede que existan usuarios que reenvíen múltiples paquetes o tengan malos requisitos de red, y que provoquen bloqueos en la conexión, afectando a la experiencia del resto de usuarios en la web.

4.4 Alternativas para solucionar estos problemas

Durante años, los desarrolladores web iban solventando los problemas de HTTP realizando pequeños trucos y herramientas como “spriting” [38], “inlining” [39] o “sharding” [39]. Sin embargo, los desarrolladores se han dado cuenta de que la mejor solución no es introducir mejoras o cambios a un protocolo con carencias, sino desarrollar nuevas soluciones o nuevos protocolos diseñados específicamente para adoptar la funcionalidad y corregir defectos mediante técnicas más renovadas. Para ello se han creado nuevos protocolos que mejoran la seguridad y velocidad de las solicitudes y que permiten la combinación con el protocolo de seguridad TLS 1.3, lo que ya mejora la velocidad en el establecimiento de conexión.

Entre estos protocolos se encuentran SPDY, HTTP2, DTLS y QUIC, que se abordan en el capítulo siguiente.

Capítulo 5. Protocolos alternativos a HTTPS

5.1 SPDY

SPDY [40] es un protocolo de nivel de sesión según el modelo OSI que funciona sobre TCP/IP. Surgió a mediados del 2009 para mejorar la seguridad que ofrecía el protocolo HTTP en su versión HTTP/1.1 y conseguir una mayor eficiencia y rapidez. SPDY no reemplaza al protocolo HTTP, sino que añade una capa a nivel de sesión que permite mejorar la funcionalidad ya existente pudiendo enrutar mayor número de flujos de comunicación mediante la misma conexión TCP. En la siguiente tabla se puede observar que entre la capa de aplicación y la de presentación se incluye la capa de sesión cuyo protocolo es SPDY.

Application	HTTP
Session	SPDY
Presentation	SSL
Transport	TCP

Tabla 1: Pila de protocolos donde se incluye SPDY entre HTTP y SSL

Además de buscar el dotar de mayor seguridad a las comunicaciones, su principal propósito de es reducir el tiempo de carga de las páginas web. Para ello, implementa las siguientes características:

- Multiplexación de conexiones: Permite realizar diferentes peticiones concurrentes mediante una única conexión TCP, minimizando la sobrecarga que produce la negociación a tres vías (Three-way Handshake[41]) del protocolo TCP.
- Evita el problema Head-Of-Line Blocking[42] al permitir especificar prioridades en las distintas peticiones de recursos.
- Las cookies de sesión tan sólo se retransmiten si su valor cambia desde que se enviaron.
- Reduce el consumo de ancho de banda al comprimir todas las cabecera HTTP y eliminar las innecesarias. Esto es útil puesto que las cabeceras de este protocolo puede llegar a ser grande, sobre todo en peticiones con pocos datos.
- El servidor puede empezar a enviar datos al cliente sin que éste la haya solicitado antes. Esto permite que, si el servidor sabe que el cliente

necesitará un recurso determinado, se ahorren los costes de tener que realizar la petición. Esta mejora se conoce como “Server Push”.

En cuanto a velocidad, a diferencia de SPDY, el uso de HTTP conlleva gran latencia y, por consiguiente, mayor lentitud debido a que es un protocolo que basado en petición-respuesta, es decir, las conexiones se establecen y se terminan bajo petición del cliente al servidor. Google asegura haber obtenido resultados de cargas de un 64% más rápidas que con el protocolo HTTP.

Sin embargo, en cuanto a seguridad, no existe gran diferencia entre HTTPS y SPDY, pues SPDY hace que el uso de TLS sea obligatorio.

5.2 HTTP/2

El protocolo HTTP/2 [43] surge en el año 2012 (aunque se empieza a utilizar en los navegadores en el año 2015), con el objetivo de mejorar la velocidad y latencia del protocolo SPDY. Durante el año 2015 SPDY se abandona y se empieza a utilizar HTTP/2 puesto que la mayoría de ventajas que aportaba SPDY también se encuentran en este protocolo, añadiendo mejoras como la compresión de cabeceras HPACK.

Este nuevo protocolo logró mejorar hasta en un 60% la velocidad de carga de las páginas web estándar y hasta en un 55% las conexiones protegidas con cifrado SSL/TLS.

Cuenta con la ventaja comentada en SPDY del uso de una única conexión para ofrecer múltiples solicitudes y respuestas en paralelo. Esto aumenta la velocidad ya que, si tenemos en cuenta que las páginas web contienen objetos HTML, CSS, JS, imágenes, etc, la diferencia en velocidad de usar una única conexión para enviar todos estos objetos o utilizar una conexión para cada uno de ellos, es bastante notoria.

En la siguiente tabla se puede observar las diferencias más significativas entre ambos protocolos [44]:

SPDY	HTTP/2
Requiere TLS.	No requiere TLS.
Conexiones cifradas rápidas. Utiliza la extensión NPN.	Conexiones cifradas más rápidas. La nueva extensión ALPN [45] permite a los navegadores y servidores determinar qué

	protocolo de aplicación usar durante la conexión inicial.
Multiplexación de un solo servidor. La multiplexación ocurre en un host a la vez.	Multiplexación de múltiples hosts. La multiplexación ocurre en diferentes hosts al mismo tiempo.
Compresión DEFLATE. Deja un pequeño espacio para las vulnerabilidades.	Compresión HPACK. Es más rápida y segura. Está diseñado para acortar encabezados y prevenir vulnerabilidades.
Priorización.	Priorización mejorada. Permite a los navegadores web determinar cómo y cuándo descargar el contenido de una página web de manera más eficiente.

Tabla 2: Comparación entre SPDY y HTTP/2

El protocolo HTTP/2 fue diseñado para aprovechar al máximo posible el ancho de banda de la red mediante las técnicas de compresión y descargas paralelas. Esto permite menor sobrecarga de la red y menor uso de memoria en los servidores.

Si bien, siguiendo la temática de este proyecto, en cuanto a seguridad extraña que HTTP/2 no requiera el uso obligatorio de TLS. Aún así, dos de los navegadores más destacados como son Firefox y Chrome implementan HTTP/2 sobre TLS, no lo incorporan como un requisito opcional, sino obligatorio, y es que hoy en día estos fabricantes no conciben la idea de la transmisión de datos por la red sin cifrar. Además, HTTP/2 sobre TLS requiere el uso de TLS 1.2 o superior, de lo contrario se cancela la conexión, lo que hace que la buena seguridad y el rendimiento vayan de la mano.

Asumimos que TLS 1.3 es un gran avance para el rendimiento y la seguridad web. Al utilizar TLS 1.3 con 0-RTT, las ganancias de rendimiento son aún más significativas. Por lo que combinando esto con HTTP/2 resulta que obtenemos la mayor rapidez posibles en las comunicaciones cifradas.

Por último, indicar que HTTP/2 mantiene la compatibilidad total con protocolos sucesores como HTTP/1.1 y SPDY, lo que permite a los desarrolladores tiempo para la migración de sus aplicaciones webs sin afectar a sus usuarios.

5.3 QUIC

QUIC (Quick UDP Internet Connection) es un protocolo experimental en el que Google está trabajando desde julio de 2016. En el mes de abril de 2018, se publicó la versión 11 del borrador de QUIC [8].

Este protocolo surge con la idea de evitar conexiones innecesarias y mejorar la velocidad y latencia tanto en la transmisión de datos entre clientes y servidores en escenarios como en la conexión, donde la transferencia mediante TCP no es lo suficiente estable y los servicios son sensibles a la latencia como, por ejemplo, al intentar navegar mientras viajamos en metro donde la experiencia se interrumpe constantemente por pérdidas de señal.

El objetivo principal es mejorar el rendimiento percibido de las aplicaciones web orientadas a la conexión que actualmente utilizan TCP. QUIC tiene como objetivo obtener lo mejor de los protocolos UDP y TCP. Soporta un conjunto de conexiones multiplexadas a través de UDP y ha sido diseñado para proporcionar una seguridad equivalente a TLS. Consigue menor latencia a la hora de establecer las conexiones pues se requieren menos viajes de ida y vuelta, tiene un control mejorado de la congestión y permite una mejor recuperación tras las pérdidas de conexión.

En una conexión TCP, un único paquete perdido paraliza todos los flujos HTTP/2 o SPDY multiplexados en esa conexión. No obstante, QUIC con UDP soporta entregas desajustadas de manera que si se pierde un paquete se paraliza como mucho un flujo [46].

Además, QUIC permite que si un cliente ha conectado anteriormente con un servidor determinado, puede comenzar a enviar datos sin tener que negociar previamente los parámetros de seguridad, lo que hace que las páginas web se carguen más rápido. En la siguiente imagen extraída del blog de Chromium [47] se compara el número de RTT y el tiempo que se tarda en resolver una petición HTTP, HTTPS y QUIC.

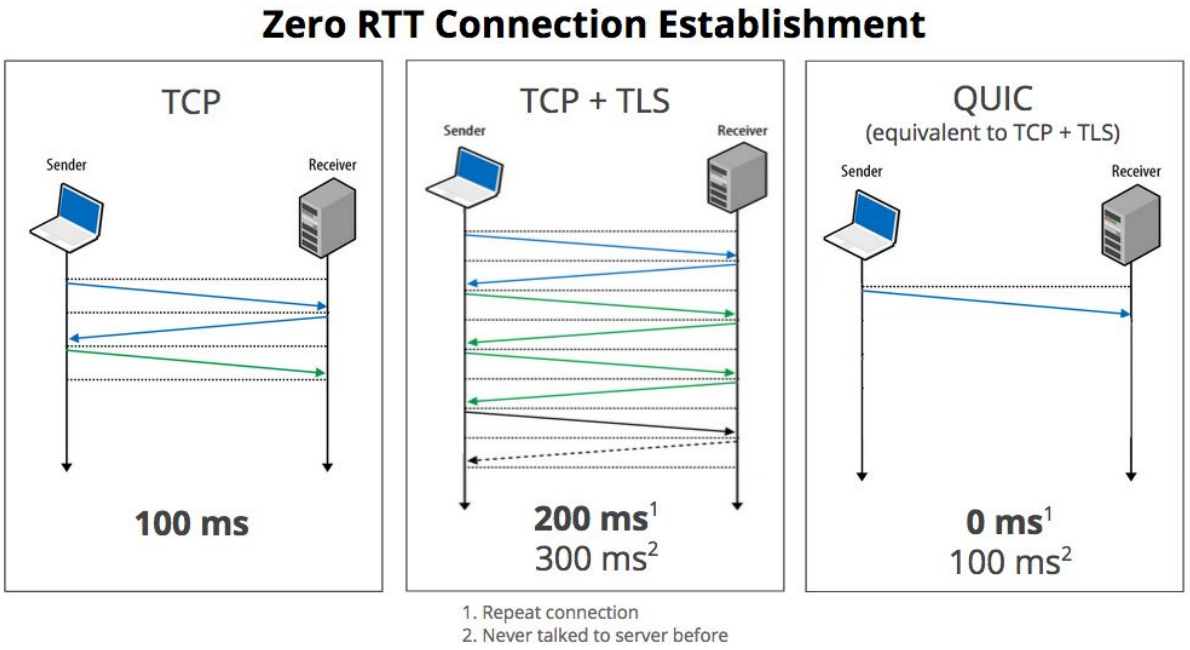


Figura 11: Comparación de TCP, TCP+TLS y QUIC en el establecimiento de conexión 0-RTT

Este protocolo es muy adecuado para aplicaciones de baja latencia, incluso los clientes móviles tienden a ganar mucho con este enfoque.

Por tanto, al igual que HTTPS, QUIC también se implementa sobre TLS ofreciendo una capa de seguridad a las conexiones, no obstante adquiere mayor velocidad puesto que opera sobre UDP y no sobre TCP. Es muy adecuado para aplicaciones en las que no resulte esencial trabajar sobre orientación a conexión y pérdida o desajuste del orden de los paquetes y sobre todo en las que se requiera velocidad, pues QUIC soporta TLS 1.3 y 0-RTT. En la siguiente tabla se comparan gráficamente las capas de los protocolos HTTP/2 y QUIC.

<table>
<tr><th>HTTP/2</th><th>HTTP over QUIC</th></tr>
<tr><td>TLS</td><td rowspan="2">QUIC
TLS 1.3
TCP-like congestion control, loss recovery</td></tr>
<tr><td rowspan="2">TCP</td></tr>
<tr><td>UDP</td></tr>
<tr><td colspan="2">IP</td></tr>
</table>

Tabla 3: Comparación de capas de los protocolos HTTP/2 y QUIC

Si bien, por ahora tan solo es un protocolo experimental de Google y solamente Chrome y los servidores de Google admiten este protocolo, por ahora existen algunas implementaciones de código abierto en Go. De todas formas, según explican en el blog de Chromium [47], tienen la intención de proponer formalmente QUIC al IETF como un estándar de Internet, así mismo pretenden implementar SPDY sobre QUIC y HTTP/2 sobre QUIC.

En la siguiente imagen, podemos ver una comparación de la pila de protocolos.

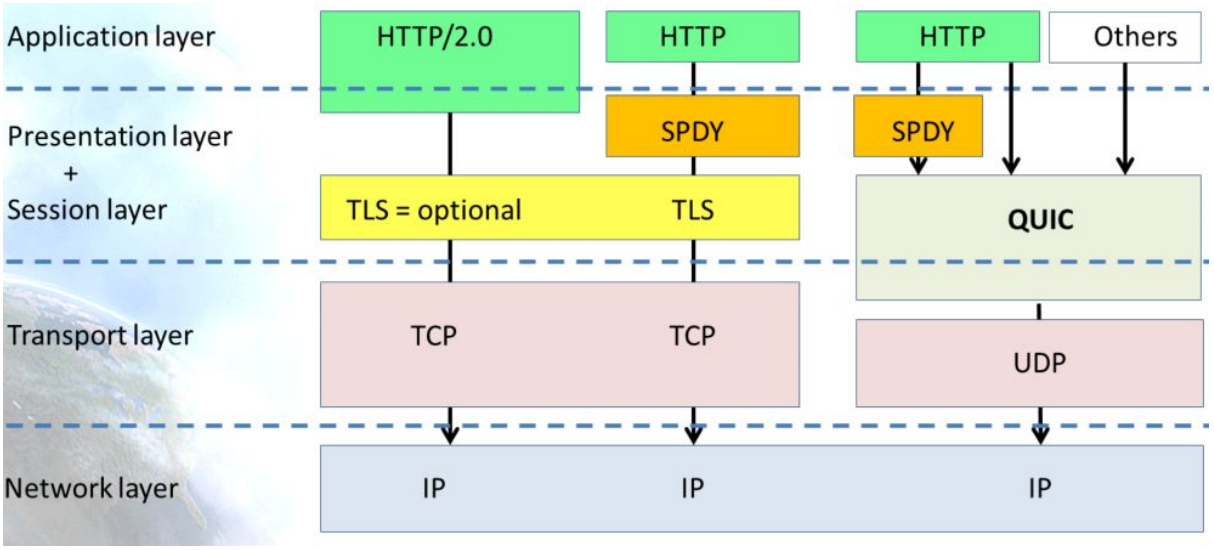


Figura 12: Comparación de capas de trabajo entre HTTP/2, HTTPS y HTTP + SPDY

5.4 DTLS

El protocolo DTLS (Datagram Transport Layer Security)[20] está basado en el protocolo TLS, con la diferencia de que DTLS opera sobre protocolos de datagramas como UDP (User Datagram Protocol). Al igual que QUIC, mejora la solución de dos problemas principales en TCP como la pérdida y la reordenación de paquetes.

DTLS está diseñado para proporcionar garantías de seguridad similares a TLS, nos permite baja latencia, alta velocidad de transferencia de datos y tolerancia a la pérdida de comunicación, sin embargo, DTLS no proporciona fiabilidad ni tampoco transporte en orden de los datos, tal y como ocurre con el protocolo UDP.

Actualmente la versión DTLS 1.2 [48], a diferencia de TLS 1.2 implementa técnicas de detección de repeticiones. Las principales diferencias entre DTLS 1.2 y TLS 1.2 son las siguientes:

- **Datagramas:** TLS divide los datos en registros con una longitud fija, en DTLS los registros se dividen en datagramas de mayor tamaño.
- **Números de secuencia explícitos.** En TLS los registros incluyen un código MAC de 64 bits que permite garantizar la integridad del mismo, este código incluye implícitamente un número de secuencia de registro que permite verificar que ningún registro se ha perdido o desordenado. En DTLS, el número de secuencia es explícito en cada registro.
- **Se toleran las alteraciones.** DTLS tolera que los datagramas puedan perderse, duplicarse o desordenarse dado que utiliza un mecanismo de "ventana" en la que se mantienen los registros en un buffer hasta que lleguen o se descarten los anteriores (por ejemplo se reciben los datagramas 1, 2 y 5; el datagrama "5" queda en un buffer hasta que se reciban el 3 y 4 o se omiten si se han perdido). En líneas generales, los registros anómalos (faltantes, duplicados, etc) simplemente se descartan y se recibe el resto. En el caso de que haya muchos errores se puede advertir al usuario. Sin embargo, TLS no tolera de manera tan flexible casos en los que haya mucho "ruido" en la conexión.
- **Cifrado sin estado.** En DTLS, como los registros pueden perderse, el cifrado no debe utilizar un estado que se modifique en cada registro como se hace en RC4.
- **No se verifica el fin de conexión.** En TLS se utiliza el mensaje de alerta close_notify para advertir del fin de la conexión. En DTLS no se puede saber si el emisor ha dejado de enviar datos porque ha terminado la conexión o si es que los datos se han perdido. Se supone que DTLS debe emplearse en contextos en los que el saber el fin de la conexión no tiene sentido.

El dominio de aplicación de DTLS es distinto del de TLS; está destinado a ser aplicado a aplicaciones de transmisión de datos donde las pérdidas son menos importantes que la latencia, por ejemplo, VoIP o feeds de vídeo en vivo. Para una aplicación determinada, TLS tiene más sentido que DTLS.

El sistema de nombres de dominio (protocolo DNS), uno de los protocolos fundamentales de Internet, funciona sobre los protocolos de la capa de transporte TCP y UDP, por lo que no lleva ningún tipo de cifrado ni autenticación punto a punto. DTLS es más adecuado que TLS para el transporte de consultas DNS puesto que el transporte de datagramas es más rápido y permite cifrar los datos.

Si en lugar de utilizar DNS sobre DTLS lo utilizáramos sobre TLS 1.2, veríamos que las consultas DNS serían mucho más lentas, sin embargo el RFC 8310 [49] explica los perfiles de uso para DNS sobre TLS y DNS sobre DTLS.

Por otro lado, al igual que para TLS, en marzo de 2018 se publicó una nueva versión borrador del protocolo DTLS 1.3 [50] que pretende convertirse en el nuevo estándar de DTLS.

Capítulo 6. Conclusiones y líneas futuras

Como proyecto de fin de posgrado decidí investigar sobre los protocolos SSL y TLS porque creo que la transmisión de datos seguros mediante la red tiene un crecimiento exponencial. Hoy en día almacenamos en la nube una gran cantidad de datos personales, imágenes, vídeos, etc, compramos online con los datos de nuestras tarjetas, consultamos nuestras cuentas bancarias y tarde o temprano todo el mundo acabará transmitiendo datos que deben estar protegidos.

Además, creo que TLS es un protocolo bastante robusto que lleva más de 20 años securizando las comunicaciones, siendo atacado y actualizándose y protegiéndose en versiones nuevas, pero siempre siendo el estándar utilizado.

Durante el desarrollo de este proyecto, gracias a la investigación y estudio que he realizado, he podido ampliar el conocimiento sobre el protocolo TLS y la importancia de cifrar los datos para transmitirlos por la red, así como el proceso, que era algo que me llamaba bastante la atención, sobre todo tras haber cursado el Posgrado de seguridad en redes y sistemas y haberme adentrado en el mundo de la ciberseguridad. También, he profundizado en conocer cómo los atacantes consiguen aprovechar vulnerabilidades existentes en las diferentes versiones de SSL/TLS, algo que me resultó bastante interesante, pues no me ha parecido un proceso sencillo el atacar las vulnerabilidades de este protocolo, sin embargo, muchos de estos ataques se han saldado con gran éxito.

Respecto a conclusiones específicas que he sacado a lo largo del desarrollo del proyecto, indicar que me parecen bien las medidas a tomar en los principales navegadores, que no permitan el uso de versiones antiguas y vulneradas del protocolo TLS, sino que tan sólo sean compatibles con versiones nuevas, las cuales ya corrigen estas debilidades y deficiencias e implementan métodos de cifrado que actualmente no han sido comprometidas y utilizan algoritmos más robustos.
Por otro lado, me parece acertada la medida tomada por Google, que catalogará como inseguros las aplicaciones o webs que no utilicen certificados SSL para cifrar sus comunicaciones, pienso que hoy en día todo lo que se transmite por la web debe ir cifrado, pues ya se hace negocio con datos que ignoramos que podrían ser útiles y hay demasiados ciberdelincuentes en Internet que pueden hacer bastante daño con la filtración de ciertos datos.

No obstante, algo que me gustaría criticar sería que se siga manteniendo la codificación de relleno PKCS#1 1.5, para mi parecer tendría que haber quedado en desuso tras descubrir el ataque de Bleichenbacher, y no seguir manteniéndola en

los protocolos nuevos para conservar la compatibilidad. Es insegura y tendrá esas deficiencias siempre. Por lo que, puestos a mejorar los estándares, yo no permitiría conservar métodos inseguros.

Por último, como líneas futuras propongo la creación de una guía que indique cómo configurar de manera correcta las propiedades de seguridad de un servidor para que utilice solamente protocolos seguros como TLS 1.2 (de momento) y TLS 1.3 (una vez se publique), así mismo, que indique cómo debe configurar los navegadores un cliente para que solamente se comunique con servidores que implementen protocolos seguros. Esta idea me parece bastante interesante, porque puede que haya empresas que quieran proporcionar conexiones seguras y robustas pero se olviden o desconozcan el ajustar bien las propiedades de seguridad.

Así mismo, sería bastante interesante investigar cómo ligar TLS a DNS para establecer seguridad en las peticiones DNS y que no puedan ser leídas o modificadas por un tercero que monitorice la conexión, puesto que el primer paso de casi todas las conexiones en Internet es una consulta de DNS.

En cuanto a la satisfacción personal lograda he de decir que he cumplido todos los objetivos que se plantearon al inicio de este proyecto, quizá me ha faltado tiempo para hacer un análisis con mayor grado de profundidad y me hubiese gustado experimentarlo de manera práctica y hacer comparaciones jugando con las propiedades sobre TLS que permiten los navegadores. No obstante, aunque no quede reflejado en esta memoria, sé que seguiré en esta línea de investigación, pues me ha parecido bastante interesante. He de concluir afirmando que he alcanzado las metas propuestas adquiriendo una gran satisfacción personal tras comprobar el resultado conseguido y el trabajo realizado.

Bibliografía

[1] Transport Layer Security (TLS). O' Reilly. High Performance Browser Networking. Fecha de última consulta: 20/05/2018. https://hpbn.co/transport-layer-security-tls/#

[2] The TLS Protocol Version 1.0. Ene 1999. T. Dierks, C.Allen (Certicom). https://www.ietf.org/rfc/rfc2246.txt

[3] SSL vs. TLS - ¿Cuál es la diferencia?. 19 sep 2016. Eduardo Zambrano (GlobalSign). Fecha de última consulta: 16/03/2018. https://www.globalsign.com/es/blog/ssl-vs-tls-difference/

[4] Paper: Attacking RSA-Based Sessions in SSL/TLS. 2003. Vlastimil Klíma, Ond ej Pokorný, and Tomáš Rosa. https://link.springer.com/content/pdf/10.1007%2F978-3-540-45238-6_33.pdf

[5] Fuga de información en implementaciones TLS con cifrado RSA. 13 dic 2017. Certsi. Fecha de última consulta: 24/03/2018. https://www.certsi.es/alerta-temprana/avisos-seguridad/fuga-informacion-implementaciones-tls-cifrado-rsa

[6] Logjam: nueva vulnerabilidad crítica en TLS. 20 may 2005. Fecha de última consulta: 25/03/2018. https://blog.segu-info.com.ar/2015/05/logjam-nueva-vulnerabilidad-critica-en.html

[7] Vulnerabilidades en TLS de Mbed. 06 feb 2018. Certsi. Fecha de última consulta: 26/04/2018. https://www.certsi.es/alerta-temprana/avisos-seguridad/vulnerabilidades-tls-mbed

[8] QUIC: A UDP-Based Multiplexed and Secure Transport. 17 Abr 2018. J. Iyengar (Fastly) y M. Thomson (Mozilla). https://datatracker.ietf.org/doc/draft-ietf-quic-transport/

[9] PROYECTO INTEGRADO HTTP/2. 14 Jun 2018. Juan Luis Ramírez Vaquero. http://informatica.gonzalonazareno.org/proyectos/2016-17/Juan%20Luis%20Ramirez-ProyectoFinal-HTTP2.pdf

[10] The Transport Layer Security (TLS) Protocol Version 1.1. Abr 2006. T. Dierks, E. Rescorla (RTFM, Inc) . https://www.ietf.org/rfc/rfc4346.txt

[11]The Transport Layer Security (TLS) Protocol Version 1.2. Ago 2008. T. Dierks, E. Rescorla (RTFM, Inc) . https://tools.ietf.org/html/rfc5246

[12] The Transport Layer Security (TLS) Protocol Version 1.3
draft-ietf-tls-tls13-28. 20 mar 2018. E. Rescorla (RTFM, Inc)
https://tools.ietf.org/html/draft-ietf-tls-tls13-28

[13]Sitio oficial del Consejo de Normas de Seguridad de la Industria de Tarjetas de Pago. PCI Security Standards Council, LLC. Fecha de última consulta: 20/03/2018
https://es.pcisecuritystandards.org/minisite/env2/

[14] Transport Layer Security (TLS): qué es y cómo funciona. 2 oct 2014. Fecha de última consulta: 15/03/2018
https://www.swhosting.com/blog/transport-layer-security-tls-que-es-y-como-funciona/

[15] Message authentication code. 1 abr 2017. Fecha de última consulta: 22/03/2018
https://es.wikipedia.org/wiki/Message_authentication_code

[16] Attack of the week: POODLE. 15 oct 2014. Matthew Green. Fecha de última consulta: 23/03/2018 https://blog.cryptographyengineering.com/2014/10/15/attack-of-week-poodle/

[17] CVE-2014-8730 https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-8730

[18] CVE-2014-3566 https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-3566

[19] CVE-2014-0160 https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2014-0160

[20] Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension. Feb 2012. R. Seggelmann, M. Tuexen y M. Williams https://tools.ietf.org/html/rfc6520

[21] CVE-2015-0204 https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2015-0204

[22] CVE-2015-2235 https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-2235

[23] CVE-2015-1637 https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-1637

[24] CVE-2015-4000 https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-4000

[25] Details on the "Crime" Attack. 14 sep 2012. Tom Ritter. Fecha de última consulta: 25/03/2018
https://www.nccgroup.trust/us/about-us/newsroom-and-events/blog/2012/september/details-on-the-crime-attack/

[26] CVE-2012-4929 https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2012-4929

[27] Security Losses from Obsolete and Truncated Transcript Hashes (CVE-2015-7575). Karthikeyan Bhargavan y Karthikeyan Bhargavan (INRIA researchers). Fecha de última consulta: 25/03/2018 https://www.mitls.org/pages/attacks/SLOTH

[28] CVE-2015-7575 https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-7575

[29] Return Of Bleichenbacher's Oracle Threat (ROBOT)
https://eprint.iacr.org/2017/1189.pdf

[30] CVE-2017-17382 https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-17382

[31] PKCS #1: RSA Encryption Version 1.5. RFC 2313. Mar 1998. B. Kaliski.
http://www.ietf.org/rfc/rfc2313.txt

[32] SSL/TLS-based malware attacks. 2 Ago 2017. Deepen Desai. Fecha de última consulta: 02/04/2018 . https://www.zscaler.com/blogs/research/ssltls-based-malware-attacks

[33] Introducing Zero Round Trip Time Resumption (0-RTT). 15 Mar 2017. Nick Sullivan. Fecha de última consulta: 16/04/2018
https://blog.cloudflare.com/introducing-0-rtt/

[34] RFC 2616. Hypertext Transfer Protocol -- HTTP/1.1 . Junio 1999. IETF.
https://tools.ietf.org/html/rfc2616

[35] RFC 2818. HTTP Over TLS. Mayo 2000. IETF.
http://www.ietf.org/rfc/rfc2818.txt

[36] A secure web is here to stay. 8 Feb 2018. Emily Schechter, Chrome Security Product Manager. Fecha de última consulta: 20/04/2018
https://security.googleblog.com/2018/02/a-secure-web-is-here-to-stay.html

[37] Let's Encrypt. Linux Foundation. Fecha de última consulta: 25/04/2018
https://letsencrypt.org/

[38] Reducing HTTP Requests with Sprite Sheets. 3 Oct 2013. Apple Inc. Fecha de última consulta: 26/04/2018
https://developer.apple.com/library/content/documentation/NetworkingInternet/Conceptual/SafariImageDeliveryBestPractices/ReducingHTTPRequestswithSprites/ReducingHTTPRequestswithSprites.html

[39] HTTP/1.X. Año 2013. Ilya Grigorik, web performance engineer at Google and co-chair of the W3C Web Performance Working Group. Fecha de última consulta: 26/04/2018
https://hpbn.co/http1x/

[40] SPDY: An experimental protocol for a faster web. Chromium blog. Fecha de última consulta: 30/04/2018.
https://www.chromium.org/spdy/spdy-whitepaper

[41] TCP Connection Management – TCP Connection Establishment (Three-Way Handshake). 9 Sep 2014. Felipe Henriquez. Fecha de última consulta: 17/04/2018
http://www.redescisco.net/sitio/2014/09/09/tcp-connection-management-tcp-connection-establishment-three-way-handshake/

[42] Impact of the Head-of-Line Blocking on Parallel Computer Networks: Hardware to Applications. V. Puente, J.A. Gregorio Universidad de Cantabria y C. Izu, R. Beivide Universidad de Adelaida. https://www.ce.unican.es/pdf/HOLB-EUROPAR-1999.pdf

[43] RFC 7540. Hypertext Transfer Protocol Version 2 (HTTP/2). May 2015. M. Belshe (BitGo), R.Peon (Google) y M. Thomson (Mozilla).
https://tools.ietf.org/html/rfc7540

[44] A Simple Performance Comparison of HTTPS, SPDY and HTTP/2. 16 Ene 2015. HttpWatch. Fecha de última consulta: 02/05/2018.
https://blog.httpwatch.com/2015/01/16/a-simple-performance-comparison-of-https-spdy-and-http2/

[45] RFC 7639. The ALPN HTTP Header Field. Ago 2015. A. Hutton (Unify), J. Uberti (Google) y M. Thomson (Mozilla).
https://tools.ietf.org/html/rfc7639

[46] QUIC Quick UDP Internet Connections. 5 Jun 2015. Jim Roskind. Fecha de última consulta: 03/05/2018
https://docs.google.com/document/d/1RNHkx_VvKWyWg6Lr8SZ-saqsQx7rFV-ev2jRFUoVD34/edit

[47] A QUIC update on Google's experimental transport. 17 Abr 2015. Alyssa Wilk, Ryan Hamilton and Ian Swett (Chromium). Fecha de última consulta: 03/05/2018,
https://blog.chromium.org/2015/04/a-quic-update-on-googles-experimental.html

[48] RFC 6347. Datagram Transport Layer Security Version 1.2. Ene 2012. E. Rescorla (RTFM) y N. Modadugu (Google).
https://tools.ietf.org/html/rfc6347

[49] RFC 8310. Usage Profiles for DNS over TLS and DNS over DTLS. 21 Mar 2018. S. Dickinson (Sinodum), D. Gillmor (ACLU) y T. Reedy (McAfee).
https://datatracker.ietf.org/doc/rfc8310/

[50] The Datagram Transport Layer Security (DTLS) Protocol Version 1.3. 5 Mar 2018. E. Rescorla (RTFM), H. Tschofenig (Arm Limited), N. Modadugu (Google).
https://datatracker.ietf.org/doc/draft-ietf-tls-dtls13/