Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Creación de una autoridad certificadora de firmas digitales y servidor OCSP. Análisis de ataques MITM

Autor: Francisco Jesús Marchal Cebador

Tutor: Francisco José Fernández Jiménez

Dep. Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2015

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Creación de una autoridad certificadora de firmas digitales y servidor OCSP. Análisis de ataques MITM

Autor:
Francisco Jesús Marchal Cebador

Tutor:
Francisco José Fernández Jiménez
Profesor colaborador

Dep. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2015

Trabajo Fin de Grado: Creación de una autoridad certificadora de firmas digitales y servidor OCSP. Análisis de ataques MITM

Autor: Francisco Jesús Marchal Cebador

Tutor: Francisco José Fernández Jiménez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2015

El Secretario del Tribunal

A mi familia

A mis amigos

A mis maestros

Agradecimientos

La realización de este proyecto no habría sido posible sin el apoyo y ayuda de numerosas personas, que de una forma u otra han aportado su granito de arena.

En primer lugar agradecer a mi tutor, Francisco José, que me ha dado las directrices para el desarrollo del trabajo y ha estado disponible para ayudarme a solucionar dudas o problemas que han ido surgiendo.

A mis padres, que han estado ahí aguantándome y apoyándome tanto durante la realización de este proyecto como a lo largo de toda mi vida. Agradecer su esfuerzo y dedicación para que todo salga adelante y pueda cumplir mis sueños. Sin vosotros no estaría donde estoy hoy. Muchísimas gracias.

Al resto de mi familia, que siempre están ahí dando ánimos y que se alegran incluso más que yo de todos mis éxitos. Gracias.

A todos mis amigos de la universidad, Javi, Migue, Adri, Richy, Jairo, Alberto, etc. Gracias por estar ahí. Entre todos ha sido posible el hecho de llegar hasta aquí.

A mis amigos Dani y Fran, ya sabéis que sin vosotros no estaría escribiendo estas líneas. Por acompañarme desde el principio de este viaje. Por estar en los buenos y malos momentos, y aguantarme a lo largo del montón de horas que hemos echado juntos. Muchas gracias.

A mis amigos de siempre, por interesarse a cada instante. Que han sufrido casi tanto como yo el sacrificio que supone este reto. Gracias.

No quiero dejarme atrás a todos los profesores que desde pequeño me han formado para que sea la persona que soy. En especial a Juan, por apostar por mí y hacerme ver lo que realmente me gustaba. Por su optimismo y entrega. Gracias.

Este proyecto supone el fin de estos cuatro años en los que todos habéis aportado algo en mi vida, y el comienzo de nuevos objetivos en los que espero que sigáis a mi lado.

Francisco Jesús Marchal Cebador

Sevilla, 2015

Resumen

La conexión segura entre un cliente y un servidor mediante HTTPS es de las más empleadas actualmente en las aplicaciones web. Para ello es necesario que los administradores de los servidores soliciten y paguen por un certificado a una autoridad de certificación, lo cual puede resultar un proceso complejo. Es por esto que muchos optan por mantener sus servicios sin cifrar, con el problema de seguridad que supone.

El objetivo de este proyecto es la puesta en marcha de un sistema que incluya una autoridad de certificación y un servidor OCSP para la validación, de forma que firme los certificados de forma transparente al administrador y éste pueda utilizarlos para su servicio. Se parte de la idea de que en un futuro todas las aplicaciones se comuniquen de forma cifrada.

Para comprobar que esta autoridad genera certificados de confianza, se ha diseñado un plan de pruebas que incluye un cliente que se conectará de forma segura con un certificado generado por la autoridad a un servidor Apache, haciendo uso de la herramienta *nogotofail* que intentará sabotear la conexión mediante ataques "Man In The Middle".

Abstract

Secure connection between a client and a server using HTTPS is currently one of the most used in web applications. This requires that server administrators request and pay for a certificate to a certification authority, which can be a complex process. This is why many of them choose to maintain their services without encrypting, with the security problem that it means.

The purpose of this project is the implementation of a system that includes a certification authority and an OCSP responder for validation, so it signs certificates transparently to administrators and they could use them for their services. It starts from the idea that in a future all applications will communicate in an encrypted way.

To check that this authority generates trusted certificates, it has been designed a testing plan that includes a client which will connect securely with a certificate generated by the authority to an Apache server, using the tool *nogotofail* that will try to sabotage the connection by attacks “Man In The Middle”.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Acrónimos

HTTPS	Hypertext Transfer Protocol Secure
SSL	Secure Socket Layer
TLS	Transport Layer Security
CA	Certification Authority
RA	Registration Authority
VA	Validity Authority
CRL	Certificate Revocation List
OCSP	Online Certificate Status Protocol
MITM	Man In The Middle
ACME	Automated Certificate Management Environment
POODLE	Padding Oracle On Downgraded Legacy Encryption
DES	Data Encryption Standard
3DES	Triple Data Encryption Standard
AES	Advanced Encryption Standard
RSA	Rivest, Shamir y Adleman (criptógrafos que crearon este sistema)
PKI	Public Key Infrastructure
IDEA	International Data Encryption Algorithm
DSA	Digital Signature Algorithm
DSS	Digital Signature Standard
MD5	Message-Digest Algorithm 5
SHA	Secure Hash Algorithm
PGP	Pretty Good Privacy
GPG	GNU Privacy Guard
SSH	Secure SHell
LDAP	Lightweight Directory Access Protocol
MAC	Message Authentication Code
CSR	Certificate Signing Request

1 INTRODUCCIÓN

La mayor gloria no es nunca caer, sino levantarse siempre.

- Nelson Mandela -

La seguridad en la red hoy en día es uno de los temas más cuestionados ya que la mayoría de nuestras acciones cotidianas se centran en Internet, como son el hecho de acceder a los servicios de nuestro banco, comprar online o revisar la información de nuestros perfiles sociales. Una gran cantidad de información se mueve cada segundo y nos preocupa el hecho de que nuestros datos pasen a manos de otros que puedan hacer un uso fraudulento de ellos.

Bastantes empresas han cifrado ya sus sitios web para evitar este tipo de ataques y asegurar la información de sus clientes, usando para ello la conexión HTTPS. Se basa en utilizar el protocolo de texto plano HTTP sobre SSL/TLS, los cuales establecen el envío de paquetes de forma cifrada. Actualmente esto es soportado por casi todos los servidores, navegadores y lenguajes de programación para aplicaciones.

Sin embargo, todavía hay muchos servidores que se encuentran sin cifrar y vulnerables a que ataquen sus datos. Entonces, ¿por qué no utilizan un protocolo seguro en lugar de emplear una comunicación en texto plano? La respuesta es que el proceso para solicitar un certificado necesario para la comunicación SSL/TLS suele ser bastante engorroso y costar dinero. Es necesario registrarse en una Autoridad de Registro (RA), que comprobará los datos y enviará una petición a una Autoridad de Certificación (CA), que firmará el certificado con su clave y lo devolverá a la RA. Una vez que le llega el certificado al administrador, éste debe configurarlo en su servidor correctamente junto con la clave pública dada por la CA para que esté disponible para los usuarios que se conecten a él.

Este es el principal motivo por el que sigue existiendo la falta de cifrado en la red. A esto se le suma el problema de que en las empresas existen varios servidores para proporcionar servicios, lo que hace este proceso aún más complejo.

Y ¿qué pasa con las conexiones cifradas? ¿Son totalmente seguras? La respuesta obviamente es que no. A pesar de que el servidor tenga configurado un certificado, es necesario comprobar que éste es correcto y que la conexión que se va a realizar es segura. Nos podemos encontrar en la red todo tipo de certificados, que estén auto-firmados, firmados por una autoridad desconocida, revocados, expirados, etc. El lado del cliente debe comprobar la validez de este certificado y protegerse también ante ataques posibles en la conexión, ya que podemos encontrarnos con que la conexión haya sido secuestrada y se nos proporcione un certificado falso.

En este proyecto se va a realizar una arquitectura con la idea de intentar buscar solución a estos problemas. En primer lugar se va a estudiar el estado del arte, es decir, lo que ya se encuentra desarrollado sobre este tema. Posteriormente se van a analizar las tecnologías a utilizar en el sistema, para entrar a detallar el diseño de los diferentes componentes de éste. Por un lado la implementación de una Autoridad de Certificación (que actuará también como Autoridad de Registro) y que proporcionará a los administradores el certificado

correspondiente. Esta autoridad seguirá el modelo cliente-servidor, actuando la autoridad en el lado del servidor y el servidor que solicita el certificado actuando como cliente, mediante un programa escrito en C que comunicará ambos componentes. Por otro lado, se expondrá la instalación de un servidor OCSP, que servirá para la validación que harán los clientes del certificado proporcionado por el servidor. Para este respondedor OCSP se usará la solución libre de OpenCA y se integrará con la CA creada en la misma máquina. Para completar esta arquitectura, se ha diseñado un cliente java que intentará realizar una conexión segura con un servidor Apache configurado con un certificado de la autoridad creada.

Para comprobar que todo este sistema es correcto, se harán una serie de pruebas sobre la conexión HTTPS, incluyendo las validaciones OCSP o con ataques "Man In The Middle" mediante la herramienta *nogotofail*, creada por Google para que los desarrolladores prueben sus aplicaciones y vean si son seguras, detallando su integración en la arquitectura creada.

Tras el desarrollo de todos los componentes, se estudiarán los resultados obtenidos en las diferentes pruebas. Se terminará con las conclusiones sacadas tras hacer este proyecto, así como futuras mejoras, y con una serie de anexos que incluirán las guías de instalación y uso de los diferentes componentes y el registro obtenido de las pruebas realizadas.

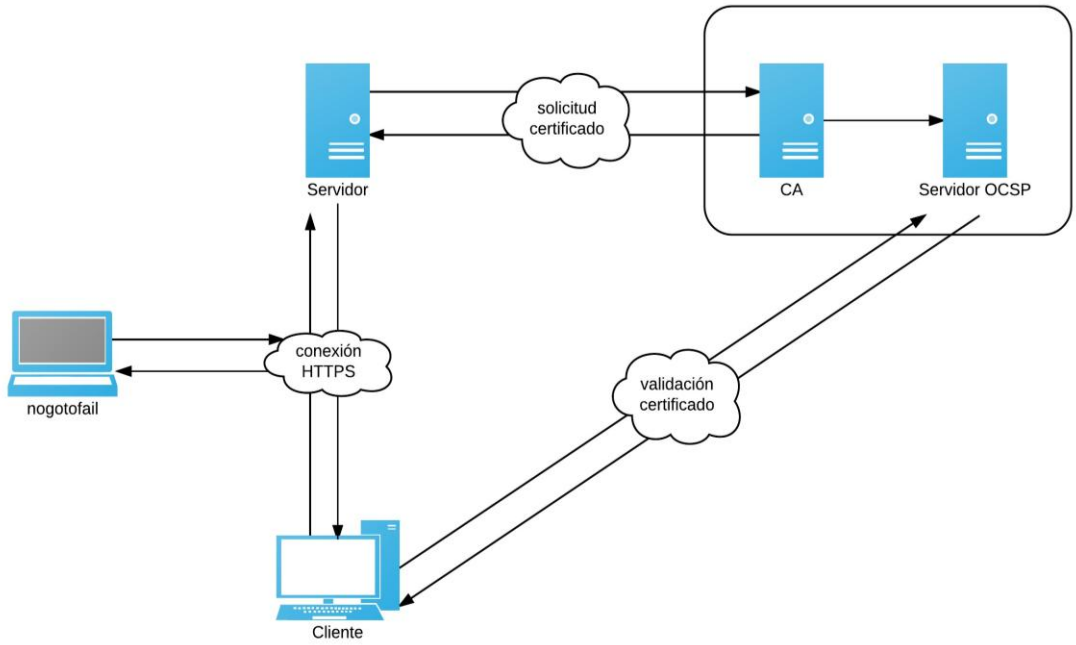

Figura 1-1. Arquitectura general del proyecto

2 MOTIVACIÓN Y OBJETIVOS

No tenemos la oportunidad de hacer muchas cosas, por lo que cada cosa que hagamos debe ser excelente. Porque esta es nuestra vida.

- Steve Jobs -

Habrá un día en el que todas las comunicaciones en la red estén cifradas. Esto parece un hecho bastante lejano pero en realidad no estamos tan lejos de conseguirlo. Ya son muchos los sitios web que integran la tecnología para este cifrado. Y la población en general cada vez está más concienciada de observar que esté bajo HTTPS sobre todo si se va a proporcionar información confidencial.

Pero para que todo esté cifrado es necesario realizar un esfuerzo tanto económico como de tiempo para proveer a las conexiones de este tipo de tecnología. Muchos no están dispuestos a perder horas de trabajo con esto o simplemente gastar el dinero cuando los datos con los que trabaja son de los usuarios. Existen soluciones de autoridades de certificación libres que firman certificados que se le solicitan. Sin embargo, puede ser que sea complicada su instalación o que no ofrezcan la configuración automática del certificado en el servidor.

2.1 Motivación

La principal motivación es que este trabajo sirva como estudio de cómo se puede automatizar y proporcionar a los administradores y desarrolladores una forma simple de cifrar sus conexiones, con la meta de que todas las comunicaciones estén cifradas en un futuro y sea más difícil el acceso a los datos.

El cifrado es necesario ya que en numerosas ocasiones no se puede evitar que alguien intercepte nuestros datos. Pero dado el caso, el cifrado permite que no pueda ver los datos y que tampoco pueda introducir información haciéndose pasar por el servidor o el cliente.

Se busca un método sencillo y seguro para solicitar un certificado, que automáticamente sea capaz de configurarlo en el servidor, que en este caso se ha elegido Apache por estar ampliamente extendido. Uno de los problemas que se presenta con esto es que al no tener el administrador que realizar ninguna acción, es difícil detectar que el servidor indicado es quien dice ser. Ésta es una de las funcionalidades que se quiere que incluya la CA a diseñar, de manera que tenga un procedimiento para validar el servidor antes de darle el certificado correspondiente.

Además, se probará todo esto con un ejemplo de cliente-servidor usando el protocolo HTTPS. Tanto SSL como TLS pueden ser usados con cualquier otro protocolo de comunicación, pero en este caso nos centraremos en HTTPS que es el más utilizado por los usuarios que acceden a Internet, ya que la mayoría lo hacen a través de navegadores que lo usan.

2.1.1 Proyecto Let's Encrypt

Esta idea se basa en el nuevo proyecto que está desarrollando una asociación de organizaciones y empresas tales como Cisco, Akami, Mozilla Corporation, Electronic Frontier Foundation, IdenTrust e investigadores de la Universidad de Michigan, formando parte del ISRG (Internet Security Research Group).

Let's Encrypt es una autoridad de certificación gratuita escrita principalmente en Python que se espera que esté lista para finales de 2015 y que presume de ser automática, segura, transparente, abierta y cooperativa. El objetivo de esto es la misma idea que hemos mostrado desde el principio, buscar que en un futuro todas las conexiones en Internet se encuentren cifradas.

El protocolo utilizado para automatizar el proceso de solicitar un certificado a esta autoridad se llama ACME, y de momento se puede encontrar un borrador de la especificación de éste[1]. En junio de 2015 se ha anunciado el periodo de pruebas y el primer certificado de la autoridad para poder descargárselo. Se espera que para septiembre de este año pueda estar operativo.

El presente trabajo busca desarrollar esta misma idea pero para un ámbito más reducido, con objeto de su posterior estudio. Se realizará la autoridad con las tecnologías existentes hasta el momento para ver que es posible automatizar todo este proceso con un programa escrito en uno de los lenguajes más conocidos: C. Se pretende conocer todas las partes de una autoridad y cómo realizar un programa que siga una especie de protocolo (sin llegar a utilizar ACME) para automatizar la solicitud y entrega de un certificado.

2.1.2 Seguridad en la conexión

Además de configurar el servidor, una de las mayores preocupaciones se encuentra en el lado del cliente. A pesar de que el servidor proporcione un certificado firmado digitalmente por una autoridad, el cliente deberá comprobar que esto es cierto y que no ha sido víctima de algún ataque.

Tanto autoridad como servidor van a utilizar OpenSSL para trabajar con los certificados, ya que es una de las herramientas de software libres más completas. Sin embargo, últimamente han surgido diferentes problemas de seguridad usando esta herramienta.

Algunas veces el agujero de seguridad se encuentra directamente en el lado del servidor, como es el caso de *heartbleed*, una vulnerabilidad que apareció en abril de 2014 que permite a un atacante leer la memoria de un cliente o del servidor, permitiendo por ejemplo obtener las claves privadas de un servidor, lo que supondría que el atacante sería capaz de descifrar todos los datos de una conexión.

En otras ocasiones el problema puede incluso encontrarse en el cliente, como ocurre con otra vulnerabilidad de OpenSSL, encontrado en julio de 2015 y que permite al atacante hacerse pasar por una web segura. El fallo está en que la aplicación trate un certificado inválido como uno verídico, permitiendo al que esté monitorizando la red modificar los datos de la conexión. Este fallo ha preocupado sobre todo a las aplicaciones Android, ya que se basa en OpenSSL. El fallo ya ha sido parcheado.

Hasta el propio protocolo de cifrado puede verse comprometido. Es lo ocurrido con el ataque POODLE, que permitía a un atacante MITM leer ciertos datos que deberían estar protegidos, como por ejemplo las cookies. Este exploit main-in-the-middle fue anunciado en octubre de 2014, y ha significado la desactivación por parte de muchas aplicaciones y navegadores del protocolo SSL 3.0, uno de los más veteranos. Conseguía obtener los datos sin cifrar gracias al relleno ("padding") del algoritmo de cifrado por bloques de este protocolo.

Por estas y muchas más vulnerabilidades que existen, es necesario comprobar que el certificado proporcionado por el servidor al cliente es seguro y la conexión se realiza sin problemas. Esto ha motivado la idea de que en este proyecto se monte una arquitectura completa y pasarle varios tipos de pruebas.

[1] El borrador del protocolo ACME puede encontrarse en: https://letsencrypt.github.io/acme-spec/

2.2 Objetivos

El proyecto que nos concierne está pensado para estudiar cómo poner en marcha una arquitectura que incluya una autoridad de certificación y un respondedor OCSP junto con un servidor que se pondrá a disposición de un cliente, todo ello de forma segura.

En la autoridad se busca:

- Fácil instalación y puesta en marcha de dicha autoridad, con interfaz sencilla
- Generación y revocación de certificados
- Integración de solución para validación OCSP con la autoridad
- Comunicación cifrada con el servidor y comprobación de la integridad de éste.
    - Se basará en la validación como eje fundamental para comprobar que el nombre proporcionado corresponde al servidor que lo solicita. Para ello pedirá que coloque un fichero que proporcionará la autoridad para poder descargarlo y comprobar que es quien dice ser.

Se pretende que los administradores puedan:

- Solicitar y revocar certificados para un servidor concreto con interfaz sencilla
- Configuración automática de certificado en máquina del servidor
- Proceso transparente al administrador, por lo que éste sólo deberá tener permisos de superusuario en su sistema y el programa se encargará de realizar todo lo demás.

Además, se realizará una implementación de cliente sencillo para conexión segura con servidor a prueba de fallos, que sirva como base para cualquier otra aplicación más compleja, pero que mantenga principalmente la seguridad en la transferencia de sus datos. Se estudiarán los posibles ataques a realizar y se verá si el sistema es vulnerable.

Respecto al tema de la seguridad, se busca también que se cumplan los principios básicos:

- Confidencialidad: Asegura que los usuarios no autorizados no accedan a determinados datos.
- Integridad: La información no ha sido borrada, ni modificada ni copiada.
- Disponibilidad: Determinados recursos de un sistema estén disponibles a los usuarios autorizados siempre que lo requieran.

3 ESTADO DEL ARTE

La información es poder.

- Bill Gates -

El uso del cifrado en la vida diaria con la integración de los servicios online seguros en nuestros días parece un descubrimiento actual. Sin embargo, desde la Antigüedad se ha buscado la forma de que no se intercepten las comunicaciones. Este capítulo muestra por un lado una introducción a la criptografía, que es la base de toda comunicación cifrada, así como la idea principal de las soluciones PKI (infraestructuras de clave pública). Además estudiaremos cómo funciona el protocolo HTTPS.

3.1 Criptografía

La criptografía es el arte de escribir ocultando la información. En ocasiones suele confundirse con otros conceptos similares como son el criptoanálisis, que engloba las técnicas utilizadas para descubrir el significado de los mensajes, o la criptología, que es la ciencia que abarca el estudio de la criptografía y el criptoanálisis. Lo bueno que pueda ser un código criptográfico se mide por lo persistente que sea ante ataques de técnicas de criptoanálisis.

A lo largo de la historia todas las civilizaciones han buscado el modo de que las comunicaciones no fueran públicas. Los orígenes se pueden encontrar con los jeroglíficos egipcios, mientras que el primer dispositivo criptográfico se fecha en el año 400 a.C, "la Scitala", de procedencia espartana y que utilizaba la transposición mediante una vara de un cierto grosor donde se enrollaba una cinta de cuero o papiro y se escribía el mensaje. Al desenrollar la cinta quedaba una serie de letras sin sentido, que era entregado al mensajero. Para descifrarla era necesaria una vara de las mismas dimensiones que la original.

Después aparecieron otros métodos, como el de Polybios (siglo II a.C), primero en ser de sustitución basado en sustituir caracteres por un número o letra de una columna o fila de una tabla de correspondencia; o el método del César, primero que fue debidamente documentado que sustituía la letra a escribir por la tercera que le siguiera en el alfabeto (por ejemplo la A se sustituía por la D o la B por la E).

La cultura árabe no destacó por el uso de la criptografía, sino que la aportación que realizaron fue la invención del criptoanálisis. No será hasta el Renacimiento cuando aparecen nuevos métodos como el cifrado de disco de Alberti con una sustitución polialfabética que usa varios diccionarios, en 1466. Un siglo después Giovan Battista Belasso creó una nueva técnica basada en una clave que debe transcribirse letra a letra sobre el texto original. Cada letra del texto se cambia por la correspondiente en el alfabeto que empieza con la letra clave. Esto es conocido como la tabla de Vigenère.

La revolución de la criptografía llega en el siglo XX y la máquina Enigma. Inventada en 1919 por Arthur Scherbius, es una máquina criptográfica a rotor que los nazis pensaban que era indescifrable. Supuso un factor decisivo en la Segunda Guerra Mundial cuando una organización secreta británica, en la que participó Alan

Turing (considerado uno de los padres de la informática y de la inteligencia artificial), logró descubrir las claves de Enigma y desvelaron todos sus mensajes cifrados. Con ello se aceleró la derrota de Alemania.

Acabada la guerra, las nuevas tecnologías dieron paso a sistemas criptográficos más modernos. Las bases teóricas las establece Claude Shannon en 1948, sobre las que se apoyan prácticamente casi toda la criptografía actual.

Figura 3-1. Máquina Enigma

3.1.1 Sistemas simétricos o de clave secreta

Se utiliza la misma clave secreta tanto para cifrar como para descifrar. Es un método simple a nivel matemático y ha sido utilizado durante años. La seguridad se reduce únicamente a la seguridad de la clave. La potencia de esto se encuentra en la robustez del algoritmo de cifrado.

En la década de los 70, IBM junto con el apoyo del gobierno estadounidense desarrolló un sistema criptográfico denominado DES, que es producto de transposiciones y sustituciones. Pero esto tenía un grave problema, y es que el tamaño de su clave de 56 bits era bastante corto, lo que podía llegar a "romperse" en menos de 24 horas. Para corregir estos problemas, en 1998 se lanzó el 3DES, un algoritmo que hace un triple cifrado del DES. La gran diferencia que existe es que se utiliza una clave de 192 bits, aunque en realidad son 156 efectivos eliminando los bits de paridad. Este algoritmo se usa aún en tarjetas de crédito y otros métodos de pago electrónico. Sin embargo, por su diseño tanto DES como 3DES son algoritmos lentos, por lo que su uso está desapareciendo lentamente.

En 2001 se anunció un nuevo sistema que perdura hasta nuestros días, el AES. Tiene un tamaño de bloque de 128 bits y tamaños de llaves de 128, 192 o 256 bits. AES puede llegar a ser hasta 6 veces más rápido que 3DES y mucho más seguro, hasta tal punto de denominarse inquebrantable. No ha sido hasta 2011 cuando un grupo de investigadores han descubierto la manera de romper AES, pero esto no significa que haya sido posible conseguirlo, ya que se con la tecnología actual no es posible porque se tardarían 3 millones de años.

Los protocolos SSL/TLS hacen uso de estos cifrados simétricos (dentro de su sistema híbrido) en sus numerosas versiones. A pesar de que se siguen utilizando los cifrados mencionados, también existen otros como IDEA (propuesto también como alternativa a DES y utilizado en las primeras versiones de PGP), RC4 (cifrado de flujo que actualmente se considera bastante inseguro), SEED o Camellia.

3.1.2 Sistemas asimétricos o de clave pública

El mayor problema que presentan los sistemas simétricos es el de la distribución de claves. Por ello la revolución llegó con este nuevo cifrado, llamado asimétrico.

Existen dos claves, se cifra con una y se descifra con otra. El objetivo es que cualquiera pudiera cifrar el mensaje, usando la clave pública, pero sólo el receptor del mensaje pudiera descifrarlo, usando la clave privada. Cada persona tendrá dos claves, una pública que podrá distribuir a todo el mundo, y otra privada que deberá guardar para que nadie tenga acceso a ella. Ya no es necesario que ambos lados de la comunicación se pongan de acuerdo en la clave que van a usar, sólo se requiere que el destinatario tenga una copia de la clave pública de quien envía el mensaje. Con esto se garantiza confidencialidad, por lo que permite aplicaciones como la autenticación y la firma digital.

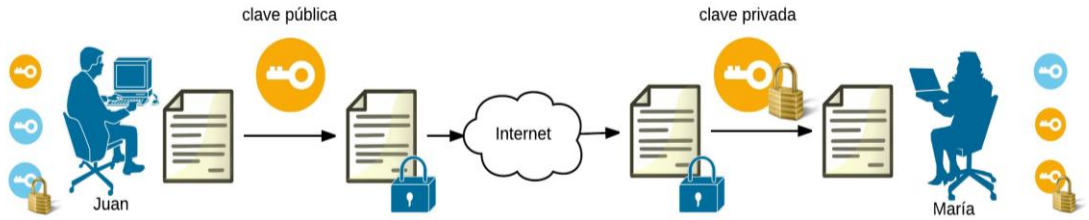

Figura 3-2. Ejemplo cifrado de mensaje con clave pública

En el ejemplo de la figura, Juan quiere enviar un mensaje cifrado a María con este sistema. Para ello cifra el mensaje con la clave pública de María antes de enviarlo. Una vez que le llega a María, ésta descifrará con su clave privada, manteniéndose en esta conexión la confidencialidad.

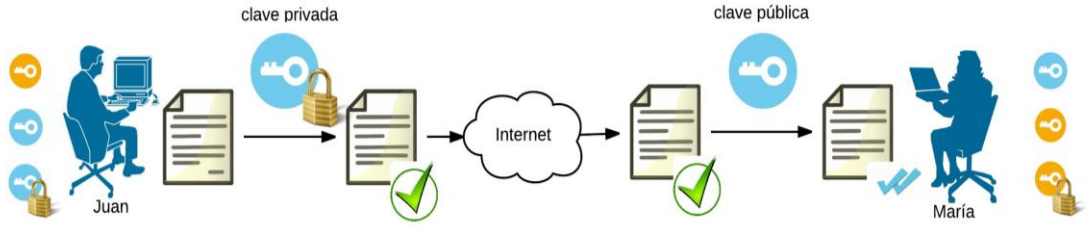

Figura 3-3. Ejemplo de firma digital con clave pública

Si se usa la clave privada para cifrar el mensaje, cualquiera puede descifrarlo usando la clave pública. Esto es la idea principal de la firma digital. Para el caso del ejemplo en el que Juan firma el mensaje con su clave privada, cuando éste llega a María tendrá que comprobar su autenticidad (integridad, no repudio) con la clave pública proporcionada por Juan, pudiendo leer el mensaje con total seguridad.

Los sistemas asimétricos se clasifican según el método utilizado para garantizar su seguridad, sobre un canal de comunicaciones no seguro. El primero en aparecer es el de RSA. Este método se basa en que no hay ninguna forma eficaz para factorizar[2] un número obtenido de la multiplicación de dos números primos. Se necesitarían cien años para calcularlo. Actualmente estos primos son del orden de 10^{200} y se prevé que se aumente con el incremento de la capacidad de cálculo de los ordenadores.

Procedimiento 3–1. *Proceso de la obtención del par de claves mediante RSA*

- *Se calculan dos números primos p y q (secretos) y se realiza el producto n (público).*
- *Se hace el siguiente producto:*

$$\varphi(n) = (p-1) * (q-1) \qquad (3\text{–}1)$$

- *Se busca un número, e, sin múltiplos comunes a* $\varphi(n)$.
- *Calculamos el número d,* $1 < d < \varphi(n)$*,usando la siguiente fórmula:*

[2] Factorizar es escribir un número en forma de productos.

$$e * d \equiv 1\ MOD(\varphi(n)) \qquad (3\text{–}2)$$

Como solución, n y e forman la clave pública, mientras que d es la clave privada. Para evitar que la clave privada pueda ser descubierta se eliminan p, q y $\varphi(n)$.

Otros métodos criptográficos de clave pública son los denominados de logaritmo discreto o de Diffie-Hellman. Estos se basan en la gran dificultad de calcular logaritmos discretos de un cuerpo finito. La idea es que cada extremo de la comunicación elija un número público y otro secreto. Usando una fórmula exponencial, cada uno realizan una serie de operaciones con los dos números públicos y el secreto. Posteriormente se intercambian los resultados públicamente. Cada uno de ellos usan por separado otra fórmula matemática con la que al final llegan ambos a un mismo resultado que será la clave compartida.

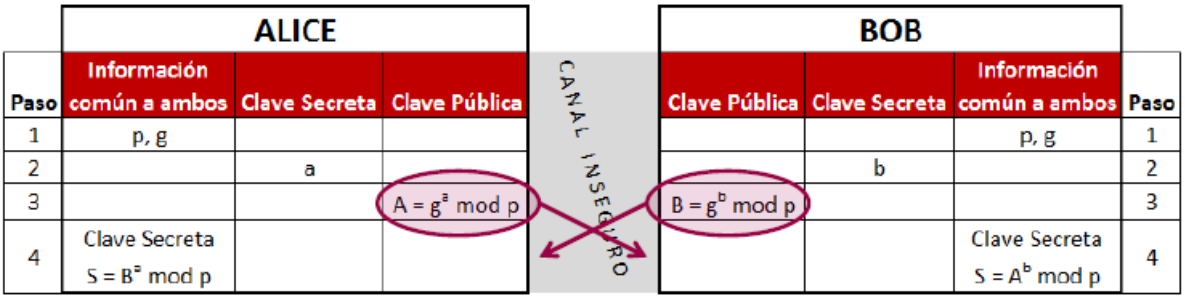

Figura 3-4. Algoritmo discreto de Diffie-Hellman

Fuente: Apuntes de Seguridad, 3º GITT

El problema es que si p es un número primo muy grande, la solución del algoritmo discreto no se puede tratar con la tecnología actual. Como observamos en la figura, si alguien quisiera descifrar el secreto, S, tendría que calcular:

$$a = \log discreto_p\ (A)$$

$$b = \log discreto_p\ (B)$$

Por último, otro algoritmo para clave pública es el de curva elíptica, el cual se basa en el anterior, logaritmo discreto, pero en este caso se usan coordenadas cartesianas en lugar de números enteros. Sobre estos puntos se define un grupo con las propiedades conmutativa, asociativa, elemento neutro y elemento simétrico, y las operaciones de suma y multiplicación.

3.1.2.1 Firma digital

La firma digital es un proceso de cifrado que permite a cualquiera comprobar la autenticidad de los datos cifrados. No debe confundirse con la firma electrónica, otro concepto bastante utilizado en estos tiempos y que son unos datos electrónicos que acompañan la información e identifican al firmante (con el mismo objetivo de la firma manuscrita). La firma electrónica tiene valor jurídico y permite dar fe de un acto con la voluntad del que firma, así como identificarle. Ambos conceptos tienen su relación, ya que la firma digital es el procedimiento que permite que la firma electrónica pueda ser atribuida a una persona con seguridad.

Pero aquí nos concierne el hecho de que la firma digital nos va a permitir identificar a una máquina, por ejemplo un servidor, manteniendo la integridad y la autenticidad del origen (no repudio).

Se basa en los sistemas criptográficos de clave pública, donde se firma el mensaje con la clave privada y cualquiera puede descifrarlo con la clave pública (ver Figura 3-3). Suelen usarse diferentes algoritmos para este proceso, como RSA y DSA, siendo este último un algoritmo basado en logaritmos discretos diseñado para firmas digitales por el gobierno de Estados Unidos, propuesto para el Estándar de Firma Digital (DSS). DSA es una variante del algoritmo de firmas ElGamal, y sirve como su nombre indica sólo sirve para firmar y no para cifrar. Una desventaja de DSA es que necesita más tiempo de cómputo que RSA, haciendo que sólo se

puedan firmar mensajes de pequeño tamaño.

El empleo de logaritmos que requieren mensajes pequeños provoca la aparición de las funciones resumen o hash. Generan un "extracto" siempre del mismo tamaño de cualquier archivo. Con esto, el proceso de firma puede limitarse a la firma del resumen. A un mensaje se le genera su hash y a éste se le aplica un cifrado con la clave privada, transmitiendo tanto el mensaje como el hash cifrado. Si al mensaje recibido se le aplica el mismo hash inicial, se debe obtener lo mismo que al descifrar con la clave pública el hash cifrado recibido.

Las propiedades esenciales que debe cumplir toda función hash son:

- Unidireccional: Existe una relación de un único sentido desde el mensaje original hacia el hash.
- Resistencia a colisión: No se pueden encontrar dos mensajes con el mismo resumen.

Los algoritmos de hash más usados son el MD5 (de 128 bits, RFC 1321) y SHA-1 (de 160 bits, aunque también existen otros como SHA-256 y SHA-512). El problema de MD5 es que han aparecido una serie de vulnerabilidades, por lo que su uso se ha reducido a la validación de la integridad de ficheros propios cuyo contenido conocemos. Para firmas o certificados se recomienda usar algoritmos de la familia SHA. Sin embargo SHA-1 también ha sido ya comprometido por un grupo de investigadores chinos en 2005. Aunque el NIST afirma que sobre el contexto de las aplicaciones es difícil llevar a cabo este ataque en la práctica, existe el llamado SHA-2 (incluyendo los SHA-256 y SHA-512) que se consideran inmunes y son utilizados en la mayoría de aplicaciones de hoy en día como SSL, TLS, PGP o SSH. Cabe destacar que se podrían utilizar otro tipo de funciones hash como Tiger (de 192 bits), de los creadores de AES.

3.1.3 Sistemas híbridos

Se caracterizan por usar tanto sistemas simétricos como asimétricos. Se cifra el mensaje con la clave secreta y se comparte la clave secreta que ha sido cifrada con la clave pública. Cuando llega al destinatario, éste descifra la clave secreta con su clave privada, y entonces podrá descifrar el mensaje.

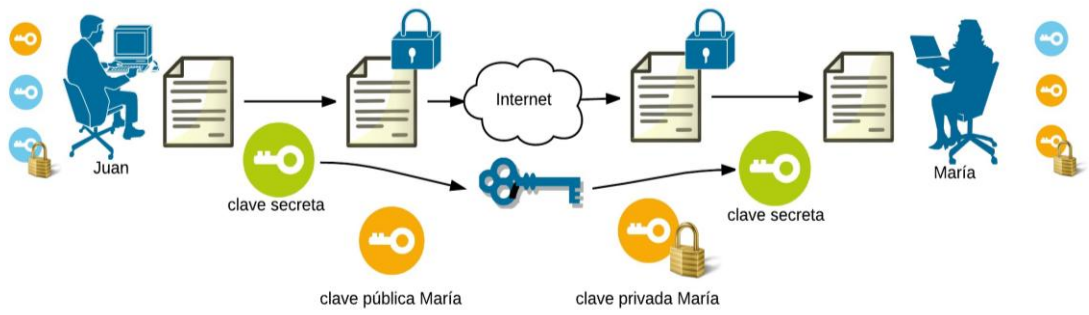

Figura 3-5. Ejemplo de cifrado con sistema híbrido

La principal ventaja de usar este tipo de sistemas es que los cifrados simétricos son más rápidos que los asimétricos. Además permite la utilización de una clave secreta diferente para cada sesión, debido a que el canal por el que se va a compartir no va a ser seguro.

Ejemplos de criptografía híbrida los podemos encontrar en los sistemas PGP o su equivalente GPG, que son herramientas de cifrado y firmas digitales, cuya finalidad es proteger la información distribuida. Lo mismo ocurre con SSH, que da nombre tanto al protocolo como al programa que sirve para acceder a máquinas remotas a través de la red.

Los protocolos SSL y TLS también están considerados sistemas híbridos porque emplean tanto clave secreta para cifrar los datos como clave pública para el cifrado de esta clave secreta. Más adelante se verán en detalle, ya que serán la base de nuestro trabajo.

3.1.4 Sistemas cuánticos

Las leyes de la mecánica cuántica llevadas a la informática suponen un nuevo paradigma en la computación. Un ordenador cuántico será capaz de ejecutar múltiples de operaciones en un mismo instante. Cuando esto ocurra, acabaría con la criptografía tal y como la conocemos hoy en día, porque todos los sistemas se basan en operaciones matemáticas imposibles de realizar con la tecnología actual, pero que sí podrían calcularse con la llegada de los ordenadores cuánticos.

Con la ruptura de todos los mecanismos de cifrado de la criptografía clásica, es necesario basarse también en las leyes cuánticas para usarla en la criptografía. Mientras que con los métodos actuales un intruso tendría que resolver complejas operaciones matemáticas para descifrar la información, con la criptografía cuántica tendría que violar las leyes de la mecánica cuántica para leer los datos, cosa que a día de hoy se desconoce.

El funcionamiento podemos verlo con un ejemplo. María quiere mandarle a Juan un mensaje a través de un canal cuántico como puede ser la fibra óptica. María envía una serie de bits codificados en 'qubits' a Juan. Una vez finalizada la transmisión de éstos, emplean un canal clásico para comunicarse, con la clave secreta compartida para descifrar el mensaje. Si un espía intenta leer la comunicación se encontrará con un problema: si mide un sistema cuántico, lo perturba, lo que significa que los 'qubits' enviados serían modificados. Cuando María y Juan empiecen a hablar inmediatamente se darán cuenta de que hay un intruso debido a las perturbaciones.

Pero la criptografía cuántica no es un hecho del futuro. En 1984 se publicó el primer protocolo y en 2001 se creó la primera empresa que vendía equipos de distribución de claves cuánticas, ID Quantique. Aunque todavía no ha llegado a los hogares, se espera que esta tecnología sea la base de la criptografía del futuro.

3.2 Infraestructura de clave pública (PKI)

Hemos visto cómo los mecanismos de criptografía de clave pública han conseguido importantes objetivos en la seguridad. La ventaja es que no se requiere el intercambio de una clave secreta. Pero en esta criptografía asimétrica, las claves también deben ser gestionadas correctamente. La gestión eficiente y segura de los pares de claves durante todo su ciclo de vida es el objetivo principal de las infraestructuras de clave pública.

El ciclo de vida comienza con la fase de generación de las claves, continuando con la fase en la que las claves son usadas. En este paso, la clave privada es utilizada para descifrar o para firmar información. También, los usuarios tienen acceso a las claves públicas de otros usuarios para cifrar o verificar mensajes. En la última etapa de este ciclo de vida, el par de claves se vuelve inválido, como consecuencia por ejemplo de la expiración de su periodo de validez.

Una PKI es "una combinación de hardware y software, políticas y procedimientos de seguridad que permiten la ejecución con garantías de operaciones criptográficas como el cifrado, la firma digital o el no repudio de transacciones electrónicas", según la definición dada por Wikipedia.

3.2.1 Certificados

La principal tarea de una PKI es proporcionar pruebas que verifiquen la autenticidad de las claves públicas. Por ejemplo si María quiere verificar una firma digital de Juan, María necesita tener la convicción de que la clave pública es auténtica. Los certificados son estructuras de datos que asocian las claves públicas a las entidades que están firmados por una tercera parte. Si María tiene un certificado con la clave pública firmado por una tercera parte de confianza, entonces si verifica la firma del certificado puede dar a María la autenticidad de que la clave pública que va a utilizar es la de Juan. De esta forma, los certificados reducen tener que confiar en todas las claves públicas de las entidades a la confianza de algunas autoridades.

Los contenidos mínimos de un certificado son:

- Nombre o seudónimo del sujeto a quien pertenece la clave pública
- La clave pública que está unida a la entidad
- El algoritmo criptográfico que la clave pública usa

- El número de serie del certificado
- El periodo de validación del certificado
- El nombre del emisor que firma el certificado
- Restricciones o extensiones de uso de la clave pública en el certificado

Normalmente el emisor del certificado es diferente al sujeto. Sin embargo, hay certificados auto-firmados en los que el sujeto y el emisor son el mismo. Los certificados pueden ser emitidos a título personal, o identificando a un servidor, un software determinado o a una entidad.

El estándar de certificados más importante es el X.509, especificado en la recomendación X.509 de la ITU-T (1). Los certificados son especificados en el lenguaje ASN.1. La estructura de los certificados, obtenida de la recomendación indicada es la siguiente:

```
Certificate ::= SIGNED{TBSCertificate}
TBSCertificate ::= SEQUENCE {
 version [0] Version DEFAULT v1,
 serialNumber CertificateSerialNumber,
 signature AlgorithmIdentifier{{SupportedAlgorithms}},
 issuer Name,
 validity Validity,
 subject Name,
 subjectPublicKeyInfo SubjectPublicKeyInfo,
 issuerUniqueIdentifier [1] IMPLICIT UniqueIdentifier OPTIONAL,
 ...,
 [[2: -- if present, version shall be v2 or v3
 subjectUniqueIdentifier [2] IMPLICIT UniqueIdentifier OPTIONAL]],
 [[3: -- if present, version shall be v2 or v3
 extensions [3] Extensions OPTIONAL]]
 -- If present, version shall be v3]]
}
```

Esta estructura contiene los principales datos del certificado. Pero como se puede observar, permite añadir una serie de extensiones. Entre estas extensiones cabe destacar *Subject Alternative Name*, que permite asociar un nombre alternativo al indicado en el sujeto. Esto es útil y lo vamos a utilizar en el proyecto cuando se solicite un certificado para un sujeto que sea una dirección IP, debido a que el cliente verifica que el nombre del sujeto coincide con la dirección IP, y al ser ya una IP lo busca en esta extensión.

Así mismo, usaremos también otra extensión, *Authority Info Access* para indicar la dirección donde se realizará la validación OCSP.

Los estados que podemos encontrar un certificado son:

- Emisión: inicio de su vigencia
- Expiración: finalización del periodo de validez
- Revocación: la clave privada asociada al certificado ha sido comprometida o se han cambiado los datos asociados al certificado
- Suspensión: mismas actuaciones que revocación, salvo que es reversible

3.2.2 Componentes de una PKI

Como hemos visto en el apartado anterior, un certificado debe estar firmado por una entidad de confianza para que el cliente pueda verificar su autenticidad. Pero esto no es suficiente y en muchos casos son necesarios más componentes que realicen diferentes funcionalidades. Algunos de éstos pueden estar asociados en uno solo e incluso otros no serán estrictamente necesarios, como será el caso del escenario que montaremos en el trabajo, pero ahora vamos a verlos todos por separado para comprender mejor el contexto en el que nos movemos.

3.2.2.1 Autoridad de certificación (CA)

Es el componente más importante. Es una entidad fiable que garantiza de una forma segura la identidad de un usuario o servicio asociada a una clave pública.

Es la encargada de emitir y revocar certificados, según las peticiones que reciba. Consulta a la Autoridad de Registro para ver si acepta o rechaza la petición de un certificado, aunque en otras ocasiones también puede verificar esto por sí mismo.

Las CA disponen de sus propios certificados públicos, en los que sus claves privadas son las que se usan para firmar los certificados que genera. Un certificado de CA puede estar auto-firmado si no tiene una CA superior que se lo emita. En ese caso, el certificado se denominará certificado CA raíz, considerado como el elemento inicial de cualquier estructura de certificación. Esta jerarquía de certificación comienza con la CA auto-firmada y de la que pueden partir otras CA que pueden firmar certificados para entidades finales.

Una de las razones por las que se establece la confianza en una CA es por esta jerarquía de certificación. Los usuarios finales sólo deberán instalar el certificado raíz de la CA en la que se quiere confiar. A partir de ahí, cualquier certificado firmado por dicha autoridad puede ser validado, porque contiene la clave pública con la que verificar la firma que lleva el certificado.

Además, una Autoridad de Certificación también gestiona las Listas de Revocación de Certificados (CRLs). Estas listas contienen los certificados que han sido revocados, por lo que es competencia de la autoridad la de actualizarlas y publicarlas. Si la CA contiene muchos certificados, puede resultar tedioso para los usuarios descargarse largas CRLs, por lo que suelen usarse Autoridades de Validación externas.

3.2.2.2 Autoridad de Registro (RA)

Gestiona el registro de usuarios y las peticiones o revocaciones de certificados, así como los certificados devueltos como respuesta a esas peticiones. Autorizan la asociación entre una clave pública y el sujeto de un certificado, indicando a la CA si debe emitir o no un certificado.

Esta comprobación puede realizarla directamente la CA, sin tener que recurrir a esta entidad externa.

Podemos encontrar dos métodos de registro. Por un lado el registro presencial, en el cual el usuario se persona en la Autoridad de Registro y le entrega toda la documentación necesaria. Si la RA aprueba la solicitud, envía los datos a la CA para que emita un certificado. Una vez emitido, la RA proporciona el certificado al usuario.

Por otro lado está el registro remoto, en el que el usuario hace al comienzo un pre-registro en la Autoridad de Certificación. Tras esto, el usuario se persona telemáticamente ante la RA, entregando toda la documentación. El resto del proceso es similar al anterior, comprobando los datos la Autoridad de Registro y enviándolos a la CA si los aprueba, y posteriormente devuelve el certificado generado por la CA al usuario.

3.2.2.3 Entidades finales

Son aquellos que poseen un par de claves (pública y privada) y un certificado asociado a su clave pública, tales como usuarios o servidores[3]. Utilizan aplicaciones que hacen uso de la tecnología de esta infraestructura, como validar firmas digitales o cifrar documentos. Una vez obtenido el certificado, deberán configurarlo en función de la aplicación a utilizar.

3.2.2.4 Repositorio o directorio

Los directorios proporcionan almacenamiento y distribución tanto de certificados como de listas de revocación (CRLs). Cuando la CA emite un certificado o CRL, lo envía al repositorio, aunque también guarda una copia en su base de datos local.

Para el acceso a los directorios suele utilizarse el protocolo LDAP.

[3] Un servidor web es considerado una 'entidad final' cuando obtiene un certificado y lo usa para asegurar su identidad en la red.

3.2.2.5 Autoridad de Validación (VA)

Proporciona información online acerca del estado de un certificado. Puede dar dos tipos de servicio, uno basado en CRLs, en las que el usuario se las descarga y las interpreta de forma autónoma, o a través del protocolo OCSP.

Para el caso de OCSP, los usuarios que quieran saber el estado de un certificado, sólo tienen que realizar una petición OCSP contra la autoridad de validación para obtenerlo. Cada vez que se modifica el estado de un certificado, la CA actualiza la información de la VA, que tendrá información en tiempo real.

3.2.2.6 Componentes opcionales

Mencionar otros componentes que pueden ser incluidos en la PKI, como puede ser la Autoridad de Sellado de Tiempos (TSA), que permite obtener una prueba de que un determinado dato existía en una fecha concreta; o la Autoridad de Recuperación de Claves (KA, Key Archive), que almacena y recupera archivos de clave PKCS#12 y contraseñas generados por la CA.

3.2.3 Funcionamiento de una PKI

Una vez analizados los componentes principales de la infraestructura, vamos a ver un ejemplo de funcionamiento de ésta.

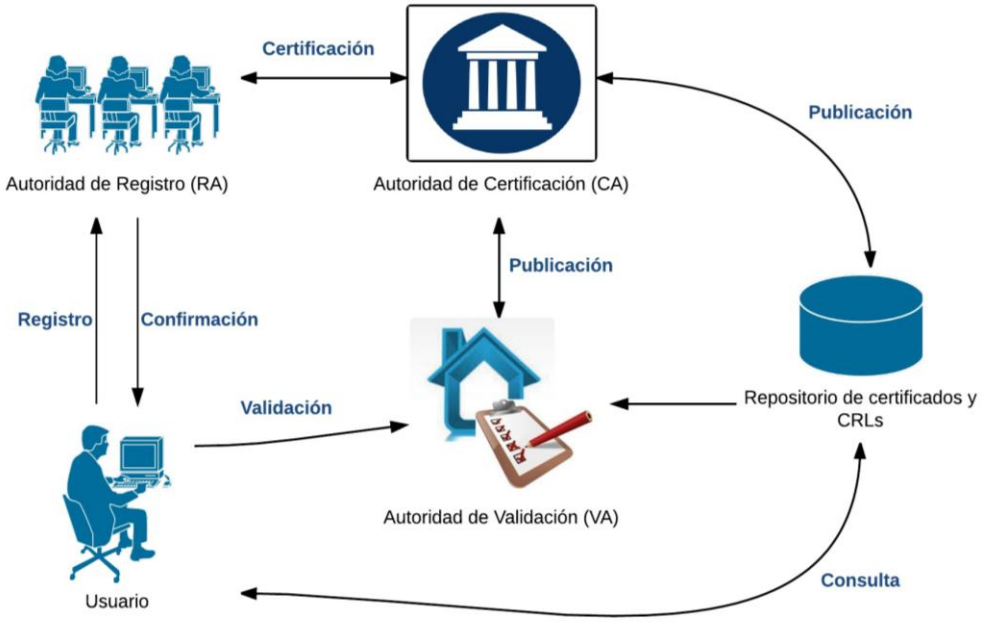

Figura 3-6. Esquema de funcionamiento de una PKI

- En primer lugar el usuario se registra para obtener un certificado.
- La Autoridad de Registro inicia la captura de información y activa la generación de claves.
- El usuario devuelve a la RA la clave pública y la información necesaria.
- Una vez comprobada, se hace una petición a la CA.
- La Autoridad de Certificación firma la petición válida y envía el certificado que acaba de generar a la RA.
- La RA entrega al usuario el certificado firmado.
- La CA publicará los certificados tanto en el repositorio de certificados como en la Autoridad de Validación, para que los usuarios puedan consultar su estado.

3.2.4 Validación de certificados

El periodo de validez de un certificado puede ser largo. Sin embargo, puede ocurrir que durante este periodo haya que invalidarlo por ejemplo porque la clave privada haya sido comprometida. Este proceso de invalidar antes de que expire se denomina revocación.

Existen varios métodos para comprobar la validez del certificado, es decir, saber en el estado en el que se encuentra:

- Consulta de la lista de certificados si se encuentra en un repositorio (o directorio LDAP)
- Consulta de la lista de certificados revocados (CRLs)
- Consulta por protocolo OCSP
- Consulta mediante sistemas adhoc (servicio web que responde a las consultas consultando a su vez a las CRL o a una Autoridad de Validación OCSP, no es un estándar)

Vamos a centrarnos en los dos más empleados, que son las CRLs y el protocolo OCSP.

3.2.4.1 Lista de certificados revocados (CRL)

Una CRL es una lista que contiene los números de serie de los certificados revocados que está firmada digitalmente por una CA para probar su autenticidad. Normalmente suele estar actualizada por la CA, es decir, cuando cambia el estado de un certificado y se convierte en revocado, éste es agregado a la CRL.

Hay CRLs directas que sólo contienen certificados de un emisor y están firmados por ese emisor, mientras que por otro lado existen las indirectas que pueden contener certificados de varios emisores (pueden tener listas de diferentes CA) y están firmadas por el llamado emisor de la CRL.

Los usuarios que quieran saber el estado de un certificado deben descargarse la CRL desde la CA y verificar su firma digital. Entonces pueden comprobar si el certificado en el que están interesados se encuentra incluido en la lista, y por tanto, revocado.

El problema que tienen es que cuando una CA tiene muchos certificados, puede tener una CRL de gran tamaño, lo cual puede resultar molesto para el usuario que debe descargarse dicha CRL.

Las CRLs están especificadas en el estándar X.509, y su estructura de datos en ASN.1 es la siguiente:

```
CertificateList ::= SIGNED{CertificateListContent}
CertificateListContent ::= SEQUENCE {
 version Version OPTIONAL,
 -- if present, version shall be v2
 signature AlgorithmIdentifier{{SupportedAlgorithms}},
 issuer Name,
 thisUpdate Time,
 nextUpdate Time OPTIONAL,
 revokedCertificates SEQUENCE OF SEQUENCE {
 serialNumber CertificateSerialNumber,
 revocationDate Time,
 crlEntryExtensions Extensions OPTIONAL,
 ...} OPTIONAL,
 ...,
 ...,
 crlExtensions [0] Extensions OPTIONAL }
```

Las extensiones que pueden agregarse a una CRL pueden ser: *Authority Key Identifier, Issuer Alternative Name, CRL Number, Delta CRL Indicator, Issuing Distribution Point*. Para una entrada en la CRL se puede añadir las *extensiones Reason Code, Hold Instruction Code, Invalidity Date y Certificate Issuer*. Para mayor información de éstas, consultar la recomendación (1).

3.2.4.2 OCSP

El problema de una CRL llega cuando alcanza un gran tamaño, provocando que se consuma bastante tiempo en su descarga y necesitando un gran espacio de disco para su almacenamiento, el cual no siempre tiene por qué estar disponible en las máquinas de los clientes, por ejemplo en los dispositivos móviles.

Es por ello que se inventó OCSP. Es un protocolo que permite a los usuarios solicitar el estado de revocación de un certificado a un servidor OCSP. Este método tiene varias ventajas respecto a las CRLs. En primer lugar, la información siempre va a estar actualizada, ya que se trata de un servicio online, aunque esta ventaja puede verse afectada si el servidor OCSP obtiene la información en función de las CRLs de la autoridad de certificación. Otra ventaja es que no se requiere mucho espacio de almacenamiento, al tratarse de un servicio externo. OCSP está especificado en la RFC 2560 (2). Son numerosas las PKI que incluyen este servidor, y muchos clientes que lo usan, como puede ser el navegador Firefox, que permite la configuración para que se valide un certificado mediante OCSP antes de acceder al sitio web.

El funcionamiento se basa en que un cliente OCSP realiza una petición al servidor OCSP (también conocido como respondedor) del estado de un certificado. Para cada certificado, la petición incluye el número de serie del certificado, el valor hash del emisor, y el valor hash de la clave pública del emisor. Esta información identifica al certificado de forma unívoca. La estructura de la petición en ASN.1 obtenida de su recomendación la podemos ver a continuación:

```
OCSPRequest     ::=     SEQUENCE {
       tbsRequest                  TBSRequest,
       optionalSignature   [0]     EXPLICIT Signature OPTIONAL }

TBSRequest      ::=     SEQUENCE {
       version             [0]     EXPLICIT Version DEFAULT v1,
       requestorName       [1]     EXPLICIT GeneralName OPTIONAL,
       requestList                 SEQUENCE OF Request,
       requestExtensions   [2]     EXPLICIT Extensions OPTIONAL }

Signature       ::=     SEQUENCE {
       signatureAlgorithm      AlgorithmIdentifier,
       signature               BIT STRING,
       certs               [0] EXPLICIT SEQUENCE OF Certificate
   OPTIONAL}

Version         ::=             INTEGER  {  v1(0) }

Request         ::=     SEQUENCE {
       reqCert                     CertID,
       singleRequestExtensions     [0] EXPLICIT Extensions OPTIONAL }

CertID          ::=     SEQUENCE {
       hashAlgorithm       AlgorithmIdentifier,
       issuerNameHash      OCTET STRING, -- Hash of issuer's DN
       issuerKeyHash       OCTET STRING, -- Hash of issuer's public key
       serialNumber        CertificateSerialNumber }
```

Al igual que las CRLs, el protocolo OCSP permite que se añadan algunas extensiones en la petición, que pueden ser consultadas en la RFC (2).

Cuando esta petición llega al servidor OCSP, éste devuelve una respuesta con tres posibles estados (*revoked, good y unknown*). Además incluye los campos *thisUpdate* y *nextUpdate* que incluyen el tiempo de actualización en el que se comprueba el certificado. Estos campos los podemos observar en la definición ASN.1 de la recomendación:

```
OCSPResponse ::= SEQUENCE {
      responseStatus         OCSPResponseStatus,
      responseBytes          [0] EXPLICIT ResponseBytes OPTIONAL }
```

```
BasicOCSPResponse       ::= SEQUENCE {
      tbsResponseData      ResponseData,
      signatureAlgorithm   AlgorithmIdentifier,
      signature            BIT STRING,
      certs            [0] EXPLICIT SEQUENCE OF Certificate OPTIONAL }

ResponseData ::= SEQUENCE {
      version              [0] EXPLICIT Version DEFAULT v1,
      responderID              ResponderID,
      producedAt               GeneralizedTime,
      responses                SEQUENCE OF SingleResponse,
      responseExtensions   [1] EXPLICIT Extensions OPTIONAL }

SingleResponse ::= SEQUENCE {
      certID                       CertID,
      certStatus                   CertStatus,
      thisUpdate                   GeneralizedTime,
      nextUpdate         [0]       EXPLICIT GeneralizedTime OPTIONAL,
      singleExtensions   [1]       EXPLICIT Extensions OPTIONAL }

CertStatus ::= CHOICE {
      good        [0]     IMPLICIT NULL,
      revoked     [1]     IMPLICIT RevokedInfo,
      unknown     [2]     IMPLICIT UnknownInfo }

RevokedInfo ::= SEQUENCE {
      revocationTime              GeneralizedTime,
      revocationReason    [0]     EXPLICIT CRLReason OPTIONAL }
```

Cuando se diseña un servidor OCSP, hay que decidir cómo obtiene la información de revocación. Una posibilidad es obtener la información de una CRL. Ésta va a ser la opción que usa el servidor utilizado en el trabajo, pero que tiene el inconveniente de no tener la información totalmente actualizada. Sin embargo, sigue siendo mucho mejor trabajar con la información de un OCSP que es más pequeña que tener que descargar la CRL directamente. Una alternativa a esto puede ser enviar la información de un certificado individual directamente al servidor OCSP.

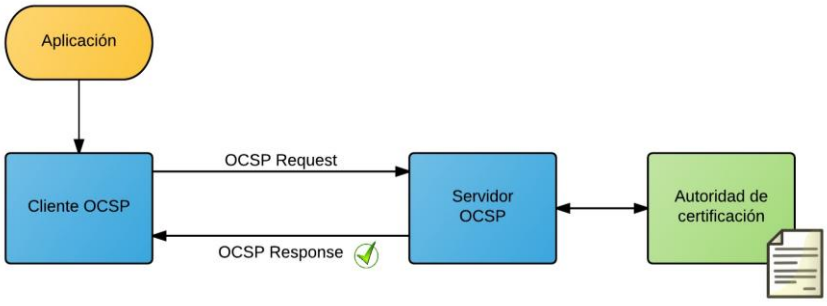

Figura 3-7. Esquema de funcionamiento de OCSP

En la figura anterior se ha representado el funcionamiento tal y como lo vamos a implementar en el proyecto, es decir, tomando la información el servidor OCSP directamente de la CRL generada por la CA.

3.2.5 PKI en la actualidad

3.2.5.1 Estándares

Existen multitud de estándares referentes a las Infraestructuras de Clave Pública. Es por ello que sólo vamos a mencionar algunos de ellos.

Por un lado, destacar el estándar PKIX perteneciente al IETF, que desarrolló RFCs a partir del estándar X.509 de la ITU-T acerca de los certificados, integrándolos en las Infraestructuras de Clave Pública para Internet.

También encontramos el conjunto de estándares PKCS, publicado por los laboratorios de RSA, y que mostramos en la siguiente tabla:

Tabla 3–1. Estándares PKCS

PKCS	Nombre
PKCS#1, #2, #4	Estándar criptográfico RSA
PKCS#3	Estándar de intercambio de claves Diffie-Hellman
PKCS#5	Estándar de cifrado basado en contraseñas
PKCS#6	Estándar de cifrado de certificados extendidos
PKCS#7	Estándar de sintaxis del mensaje criptográfico
PKCS#8	Estándar de sintaxis de información de la clave privada
PKCS#9	Tipos de atributos seleccionados
PKCS#10	Estándar de solicitud de certificación
PKCS#11	Estándar de interfaz de dispositivo criptográfico
PKCS#12	Estándar de sintaxis de intercambio de información de contraseñas
PKCS#13	Estándar de criptografía de curva elíptica
PKCS#14	Generación de números pseudo-aleatorios (sin documentación)
PKCS#15	Estándar de formato de información de dispositivo criptográfico

Cabe mencionar también los estándares desarrollados en Europa por la ETSI (*European Telecommunications Standards Institute*) acerca de las políticas para autoridades de certificación[4], así como la norma ISO 17799 que es un código de buenas prácticas para la Gestión de la Seguridad de la Información.

El marco legal en España se basa en la Directiva Europea 1993/93/CE en la que se establece un marco comunitario para la firma electrónica. En 2003 se saca en España la ley 59/2003 en la que regula la firma electrónica y la prestación de servicios de certificación. Como ya se mencionó, la firma electrónica tiene una relación directa con la firma digital y las PKI, por eso en esta ley se engloban.

3.2.5.2 Proveedores

Existe un amplio abanico de alternativas de proveedores de PKI, proporcionando una solución completa.

Comenzamos por la iniciativa de la Fábrica Nacional de Moneda y Timbre (FNMT), el proyecto CERES, con la que el gobierno español intenta acercar el servicio de la firma electrónica y los certificados a los ciudadanos. Ofrecen servicios tanto de las PKI como de verificación de firmas digitales, recuperador de claves y emisión de certificados.

Por otro lado, la PKI ofrecida por ACE (Agencia de Certificación Española), creada por el grupo Telefónica, y cuya actividad se basa en la distribución y comercialización de los servicios de certificación de Symantec.

La agencia ANF también proporciona servicios de autoridad de certificación, dentro de su PKI ANF AC,

[4] ETSI TS 102 042: Policy Requirements for certification authorities issuing public key certificates
ETSI TS 101 456: Policy Requirements for certification authorities issuing qualified certificates

siendo la primera CA en España que quedó acreditada en el año 2000.

Otros proveedores que ofrecen una solución completa de PKI son Safelayer, cuya solución es KeyOne; y Entrust que ofrece una solución tanto para empresas como para organismos.

Como soluciones libres, destacar OpenCA, que proporciona una interfaz web para usar los servicios de la PKI. De esta solución será el servidor OCSP que usaremos en nuestra implementación, y que detallaremos en el siguiente capítulo. Por otro lado, EJBC es una solución en Java que ofrece una infraestructura completa.

3.3 HTTPS

Una vez analizados los mecanismos criptográficos y la infraestructura necesaria para obtener los certificados, seguro que nos hemos llegado a preguntar ¿y todo esto cuándo lo vamos a utilizar? Si queremos transferir información segura necesitamos un protocolo que sea capaz de enviar los datos aunando todo lo mencionado anteriormente para lograr que todo esté cifrado.

HTTPS es un protocolo de aplicación que se basa en HTTP para la transferencia de datos de hipertexto de forma segura, utilizando para ello los protocolos SSL y TLS. Este protocolo será el que utilizaremos para la transferencia de información con seguridad en el proyecto.

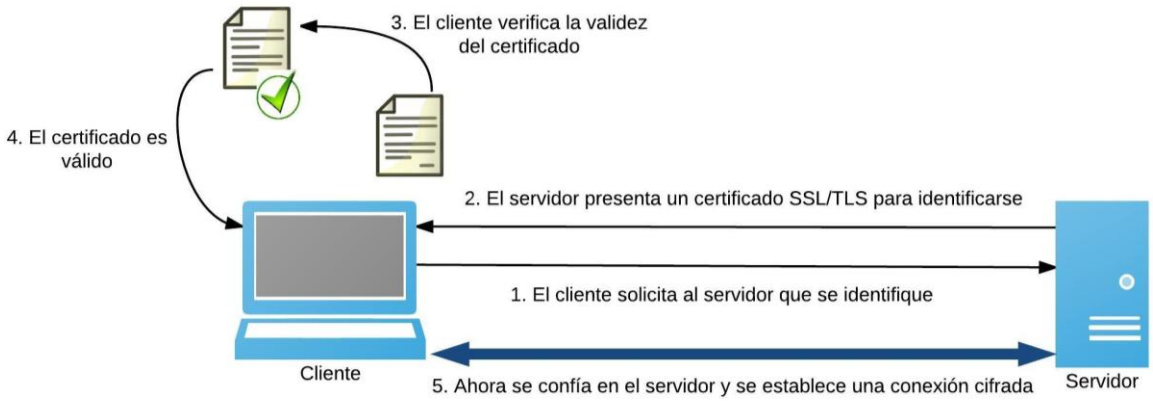

Figura 3-8. Esquema de funcionamiento HTTPS

3.3.1 HTTP

Conocido por todos, este protocolo es el empleado en las transacciones a través de la web. Desarrollado por el World Wide Web, está contemplado en una serie de RFCs, en la que la más importante es la RFC 2616, donde se define la sintaxis y la semántica que utilizan los elementos de una arquitectura web (clientes, servidores y proxies) para poder comunicarse. Sigue el método petición-respuesta, en el que un cliente (desde un navegador o cualquier otra aplicación) pide al servidor cierta información, que éste le devolverá.

Como característica a destacar, decir que es un protocolo sin estado, por lo que no almacena ninguna información sobre conexiones anteriores. Para guardar dicha información, es normal el uso de cookies en las aplicaciones y páginas web que lo requieran.

3.3.2 SSL

El protocolo de seguridad SSL (desarrollado por NetScape) ha sido bastante empleado hasta llegar al protocolo TLS que es el que suele utilizarse hoy en día. SSL es un protocolo cliente-servidor que proporciona servicios de seguridad básicos en conexiones: autenticidad, confidencialidad e integridad.

La capa en la que se asienta es una intermedia entre la de transporte y la de aplicación. Como su propio nombre indica, SSL es un protocolo orientado a sockets, lo que significa que todos los datos son enviados o recibidos mediante un socket que está protegido de forma criptográfica.

Como ya se ha mencionado, tanto SSL como TLS parten de la idea de un sistema híbrido, utilizando tanto sistemas simétricos como asimétricos.

Tiene una serie de fases básicas:

- Negociar entre ambas partes el algoritmo criptográfico que van a emplear, tanto para el cifrado de clave pública como para el de clave secreta.
- Intercambiar las claves públicas a través de los certificados
- Cifrado de la información basada en cifrado simétrico

SSL tiene una estructura basada en subcapas (3), que son:

- *SSL Record Protocol*: Para encapsular los datos de la subcapa superior para el protocolo de transporte.
- *SSL Handshake*: Es la capa principal. Permite la comunicación entre pares para la autenticación y la negociación de los parámetros necesarios para la conexión.
- *SSL Change Cipher Spec*: Permite poner los parámetros criptográficos en el lugar adecuado.
- *SSL Alert*: Permite intercambiar mensajes de alertas que indican problemas potenciales.
- *SSL Application Data*: Obtiene los datos de la capa superior (generalmente la de aplicación) para que los procese la capa SSL.

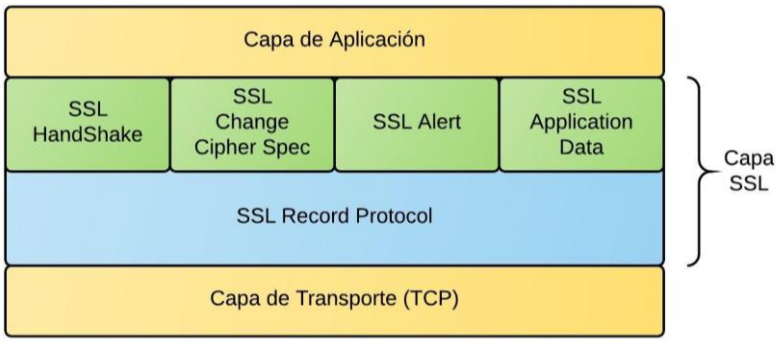

Figura 3-9. Subcapas del protocolo SSL

Este protocolo ha tenido tres versiones. La 1.0 nunca llegó a publicarse y la versión 2.0 salió en febrero de 1995 pero tenía una gran cantidad de fallos. Esto llevó al diseño de la versión 3.0, que fue publicada en 1996 y ha sido la que se ha utilizado en los últimos tiempos junto con las primeras versiones de TLS. En 2014 se generó una vulnerabilidad que ha acabado con el uso de esta versión 3.0, el ataque POODLE.

3.3.3 TLS

Es un protocolo estructuralmente idéntico a SSL. Está basado en el modelo cliente-servidor situado entre la capa de transporte y la de aplicación, que también utiliza los certificados X.509 para el cifrado asimétrico. Sigue la misma estructura de subcapas de SSL, cambiando solamente el prefijo por “SSL” por “TLS” (Figura 3-9).

TLS fue creado en 1999 por el IETF y ha sido actualizado en 2008 con la RFC 5246 y en 2011 con la RFC 6176.

La versión 1.0 es muy cerrada y permite conectarse en SSL 3.0, lo que debilita su seguridad. A pesar de sus escasas diferencias, tienen las suficientes como para evitar la interoperabilidad entre TLS 1.0 y SSL 3.0. La suite de cifrado (*Cipher Suites*) es el par de algoritmos usados para autenticar y cifrar mensajes, y además comprime el algoritmo de intercambio de clave. Esta versión soporta también el cifrado de SSL 3.0 en esta suite.

En 2001 se definió la versión 1.1, cuyas diferencias más destacadas son que añade protección contra ataques CBC y que da soporte para el registro de parámetros de IANA.

TLS 1.2 fue definido en 2008 y se basa en la versión 1.1. Esta nueva versión usa un nuevo PRF (*PseudoRandom Function Family*) que es más simple y fuerte que su predecesor (porque usa una sola función hash SHA-256 en vez de combinar dos como se hacía en las versiones anteriores). Además, para las firmas digitales la combinación de MD5 y SHA-1 también ha sido reemplazada por el uso de un solo valor hash. Se ha agregado en esta versión la definición de extensiones TLS y CipherSuites de AES.

En 2011, TLS 1.2 fue redefinido para evitar la retro-compatibilidad entre SSL y TLS, evitando que con TLS se pudiera negociar el uso de SSL 2.0. Hasta la fecha, 2015, existe un borrador de una versión 1.3, que incluye novedades pero cuyos detalles aún no están concretados.

3.3.3.1 Funcionamiento de TLS

Su funcionamiento se basa en el intercambio de registros, los cuales pueden ser empaquetados de forma opcional con un código de autenticación del mensaje (MAC). Cada registro tiene un campo *content_type* que dice el protocolo que tiene en la capa superior. Una función MAC tiene como objetivo comprobar la integridad de la información transmitida en un medio no confiable. Existen actualmente tres grandes grupos de funciones MAC: CBC-MAC tiene como idea convertir un algoritmo de cifrado simétrico en una función MAC, HMAC usa una función hash para implementar una función MAC (empleando MD5 o SHA-1) y UMAC parte de la idea de que un atacante necesita interactuar con el sistema para comprobar si el resultado MAC es válido o no.

Cuando se va comenzar la conexión, el nivel de registro encapsula otro protocolo, el *handshake* (acuerdo). El cliente manda un mensaje *ClientHello* con un listado de cifrados, métodos de compresión y versión del protocolo TLS más alto que permite. Éste recibirá un mensaje *ServerHello* donde el servidor decide los parámetros a utilizar a partir de las condiciones propuestas por el cliente. Una vez conocidos los parámetros, se intercambian los certificados X.509, con sus correspondientes claves públicas. Entonces se negocia la clave secreta que se va a utilizar para cifrar el tráfico de datos.

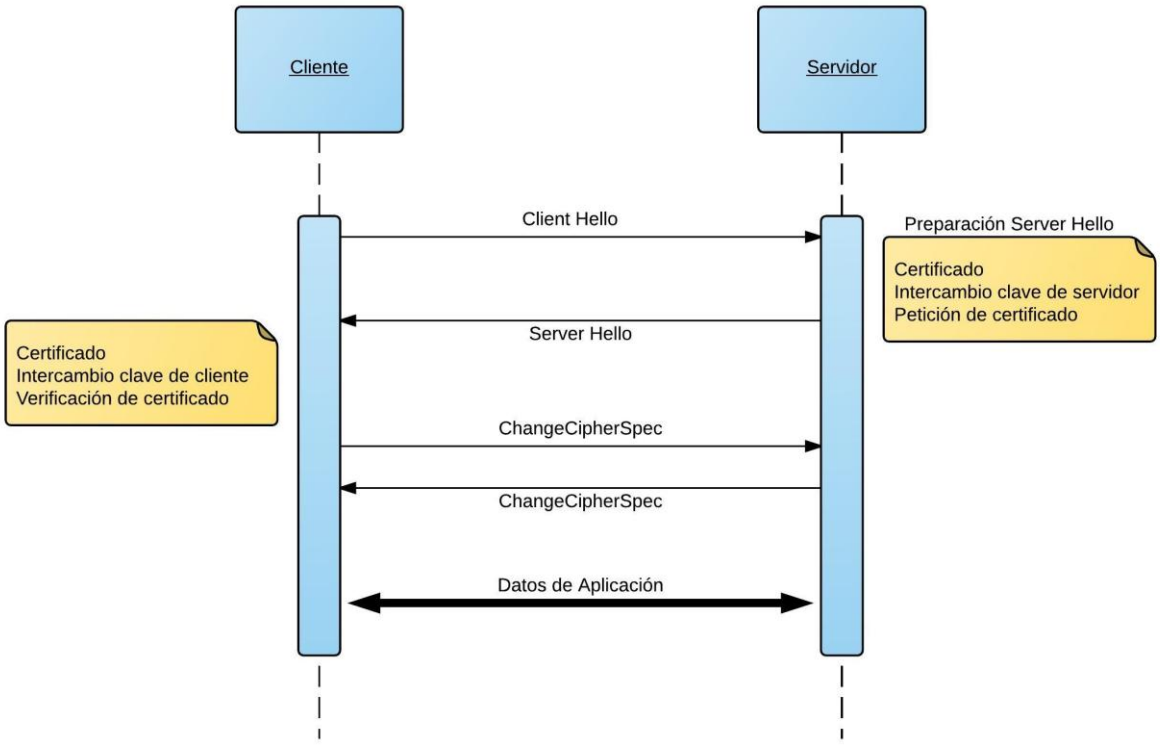

Figura 3-10. Paso de mensajes *handshake* TLS

En el proceso de intercambio de claves se intercambian las claves públicas para decidir la clave asimétrica a utilizar cuando se encripten los datos. La suite de cifrado indica los métodos criptográficos a utilizar, tanto para el cifrado simétrico como el asimétrico.

Los cifrados de clave pública que suelen utilizarse son los de RSA, de Diffie-Hellman (DH) y de curva elíptica

(ECDH). Respecto al cifrado de clave secreta, los cifrados más empleados son AES, Camellia o SEED. Algunos ejemplos de representación de las suites de cifrado son:

TLS_RSA_WITH_AES_256_CBC_SHA256

TLS_DH_DSS_WITH_AES_256_CBC_SHA

TLS_DH_RSA_WITH_AES_256_CBC_SHA256

Pueden verse el resto de suites de cifrado, en la RFC correspondiente a la versión 1.2 de TLS (4).

3.3.3.2 Ataques contra TLS

Uno de los objetivos de este trabajo es conseguir un sistema que pueda conectarse de forma segura. Pero son numerosos los ataques existentes contra estos protocolos, tanto para SSL como TLS. Todas las versiones de SSL han sido atacadas, lo que hace que ya no se recomiende su uso. TLS aún consigue resistir con las actualizaciones que recibe, aunque no deja de tener vulnerabilidades.

El ataque de renegociación, descubierto en 2009, permite introducir las peticiones del atacante en el inicio de la conversación entre cliente y servidor. No puede descifrar los datos, por lo que no se puede considerar como ataque MITM. Otro ataque es el de reversión de versiones, en la que permite que TLS vuelva a versiones anteriores y utilice cifrados más débiles en la negociación.

El ataque Beast es un exploit que usa un applet de Java a partir de las vulnerabilidades de CBC existentes en TLS 1.0. El ataque POODLE ya se ha comentado anteriormente, que aunque se basa en SSL 3.0, las versiones de TLS pueden permitir que en el *handshake* se rebaje la versión hasta llegar a SSL 3.0 y pudiendo explotar la vulnerabilidad.

Aunque el cifrado RC4 ya se sabía que estaba roto, era utilizado en TLS ya que la forma en la que estaba configurado evitaba que estos ataques pudieran realizarse. Esto fue así hasta 2013, cuando se encontró una vulnerabilidad que hizo que RC4 fuera quebrado por completo en TLS.

Otros que podemos encontrar son el ataque de truncamiento, que impedía que se cerrara la sesión y hacía que siguiera conectado al servicio que fuera, o el ataque *heartbleed* que afectaba a OpenSSL y por tanto al cifrado SSL/TLS.

4 TECNOLOGÍAS

El único sistema seguro es aquél que está apagado en el interior de un bloque de hormigón protegido en una habitación sellada rodeada por guardias armados.

- Gene Spafford -

Cuando hablamos de desarrollar un sistema seguro en Internet podemos encontrar una amplia oferta de soluciones, tanto para el caso de poner en marcha una PKI como para realizar una conexión cliente-servidor que use los certificados dados por esta infraestructura. En este capítulo detallaremos las tecnologías empleadas en el proyecto, en término de entornos y de programas utilizados.

4.1 Entorno y lenguajes de programación

El trabajo se ha desarrollado en un entorno Linux. La infraestructura de la Autoridad de Certificación, servidor OCSP y servidor Apache se ha implementado en máquinas Debian 7.3, mientras que la herramienta *nogotofail* para los ataques MITM se ha instalado en una máquina Ubuntu 14.04.1.

Gran parte del trabajo realizado, sobre todo con el tratamiento de certificados, se ha implementado usando Shell Script. El shell empleado ha sido *bash*, que es el utilizado en la mayoría de las distribuciones y consolas de Linux.

Se ha desarrollado una aplicación cliente-servidor para automatizar el proceso de solicitar y revocar certificados. Ambos lados de la aplicación distribuida se han diseñado en C, empleando para ello las librerías referidas a sockets y OpenSSL para encriptar la conexión TCP entre el servidor y la autoridad. Algunas de estas librerías empleadas son: `<sys/socket.h>`, `<openssl/err.h>`, `<openssl/rand.h>`, `<sys/types.h>`, `<openssl/ssl.h>`.

4.1.1 OpenSSL en C

OpenSSL es un proyecto colaborativo de código abierto que implementa una serie de herramientas de seguridad que utiliza los protocolos SSL y TLS, así como una robusta biblioteca de criptografía de propósito general. Está basada en la librería SSLeay desarrollada por Eric Young y Tim Hudson.

Suministra funciones criptográficas a otros paquetes como OpenSSH, navegadores web (para acceso a sitios HTTPS) e incluso es ampliamente utilizado en los sistemas Android. OpenSSL está escrito en C y da soporte para sistemas operativos de tipo UNIX, así como Microsoft Windows, DOS o MAC OS.

En nuestro proyecto, se va a emplear su librería dentro del programa escrito en C para la comunicación entre el servidor y la autoridad, así como para el tratamiento de certificados dentro de la autoridad. Esta segunda

funcionalidad se tratará en un apartado posterior.

La librería en C se denomina *ssl* e implementa tanto las versiones 2.0 y 3.0 de SSL como TLS v1. Con ella podremos encriptar una conexión TCP creada con sockets, usando alguno de estos protocolos. Las estructuras que trata son las siguientes:

- SSL_METHOD: Estructura de despachador de la librería interna ssl, para implementar los protocolos de seguridad.
- SSL_CIPHER: Soporta la información sobre un particular algoritmo de cifrado que va a utilizarse en SSL/TLS.
- SSL_CTX: Define el contexto global creado en un servidor o un cliente que define valores por defecto para la conexión.
- SSL_SESSION: Contiene los detalles de la sesión actual de la conexión: cifrado, certificados, claves, etc.
- SSL: Estructura principal creada por un servidor o un cliente para establecer la conexión.

Para la utilización de estas estructuras, en la librería se definen una serie de funciones que harán posible la comunicación entre ambas partes de forma cifrada. Para más información, ver la documentación al respecto en la web de OpenSSL (5).

4.1.2 Librerías en Java

Para el cliente, se ha realizado una aplicación desarrollada en Java, que se ha creado en un proyecto dentro del entorno Eclipse. La idea es disponer de un cliente sencillo para ver las opciones de seguridad que ofrece en su conexión con un servidor Apache al que se le ha configurado un certificado para establecer una comunicación segura.

En el servidor lo que hemos situado es un fichero en JSON, que el cliente leerá y lo mostrará en forma de tabla. Para parsear un fichero JSON, es necesario utilizar la librería *java-json* (6), la cual incluye todas las funciones necesarias para tratar con este tipo de formato.

Para tratar con certificados, en cuanto a validación y comprobaciones de seguridad, ha sido necesario recurrir a la librería Bouncy Castle Crypto Package (7). Es una API desarrollada por The Legion of Bouncy Castle Inc., y cuya última versión consiste en:

- API liviana de criptografía
- Proveedor de extensión y arquitectura de criptografía en Java
- Implementación limpia de JCE 1.2.1
- Librería para leer y escribir objetos ASN.1
- APIs livianas para TLS y DTLS
- Generadores de certificados X.509, CRLs y ficheros PCKS#12
- Generadores/Procesadores para OCSP
- Generadores/Procesadores para TSP, CMP y CRMF
- Generadores/Procesadores para OpenPGP
- Versiones jar disponibles para JDK 1.4-1.8 y Sun JCE
- Otros Generadores/Procesadores para S/MIME, CMS, TSP, CMP, CRMF, OpenPGP, etc.

Se va a emplear la última versión lanzada, la 1.52, que incluye las funcionalidades descritas, y que de las cuales vamos a emplear sobre todo las enfocadas a los certificados X.509 y a OCSP.

4.2 Tecnologías para PKI

Para la infraestructura que se ha montado se han utilizado una serie de programas y herramientas que nos facilitan su utilización. Hasta ahora hemos visto las librerías que se emplean dentro de los lenguajes de programación utilizados, pero nos quedan las aplicaciones externas que son ejecutadas en este conjunto pero que no han sido programadas para este proyecto en concreto.

Un aspecto a destacar es que todas las opciones que se van a emplear son gratuitas, dado el carácter académico del presente proyecto, que busca la creación de un conjunto que sea una solución totalmente libre, sin dependencia de licencias comerciales.

4.2.1 OpenSSL para la CA

Ya se ha comentado el uso de OpenSSL como librería de programación (Ver apartado 4.1.1). Pero además ofrece una amplia oferta de herramientas para su uso por línea de comandos, que permite una completa gestión de una autoridad de certificación.

Mediante el comando *openssl* se puede gestionar todo sobre certificados, como por ejemplo solicitarlos, generarlos y revocarlos por una autoridad, o generar CRLs. Un resumen de cómo ejecutar este comando es:

```
openssl comando [ opciones_comando ] [ argumentos_comando ]

openssl [ lista-comandos-estandar | lista-comandos-mensajes-digest | lista-comandos-cifrado | lista-algoritmos-cifrado | lista-algoritmos-mensajes-digest | lista-algoritmos-clave-publica ]

openssl no-XXX [ opciones arbitrarias ]
```

4.2.1.1 Comandos estándar

En la siguiente tabla se muestran algunos de los más útiles parámetros de la lista de comandos estándar que pueden añadirse.

Tabla 4–1 Comandos básicos OpenSSL

Nombre	Función
asn1parse	Parsea una secuencia ASN.1
ca	Gestión de una autoridad de certificación
ciphers	Descripción de la Suite de cifrado
cms	Utilidad CMS (Sintaxis de mensajes criptográficos)
crl	Gestión de las Listas de Revocación de Certificados
dsa	Gestión de datos DSA
ec	Procesamiento de clave de curva elíptica (EC)
enc	Codificación con cifrados
engine	Manipulación e información del motor OpenSSL
genpkey	Generación de clave privada o parámetros
genrsa	Generación de clave privada RSA
ocsp	Utilidad del protocolo OCSP
passwd	Generación de contraseñas hash
pkcs12	Gestión de datos PKCS#12
pkcs7	Gestión de datos PKCS#7
pkey	Gestión de claves pública y privada
pkeyparam	Gestión de parámetros de clave pública y privada
pkeyutl	Utilidad para algoritmos criptográficos de clave pública
rand	Generación de bytes pseudo-aleatorios

req	Gestión de petición de certificados (CSR) PKCS#10
rsa	Gestion de claves RSA
rsautl	Utilidad para firmar, verificar, cifrar y descifrar con RSA
s_client	Implementación de un cliente genérico SSL/TLS que puede establecer una conexión transparente a un servidor remoto que hable SSL/TLS. Está pensado para pruebas y test de aplicaciones, proporcionando una interfaz básica
s_server	Implementación de un servidor genérico SSL/TLS que acepta conexiones desde un cliente que habla SSL/TLS. Está pensado para pruebas y test de aplicaciones, proporcionando una interfaz básica
verify	Verificación de certificados X.509
version	Información de la versión OpenSSL

4.2.1.2 Comando openssl ca

Cabe mencionar esta opción, la cual proporciona todas las utilidades necesarias para gestionar una CA. Este comando, así como otras funcionalidades de OpenSSL, lee su configuración del fichero cuya ruta se establece con la variable de entorno OPENSSL_CNF. Si no se indica, suele situarse en la ruta */etc/ssl/openssl.cnf*. Aquí se especifican las rutas de la autoridad de certificación, de su certificado raíz, la clave privada de dicho certificado y del fichero de texto donde se representa la base de datos de los certificados emitidos por la autoridad.

A continuación se muestra un resumen de los parámetros de este comando de la CA, tomado directamente desde la web del proyecto OpenSSL y donde se puede encontrar toda la documentación al respecto (8).

```
  openssl ca [-verbose] [-config filename] [-name section] [-gencrl] [-revoke file] [-
status serial] [-updatedb] [-crl_reason reason] [-crl_hold instruction] [-
crl_compromise time] [-crl_CA_compromise time] [-crldays days] [-crlhours hours] [-
crlexts section] [-startdate date] [-enddate date] [-days arg] [-md arg] [-policy arg]
[-keyfile arg] [-keyform PEM|DER] [-key arg] [-passin arg] [-cert file] [-selfsign] [-
in file] [-out file] [-notext] [-outdir dir] [-infiles] [-spkac file] [-ss_cert file]
[-preserveDN] [-noemailDN] [-batch] [-msie_hack] [-extensions section] [-extfile
section] [-engine id] [-subj arg] [-utf8] [-multivalue-rdn]
```

4.2.1.3 Uso de algunos comandos

Vamos a ver algunos ejemplos de uso de esta potente herramienta, que serán la base de los scripts de nuestro proyecto para la gestión de certificados.

Para la creación de un certificado raíz para una nueva autoridad vamos a emplear lo siguiente:

```
openssl req -new -nodes -x509 -keyout private/cakey.pem -out cacert.pem -days 365 -
subj "/C=$PAIS/ST=$PROVINCIA/L=$LOCALIDAD/O=$ORGANIZACION/OU=$UNIDAD_ORG/CN=$NOMBRE"
```

Con este comando se crea un certificado para esta autoridad. Se genera un certifica x.509 con una clave privada que se genera en *private/cakey.pem* y con salida en el fichero cacert.pem con una caducidad de 365 días. Este certificado será el que habrá que exportar a los diferentes servidores que vayan a utilizar sus certificados. La opción *-nodes* indica que no es necesario utilizar una contraseña para generarlo, ya que sería necesario introducirla cada vez que reiniciemos el servicio.

Se le pasan como argumentos también los valores solicitados para generar el certificado, con la opción *-subj*.

Si lo que queremos es realizar una petición CSR (mensaje enviado de un solicitante a una CA para pedir un certificado digital) desde el equipo del servidor a encriptar se debe ejecutar el siguiente comando:

```
  openssl req -nodes -new -key $RUTA_ACTUAL/fich/server.key -out fich/peticion.csr -
subj "/C=$PAIS/ST=$PROVINCIA/L=$LOCALIDAD/O=$ORGANIZACION/OU=$UNIDAD_ORG/CN=$1"
```

Al igual que en el anterior, este comando es una petición, que en este caso, en vez de generar un certificado, genera un fichero tipo .csr, lo que indica que es una petición solicitando un certificado firmado.

En los argumentos que se le pasan con la opción *-subj* hay que destacar el último valor, que es el nombre o dirección del servidor, que en este caso aparece un $1 porque se le pasa como parámetro al script desde el que se ha obtenido.

Una vez se ha enviado dicha petición a la CA, esta deberá generar un certificado como se le requiere. Para ello se usa el comando *openssl ca*, como se muestra a continuación:

```
  openssl ca -batch -policy policy_anything -keyfile private/cakey.pem -out
certs/$CERT.crt -in peticion.csr
```

El comando obtiene la petición csr que ha recibido del servidor, y devuelve el certificado con una extensión .crt. La opción *-policy* indica el tipo de política para generar el certificado, que en este caso es cualquiera. La opción *-batch* indica que no se generen preguntas a la salida y cree el certificado directamente.

Si deseamos revocar un certificado (siendo en el ejemplo *certs/$file* la ruta donde se encuentra el certificado guardado en la CA), el comando a realizar desde la autoridad es:

```
  openssl ca -revoke certs/$file
```

Para la creación de una CRL que contenga los certificados revocados, puede utilizarse este comando:

```
  openssl ca -gencrl -keyfile private/cakey.pem -out crl/crl.pem
```

Todos estos comandos han sido obtenidos a partir de los diferentes scripts que se han desarrollado para la creación de la CA, habiendo mostrado los que pueden tener mayor interés. En capítulos posteriores se expondrá la estructura de esta autoridad y todos los scripts que la soportan.

4.2.2 OpenCA OCSPD

La autoridad de validación para la infraestructura desarrollada se va a basar en el protocolo OCSP. Para ello es necesario disponer de un servidor que responda a las peticiones de los clientes sobre el estado de los certificados.

El proyecto OpenCA OCSPD nace con la intención de desarrollar un respondedor OCSP que fuera robusto y fácil de instalar. El servidor está desarrollado como una aplicación autónoma para que pueda ser integrado en diferentes soluciones PKI y que no dependa de un esquema específico de base de datos. Además, puede ser configurado para responder a múltiples CAs.

Este servidor OCSP sigue la RFC 2560 (2) y su objetivo principal es el de proporcionar una herramienta online que permita verificar el estado de los certificados desde clientes o navegadores tales como Firefox.

Está incluido en el paquete principal de la distribución OpenCA (que también dispone de una solución de código abierto de PKI completa), y al igual que esta es de acceso libre. Se puede instalar de forma autónoma (como haremos en este proyecto) y necesita una CRL (o acceso a un servidor LDAP desde donde obtener la

CRL) para ver los certificados revocados por la autoridad de certificación.

Se ha instalado la última versión disponible, la 3.1.2, la cual arregla importantes fallos que afectaban a la instalación en versiones anteriores.

La instalación de OCSPD requiere la librería LibPKI, también de OpenCA, que proporciona al desarrollador todas las funcionalidades para gestionar certificados, tanto para su generación como para su validación. Realiza operaciones criptográficas complejas para que puedan ser llamadas desde una simple función en implementaciones de alto nivel. Constituye el corazón de OpenCA, tanto de su PKI como de su servidor OCSPD en el caso de instalarlo de forma autónoma.

Se han incluido en el proyecto tanto la librería como el código fuente de OCSPD, que el script de instalación de la propia autoridad se encargará de instalar. Para ello, consultar el Anexo A, correspondiente a la guía de instalación.

4.2.2.1 Comparativa con otras soluciones

La solución de OpenCA no es la única existente de forma gratuita. Es por ello que se ha comparado con otras herramientas para ver cuál era la que mejor se adaptaba al propósito de nuestro trabajo.

El comando por consola de OpenSSL permite también configurar un servidor OCSP que escuche dichas peticiones. Se realiza con la opción *ocsp* y puede tanto ejecutarse el respondedor como realizar peticiones como si de un cliente se tratara para comprobar el estado de un certificado.

Si se desea poner en marcha el servidor tan solo hay que ejecutar lo siguiente:

```
  openssl ocsp -index index.txt -CA cacert.pem -rsigner cacert.pem –rkey
private/cakey.pem -port 8888
```

Como podemos observar, se actúa directamente sobre el fichero *index.txt* que contiene la base de datos de los certificados de la autoridad. Esto hace que no se necesite regeneración de CRL, ni siquiera generarla, lo que puede resultar más cómodo.

Sin embargo, se busca una solución que aporte más seguridad, es decir, que no interfiera en el fichero índice de la autoridad, para que no se pueda llegar a acceder a ésta.

Si somos un cliente y queremos obtener el estado de un certificado, basta con escribir el comando que se muestra a continuación, siempre que tengamos instalada la herramienta *openssl* en nuestro sistema.

```
  openssl ocsp -CAfile cacert.pem -issuer cacert.pem –cert certs/prueba.crt -url
http://localhost:8888
```

La solución EJBCA está desarrollada en Java, y se basa en una robusta, flexible, escalable e independiente, que puede ser usada de forma autónoma o integrada con otras aplicaciones. También contiene una PKI completa que se puede usar de forma gratuita.

Las principales características del respondedor OCSP de EJBCA son las siguientes:

- Implementación de RFC 2560, RFC 6090 y RFC 5019
- Independiente del uso del software de la CA de EJBCA (aunque se requerirán ciertos componentes para su instalación)
- El servidor puede responder para cualquier número de CAs
- Estado de la información almacenado en una base de datos SQL
- No dependiente de CRL, información en tiempo real
- Alto grado de configuración y mecanismos para extensiones OCSP

- Escalabilidad lineal y alta disponibilidad para múltiples nodos

Como podemos ver esta utilidad proporciona un alto número de funciones. Como gran diferencia con la solución de OpenCA es que se basa en el lenguaje SQL para almacenar los datos. Esto supone tener una base de datos configurada donde se almacene la información acerca de los certificados. Al no tener que depender de CRLs permite obtener el estado en tiempo real. Como inconveniente, esta solución es mucho más pesada y tiene una instalación y configuración más difícil.

El motivo por el que se ha decidido trabajar con la solución de OpenCA es que uno de los objetivos del proyecto es que el proceso sea simple, y pueda integrarse en el script de instalación para que el usuario tan solo tenga que ejecutar un comando. EJBCA, a pesar de proporcionar más características, tiene un proceso de instalación más complejo, además de requerir máquinas más potentes y consume más recursos.

El problema de tener que trabajar sobre CRLs desaparece al instalar el servidor OCSP y la autoridad en una máquina de forma conjunta. Así se toman las CRLs cada vez que haya una revocación en la CA, automatizando el proceso en el programa creado para dar una solución "en tiempo real". En el caso de instalar ambos componentes en máquinas diferentes habría que reconsiderar esto.

Tabla 4–2 Resumen comparativa de soluciones servidor OCSP

	OpenCA	OpenSSL	EJBCA
Lenguaje	C	C	Java
Soporte	Multiplataforma	Multiplataforma	Multiplataforma
Funcionamiento modo autónomo	Sí	Sí	Sí
Requiere otras librerías en su instalación	Sí	No	Sí
Soporta múltiples CAs	Sí	No. Un servidor ejecutado por CA	Sí
Obtención información de certificados	CRL	Fichero index.txt	Base de datos SQL
Información en tiempo real	No. Necesita actualización de CRL	No	Sí

4.2.2.2 Configuración de OCSPD

Para la configuración, OCSPD dispone de unos ficheros en XML, lo que facilita su configuración por ser un estándar conocido. Este fichero se va a encontrar en la ruta donde se indique en su instalación, por defecto */usr/etc/ocspd/ocspd.xml*. A continuación se detalla el contenido de este fichero.

```
<?xml version="1.0" ?>
<!-- OCSP Daemon configuration -->
<pki:serverConfig xmlns:pki="http://www.openca.org/openca/pki/1/0/0">
   <!-- General Section: details about server configuration and dirs -->
   <pki:general>
      <pki:pkiConfigDir>/usr/etc/ocspd/pki</pki:pkiConfigDir>
      <pki:token>ocspServerToken</pki:token>
      <pki:caConfigDir>/usr/etc/ocspd/ca.d</pki:caConfigDir>
      <pki:pidFile>/usr/var/run/ocspd.pid</pki:pidFile>
      <pki:spawnThreads>10</pki:spawnThreads>
      <pki:crlAutoReload>3600</pki:crlAutoReload>
      <pki:crlReloadExpired>yes</pki:crlReloadExpired>
      <pki:crlCheckValidity>600</pki:crlCheckValidity>
   </pki:general>
```

```
    <!-- Security Related Configurations -->
    <pki:security>
       <pki:user>tfgca</pki:user>
       <pki:group>tfgca</pki:group>
       <!-- <pki:chrootDir></pki:chrootDir> -->
    </pki:security>

    <!-- Service Network Configuration -->
    <pki:network>
       <pki:bindAddress>http://0.0.0.0:2560</pki:bindAddress>
       <pki:httpProtocol>1.0</pki:httpProtocol>
       <pki:httpBaseURL></pki:httpBaseURL>
       <pki:timeOut>5</pki:timeOut>
    </pki:network>

    <!-- OCSP response configuration -->
    <pki:response>
       <pki:maxReqSize>8192</pki:maxReqSize>
       <pki:digestAlgorithm>SHA1</pki:digestAlgorithm>
       <pki:signatureDigestAlgorithm>SHA1</pki:signatureDigestAlgorithm>
       <pki:validity>
          <pki:days>0</pki:days>
          <pki:mins>5</pki:mins>
       </pki:validity>
    </pki:response>
 </pki:serverConfig>
```

Se van a describir cada uno de los campos que se pueden configurar de este xml:

- Configuración general
    - `pkiConfigDir`: Directorio donde se encuentran las configuraciones de LibPKI
    - `token`: Nombre del identificador de configuración que se va a usar en el servidor
    - `caConfigDir`: Directorio que contiene todos los ficheros con las configuraciones que soportan las CAs
    - `pidFile`: Fichero donde el servidor escribirá su propio identificador de proceso (PID) en el arranque
    - `spawnThreads`: Número de hilos que van a generarse
    - `crlAutoReload`: Tiempo de auto-recarga (segundos)
    - `crlReloadExpired`: Recarga de CRLs expiradas (sí/no)
    - `crlCheckValidity`: Comprueba la validez de las crls cada x segundos
- Configuración relativa a la seguridad
    - `user`: Usuario con el que el servidor va a ser ejecutado y que tiene los privilegios para acceder a los ficheros necesarios
    - `group`: Grupo con el que el servidor va a ser ejecutado
- Configuración de los servicios de red
    - `bindAddress`: Dirección y puerto donde se va a escuchar desde el arranque del servidor. Si la dirección es 0.0.0.0 es que se escucha en cualquier dirección, y el puerto por defecto es 2560
    - `httpProtocol`: Indica si el soporte es para HTTP 1.0 o 1.1
    - `httpBaseURL`: Se usa si se quiere responder sólo a ciertas urls, dejándolo en blanco en otro caso

  - `timeOut`: Tiempo en el que se intenta una petición entrante al servidor. Si pasa este tiempo, el servidor cerrará el socket

- Configuración de la respuesta OCSP
  - `maxReqSize`: Tamaño máximo de las peticiones entrantes
  - `digestAlgorithm`: Algoritmo de resumen usado para construir las respuestas. Actualmente, el estándar especifica que sólo se soporta SHA1
  - `signatureDigestAlgorithm`: Algoritmo de resumen usado para firmar las respuestas. Actualmente, el estándar especifica que sólo se soporta SHA1
  - `validity`: Periodo de validación de las respuestas. Se supone que los clientes no van a realizar consultas de la misma CA en ese periodo

Dentro del directorio de configuración de OCSPD, encontramos otro directorio que es `ca.d/`, donde se encuentran las configuraciones de las distintas CAs soportadas. Aquí tendremos un fichero XML por cada CA, destacando de ellos los campos `caCertUrl`, que es la url desde donde se descargará el certificado raíz de la autoridad, y `crlUrl`, que indica la dirección desde donde descargarse la CRL que contiene el listado de certificados revocados.

4.2.3 Otras herramientas

Además de las utilidades para la gestión y consulta de certificados, en la infraestructura se han empleado otros programas que mencionamos en este subapartado.

4.2.3.1 GNU Wget

Permite la descarga de contenidos desde un servidor web de forma muy simple. Soporta HTTP, HTTPS y FTP. Escrito en C, no necesita muchos recursos y es totalmente portable, necesitando tan solo un compilador de C y una conexión a la red. Está diseñado para ejecutarse por línea de comandos, y se puede emplear tanto en sistemas Unix como Windows.

Esta utilidad ha sido usada en el proyecto en primer lugar como forma de descargarse el fichero de validación propuesto por la autoridad al servidor, para comprobar que éste existe y funciona correctamente. Se ha elegido por su facilidad de uso y su buena integración con Shell Script.

Además, se ha utilizado para realizar pruebas con *nogotofail* con el servidor, comprobando la conexión con el servidor ante ataques MITM.

4.2.3.2 Apache

El servidor web Apache (httpd) es archiconocido por ser de código libre y empleado en plataformas Unix, Windows, Macintosh, etc. El objetivo es proporcionar un servidor seguro, eficiente y extensible que proporcione servicios HTTP en sincronización con los estándares HTTP. Fue lanzadado en 1995, y es el servidor web más popular en Internet desde abril de 1996.

Su utilización se basa en la puesta en marcha del servidor seguro, para el cual se solicitan los certificados a la autoridad. Es importante destacar que dentro del trabajo uno de los objetivos es la configuración de este servidor para que sea seguro. Para ello es necesario activar el módulo ssl y modificar los ficheros de configuración. En la versión utilizada en Debian (Apache 2.2.22) el archivo principal modificado se encuentra en la ruta por defecto /etc/apache2/sites-available/default-ssl, que incluye un VirtualHost al puerto 443 con la configuración del módulo ssl.

De este fichero nos centraremos principalmente en las líneas que indican dónde se encuentra el certificado proporcionado por la autoridad, que debe estar correctamente configurado:

```
SSLCertificateFile    /etc/ssl/certs/certTFG.crt
SSLCertificateKeyFile /etc/ssl/private/server.key
```

Una de las claves del éxito de Apache reside precisamente en sus módulos. Este servidor es extensible y soporta módulos tales como el de ssl para seguridad, o como PHP, CGI o JSP para soporte de estos lenguajes de programación.

En este servidor nos basaremos para situar el fichero con formato JSON que posteriormente el cliente leerá. Además de esto, Apache será utilizado en el proceso de solicitud de un certificado a la CA para situar el fichero de validación que proporciona la autoridad, para que ésta pueda descargárselo y validarlo.

4.2.3.3 GNU sed

Potente editor de texto de flujos (stream editor) que se basa en el entorno de la consola de sistemas Unix. Acepta como entrada un fichero (flujo), que lee y modifica y muestra su resultado normalmente por salida estándar o guardándolo en otro o en el mismo fichero.

Se ha empleado en los scripts de la autoridad y del servidor para modificar los ficheros de configuración de OpenSSL y de Apache que han sido necesarios.

4.2.3.4 cURL

Herramienta software para transferencia de archivos a partir de una URL mediante línea de comandos. Soporta FTP, FTPS, HTTP, HTTPS, TFTP, SCP, SFTP, Telnet, LDAP, etc. Es de código abierto y su objetivo principal es el de automatizar transferencia de archivos para operaciones no supervisadas, siendo útil por ejemplo para su uso en scripts.

En el presente trabajo se va a emplear como herramienta de pruebas para el ataque *heartbleed* de *nogotofail*, con el que se probará que es posible encontrar hoy día clientes que tengan vulnerabilidades a los ataques que se van a realizar.

4.3 Herramienta nogotofail

Nogotofail es una herramienta de pruebas en red diseñada por Google para ayudar a los desarrolladores e investigadores de seguridad a localizar y solucionar conexiones débiles TLS/SSL y tráfico sensible a lectura de terceros en dispositivos y aplicaciones de forma flexible, escalable y potente. Incluye pruebas para la verificación de certificados SSL comunes, así como fallos en HTTPS y librerías TLS/SSL, tráfico no cifrado, asuntos de vulnerabilidades SSL y STARTTLS, etc.

Los tres casos de uso para los que está pensado son:

- Encontrar fallos y vulnerabilidades
- Verificar correcciones y ver progresos
- Entender qué trafico están generando ciertas aplicaciones y ciertos dispositivos

Lo importante es que se suponga un entorno real para realizar las pruebas. Por tanto la herramienta es flexible a cualquier dispositivo, para analizar cualquier tráfico de red. La herramienta no se interpone en el camino de forma normal, ya que los test que son destructivos se ejecutan por defecto sólo cuando son necesarios y con baja probabilidad.

Está diseñado para situarse como un "hombre en el medio" de la conexión entre cliente y servidor, de forma que lea toda la información intercambiada. Construida en Python, el servidor MITM es el núcleo de la herramienta, el cual se denomina *nogotofail.mitm* y se encarga de interceptar el tráfico TCP. Se centra en tener una serie de *handlers* (manejadores) que en cada conexión son los responsables de modificar el tráfico de forma activa para probar vulnerabilidades o aspectos pasivos de la conexión. Detecta el tráfico vulnerable usando DPI (inspección profunda de paquetes) en vez de estar basado en número de puertos. Además, es capaz de testear el tráfico TLS/SSL en protocolos que usan STARTTLS (extensión que ofrece una forma de mejorar desde una conexión en texto plano a una conexión cifrada, en lugar de utilizar un puerto distinto para la comunicación cifrada).

Esta utilidad no realiza ataques destructivos a todas las conexiones TLS/SSL que ve, porque tantos ataques

puede provocar que los clientes no vulnerables aborten las conexiones atacadas. Si se atacan todas las conexiones es posible que nunca lleguen a verse vulnerabilidades que tienen una dependencia de una conexión con éxito. Por ejemplo, en el caso de que se necesite registrarse en un servicio con HTTPS antes de realizar cualquier acción que puede ser que tenga un tráfico vulnerable. Si el acceso al registro nunca es realizado correctamente nunca se podrá detectar dicho tráfico vulnerable.

Además, el hecho de darle una baja probabilidad puede usarse para utilizar esta herramienta en largos periodos de tiempo, que da una cobertura de detección mucho mayor. En el caso de querer probar una conexión específica se puede poner una probabilidad del 100% para realizar el ataque sí o sí.

Además, es posible configurar un cliente opcional para el servidor MITM que permite ver el registro y configurar el cliente si se ha instalado en un dispositivo al que no se puede tener acceso. Esta funcionalidad no va a ser utilizada en el cliente, ya que se tiene acceso completo al servidor nogotofail configurado, desde donde se obtendrán los logs y se configurará en función del ataque a analizar.

Soporta tres modos de intercepción de tráfico. El modo *tproxy* usa iptables tproxy y reglas de ruta ip para enrutar todo el tráfico desde el dispositivo a través del servidor *nogotofail*. El modo *redirect* usa redirección NAT con iptables para enrutar el tráfico a *nogotofail*, con el inconveniente de que tiene problemas con el soporte para IPv6. Los dos modos anteriores son transparentes al cliente y al destino. El modo *socks* funciona como un proxy SOCKS5. No necesita reglas iptables, por lo que no necesita acceso root ni tener que ser configurado para que esté en la ruta de la información, pero pierde la transparencia de los otros modos. Este modo es útil para probar cambios y es el que vamos a utilizar en el proyecto.

La vida de una conexión con *nogotofail* funcionando pasa por las siguientes etapas:

- Cuando una conexión es creada por primera vez se selecciona el manejador inicial para la conexión dentro de la lista de *handlers* existente.
- Una vez que la conexión con el extremo es correcta, se llama al establecimiento de cada manejador.
- Cuando se envía datos desde el cliente al servidor se lanza una petición del manejador con la conexión de éste y entonces empieza a analizar los datos.
- Cuando se envían datos del servidor al cliente es captado a través de la respuesta del manejador.
- Si se detecta un mensaje del tipo TLS/SSL Client Hello se comprueba si debería ser atacado.
- Cuando se cierra la conexión, se llama al cierre del manejador.

Para el caso de ejecutar el servidor en modo *socks*, una posible ejecución es:

```
python -m nogotofail.mitm --mode socks --port 8080 --serverssl server.crt
```

Se le indica el puerto en el que escucha, así como el certificado utilizado para la conexión. Esta ejecución es la básica, introduciendo de forma aleatoria ciertos ataques y otras veces dejando que funcione correctamente la comunicación. En el capítulo dedicado a esta herramienta en el trabajo se verán todas las ejecuciones posibles.

4.3.1 Proxychains

Permite ejecutar cualquier programa a través de un proxy HTTP o SOCKS. Esta herramienta fuerza a todas las conexiones de la aplicación dada a seguir la lista de servidores proxy definida por el usuario. Es de código libre y puede ser usado para acceder a Internet a través de un firewall, ocultar la IP o ejecutar SSH, telnet, wget o cualquier otra aplicación a través de un proxy con un interés en particular.

En este caso se necesita para que todos los datos pasen por la máquina donde se encuentra instalado *nogotofail* antes de ser enviados al servidor. Con esto se hará funcionar la herramienta sin problemas, permitiendo ver y solucionar los problemas de las conexiones seguras que puedan ser vulnerables.

5 SERVIDOR Y AUTORIDAD DE CERTIFICACIÓN

Las organizaciones gastan millones de dólares en firewalls y dispositivos de seguridad, pero tiran el dinero porque ninguna de estas medidas cubre el eslabón más débil de la cadena de seguridad: la gente que usa y administra los ordenadores.

- Kevin Mitnick -

A partir de este capítulo, y en los tres siguientes, se va a analizar el diseño e implementación de la infraestructura que se ha creado para este proyecto, basándonos en la idea que se mostraba en la introducción (Figura 1-1).

En este en concreto se va a ver cómo se ha diseñado el programa que automatizará el proceso de solicitud y entrega de certificados entre el servidor y la autoridad, así como posteriormente el de revocación. Para ello nos basamos en los conocimientos teóricos expuestos sobre las CAs y su gestión de firmas digitales.

5.1 Creación de la autoridad

Basándonos en la herramienta OpenSSL se ha diseñado un script denominado *instalaCA.sh* donde se procede a crear la autoridad certificadora y configurarla para su uso. La creación de ésta requiere un conjunto de directorios y ficheros que se detallan a continuación (ver también Figura 5-1).

- *serial*: Contiene el número de serie del siguiente certificado que firmemos, inicialmente se crea con 01\n.
- *crlnumber*: Contiene el número de serie de la siguiente lista de certificados revocados, inicialmente se crea con 01\n.
- *index*.txt: "Base de datos" con información sobre los certificados firmados.
- *certs*/: Directorio donde se almacenan los certificados ya firmados y enviados a los clientes.
- *private*/: Directorio donde se guarda la clave privada de la autoridad, así como las demás claves que se generen para otros servicios.
- *newcerts*/: Directorio donde se almacenan los certificados que acaban de ser firmados. En este caso no se va a utilizar porque nuestro programa lo enviará directamente al servidor que lo solicita.
- *crl*/: Directorio donde se guardan los certificados que han sido revocados (por medio de las CRLs).

Cuando se genera el conjunto de directorios es necesario modificar la configuración de OpenSSL para que tome la nueva ruta donde se hayan creado los directorios como la ruta de la nueva autoridad. La ruta del

fichero a modificar es */etc/ssl/openssl.cnf* y se va a realizar la sustitución de la línea concreta con el editor sed.

Tras dar todos los permisos necesarios, se procede a generar un certificado raíz que será la esencia para firmar los certificados. Se devuelve un certificado público de la CA que deberá ser distribuido a todos los clientes para que al acceder al servidor que contenga un certificado firmado por ésta verifiquen su autenticidad. Este certificado se denominará *cacert.pem* y la clave privada generada se almacenará en el directorio *private* como *cakey.pem*.

Además, es necesario crear una CRL para que pueda ser obtenida posteriormente por el servidor OCSP.

Para la instalación es necesario invocar el script como usuario root. Durante ésta se crea un nuevo usuario y grupo en el sistema, *tfgca*, que será el que tendrá acceso a los directorios de la autoridad, y desde él se realizará la ejecución de la CA, ya que será el único que tenga permiso para realizar todas las acciones. Esto se ha realizado como medida de seguridad, para evitar que sea el usuario root el que haga todos los cambios, ya que puede resultar peligroso.

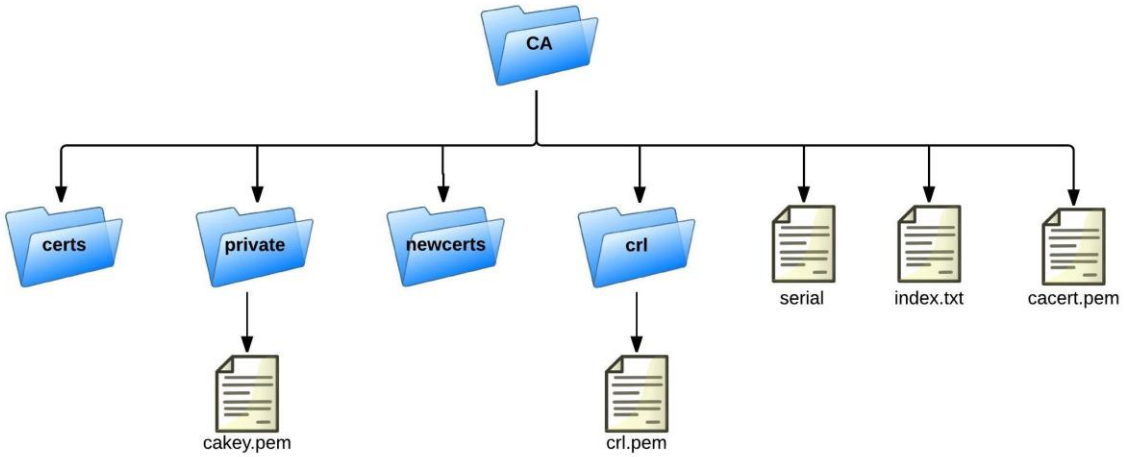

Figura 5-1. Conjunto de directorios de una CA

Para el caso de que se quiera desinstalar dicha CA, se ha creado un script llamado *desinstalaCA.sh* en el que se eliminará el conjunto de directorios creados durante la instalación, así como todos los certificados y CRLs generados por la autoridad. También se restaurará el contenido del fichero de configuración a su estado por defecto (antes de la instalación). Para ello, durante la instalación se habrá creado una copia de dicho archivo en un directorio denominado *backup*.

Una vez instalada, la autoridad tendrá que ser capaz de generar y revocar certificados firmados, así como validar la existencia de un servidor solicitante. Todo esto se realizará mediante scripts a los que accederá el programa de ejecución de la CA, y que se expone en el siguiente apartado.

5.2 Aplicación distribuida autoridad-servidor

Hay que recordar que uno de los principales objetivos de este proyecto es el de simplificar la tarea de los administradores de los servidores a la hora de solicitar un certificado a una autoridad. Es por ello que se ha diseñado un programa que permite la comunicación entre la CA y el servidor para automatizar este proceso.

Escrito en C, se basa en un modelo cliente-servidor que usa sockets, que mediante TCP y con una capa de cifrado TLS hace que se comuniquen ambas entidades. Tiene dos programas que se ejecuta cada uno en un extremo de la conexión: *servicioCA* (que será ejecutado en la máquina donde se instale la CA) y *clienteCA* (que se ejecutará en la máquina del servidor que solicita el certificado). Además, estos harán uso de una serie de scripts para la interacción con el sistema.

La idea es que el servidor necesita poner en marcha un servicio web para que sea utilizado por otros clientes, y quiere que esas conexiones sean cifradas, empleando HTTPS. Como ya se ha visto, es necesario solicitar un certificado a la autoridad que será firmado por ésta y posteriormente que el programa se encargue de

configurar Apache en la máquina servidor para que funcione bajo ese cifrado. La configuración requiere permisos de superusuario en el servidor, que serán solicitados por el programa cuando sea necesario.

5.2.1 Programa autoridad

En el lado de la autoridad de certificación, con este programa se permite estar a la escucha de peticiones y realizar diferentes acciones según sea el caso. Para ello se han creado los ficheros *servicioCA.c* (que contiene la función principal) *funcionesAutoridad.c* (con funciones necesarias para el correcto funcionamiento de la autoridad) y *conexionCifradaCA.c* (que tiene las funciones para encriptar la conexión).

Se ha creado un script principal denominado *autoridad.sh* y que será la base para la ejecución de los comandos, tales como instalación, desinstalación o ejecución. Tiene dos formas de ser ejecutado, por un lado comandos independientes que se pueden escribir directamente en la consola o mediante un menú y siguiendo las instrucciones que indica. Todo esto está más detallado en el Anexo A que contiene la guía de instalación y uso. El comando que lleva a la ejecución arrancará el programa *servicioCA*. La compilación de éste se lleva a cabo ejecutando el comando de instalación.

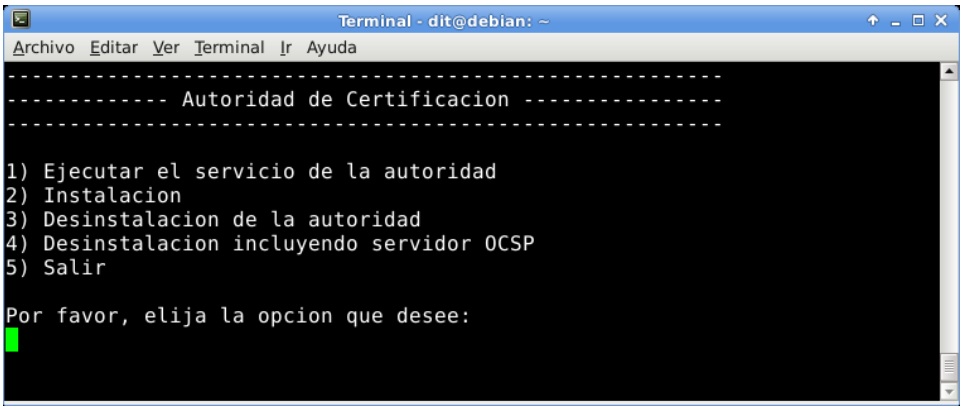

Figura 5-2. Menú principal autoridad.sh

5.2.1.1 servicioCA.c

Contiene el *main* del programa *servicioCA*, así como funciones para escuchar peticiones y atender al cliente (en este caso el servidor solicitante). Importará las funciones de los otros archivos .c que definen funcionalidades concretas de la autoridad y de la conexión.

La función principal comienza con la declaración de las variables necesarias y la comprobación de los argumentos pasados (tiene que indicarse el puerto en el que se va a escuchar). Se inicializa la librería SSL para encriptar la conexión TCP con el otro extremo, y se llama a las funciones *iniciaServidorCTX* (se encarga de configurar e inicializar el contexto SSL) y *cargaCertificados* (indicando los certificados público y privados que se van a emplear para la comunicación).

Crearemos el socket servidor llamando a la función *escucha*, que a partir del puerto realiza la inicialización del socket TCP y de escucha haciendo uso de *listen*. Se devolverá el socket nuevo a la función *main*.

Con esto, entramos en un bucle para que se estén esperando conexiones continuamente. Creamos una dirección del tipo *struct sockaddr_in* y un puntero a SSL. Si llega una petición, se llama a la función *accept* que a partir del socket de servidor creado y la dirección acepta dicha solicitud.

Ahora hay que encriptar dicha conexión entrante, por lo que se crea un nuevo estado de SSL con un contexto y se establece el socket del nuevo cliente que hace la petición a dicho estado, con la función *SSL_set_fd* de la librería de OpenSSL.

Una vez aceptada por parte de la autoridad, hay que atender la petición, con la función atiendeCliente que se detalla a continuación. Con esta función, aparte de establecer correctamente la conexión con el cliente (recordamos que será el servidor solicitante) se comprobará el estado de la petición y en función de si es una solicitud para generar o revocar un certificado se realizarán las acciones pertinentes.

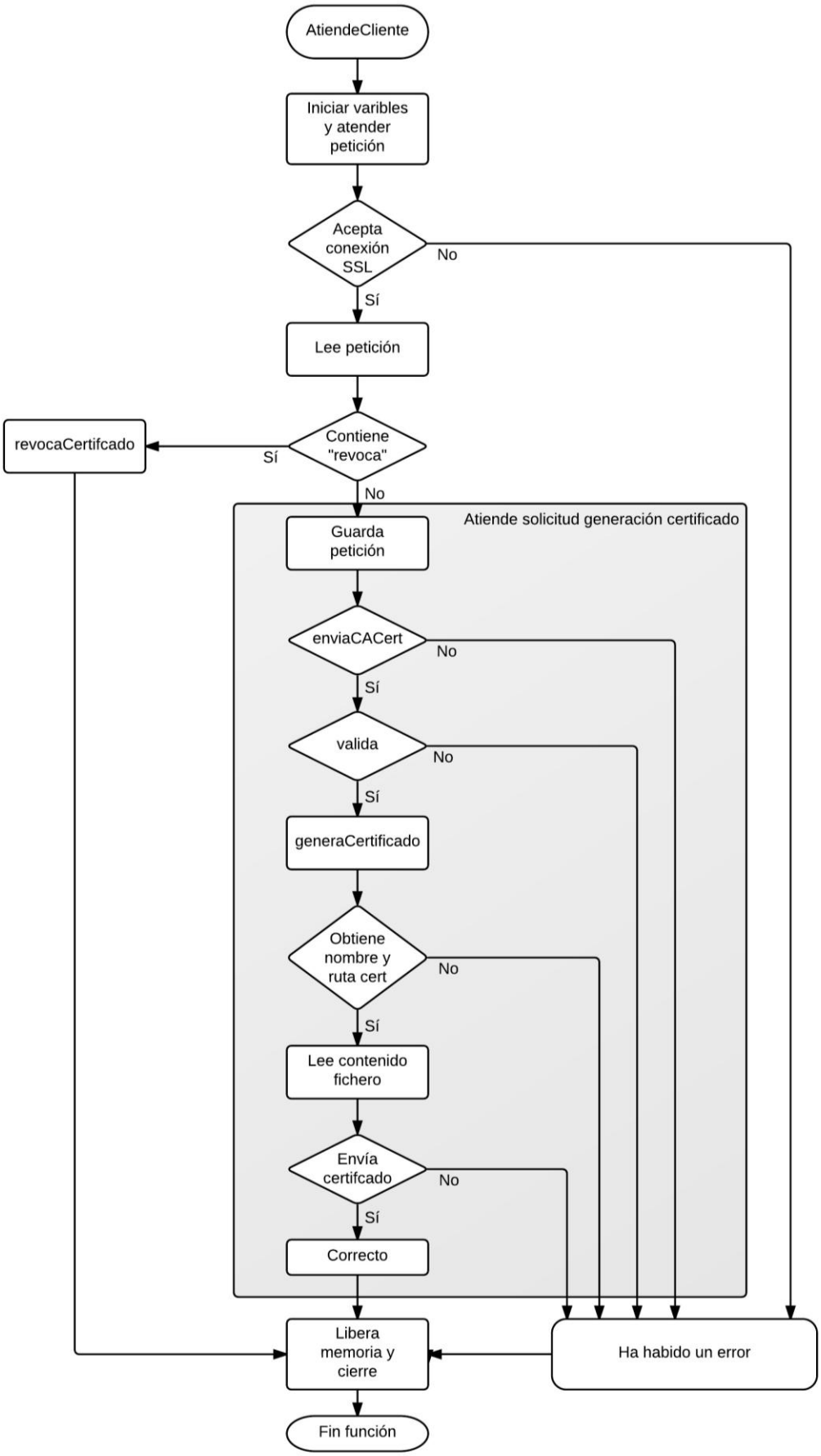

Figura 5-3. Diagrama función atiendeCliente de servicioCA.c

Esta función recibe como parámetro el puntero a SSL con el estado de la conexión y no devuelve ningún valor. Lo primero que se hace es declarar las variables a utilizar y reservar memoria dinámica para los buffers que van a ser necesarios para tratar el envío y recepción de información. Se va a disponer además de un fichero de salida donde se escribirán las salidas de las diferentes acciones y comandos invocados.

Una vez que se ha aceptado la conexión, se procede a comprobar si la petición incluye la palabra "revoca", lo que indica que se solicita la revocación de un certificado. Si es así, se llama a la función *revocaCertificado* pasándole de nuevo el estado de la conexión para que pueda seguir intercambiando mensajes.

En el caso de que no se solicite una revocación, el mensaje recibido va a ser una petición, por tanto la almacenamos. Antes que nada, se le manda el certificado público de la CA al servidor, haciendo uso de la función *enviaCAcert*. Luego se procede a validar el servidor que hace la solicitud (llamando a la función *valida*), y si el resultado es satisfactorio, se invoca al script *generaCertificado.sh*, que realizará las acciones que sean necesarias para crear un certificado firmado digitalmente por esta autoridad a partir de la petición recibida.

Obtenemos desde el programa en C la ruta y el nombre del certificado generado, y si existe, lo leemos y lo guardamos en uno de los buffers creados. A partir de esto, se envía al otro extremo, terminando la operación con éxito y marcando el envío como correcto. Si en alguno de los pasos anteriores se produce algún error, se saldrá de la función. Se llama a la función *imprimeResumen* que mostrará por pantalla un resumen de los resultados obtenidos, diciendo si se ha completado o no satisfactoriamente. Por último, se liberan todos los recursos, tanto en términos de memoria dinámica y ficheros abiertos como de cierre de la conexión (se libera el estado de la conexión SSL y el socket de la conexión).

5.2.1.2 funcionesAutoridad.c

En este fichero se recogen un conjunto de funciones que son utilizadas desde el fichero *servicioCA.c*. Tiene asociado un fichero de cabecera denominado *funcionesAutoridad.h*.

La función *imprimeResumen* muestra por pantalla los resultados a modo de resumen. Toma como parámetro un entero usado como bandera, que indica si se ha mandado o no correctamente el certificado. No devuelve nada.

En *escribeFichero* hace uso de *fwrite* para escribir en un fichero una cadena de caracteres. Se le pasa como parámetro la cadena de datos y el nombre del fichero donde se va a escribir. Tampoco va a devolver nada.

La función *enviaCAcert* envía al servidor el certificado que incluye la clave pública de la autoridad certificadora. De entrada recibe como parámetro el estado de la conexión SSL y el fichero de salida donde se guardarán lo que escriban los comandos. Básicamente obtiene el certificado guardado en la autoridad y lo envía mediante el uso de *SSL_write* al servidor. Devuelve 1 si se ha realizado con éxito y 0 en caso contrario.

Llegamos a la función *valida*, que se encarga de comprobar si el que solicita el certificado existe de verdad, pidiéndole que coloque en su servidor un fichero. Toma como parámetros el estado de la conexión y el fichero de salida. En primer lugar obtiene el nombre del fichero, así como el contenido (que se generará de forma aleatoria). Almacena en variables ambas informaciones y las envía. Cuando recibe la respuesta del servidor de que ha colocado el fichero correctamente (recibe un mensaje con "ok"), procede a comprobar esto, invocando al comando *validaCA.sh* y descargando el fichero del servidor. Se comprueba si coinciden el contenido del fichero descargado y el almacenado en la variable anteriormente. Se devuelve 1 si se ha realizado con éxito y 0 en caso contrario.

En este archivo también encontramos la función *revocaCertificado*, que se ejecuta si es solicitado por parte del servidor, y que recibe de entrada el estado de la conexión SSL. Obtiene del servidor el certificado que quiere revocar (se va a obtener el número de serie de éste) y se ejecuta el comando revocaCertificado.sh. Si es correcto devolverá 0 (ya que es el resultado de un script), en otro caso el resultado que se mandará será 1.

5.2.1.3 conexionCifradaCA.c

Describe las funciones necesarias para establecer una conexión cifrada con el otro extremo. Como ya se ha dicho se considera que este lado de la conexión va a ser el servidor dentro del modelo cliente-servidor que hemos desarrollado para esta aplicación distribuida. Tiene asociado el fichero de cabecera conexionCifradaCA.h.

La función *iniciaServidorCTX* hace la inicialización del contexto del servidor de la conexión SSL. No requiere parámetros de entrada, y devuelve un puntero a contexto nuevo, de tipo *SSL_CTX**. Haciendo uso de las funciones de la librería *ssl* de OpenSSL se cargan los algoritmos de cifrado (*OpenSSL_add_all_algorithms*) y todos los mensajes de error posibles (*SSL_load_error_strings*). Con esto se crea una instancia del método servidor para TLS en su última versión (*TLSv1_2_server_method*), necesario para generar un nuevo contexto (*SSL_CTX_new*), que será lo que se devuelva como resultado de la función.

En *cargaCertificados* se le pasan como parámetros un puntero a contexto SSL, y las rutas donde se encuentran tanto el certificado como la clave necesaria para la conexión cifrada. Con esto se establece el certificado local indicado (*SSL_CTX_use_certificate_file*) y la clave privada (*SSL_CTX_use_PrivateKey_file*). También se verifican que ambos coincidan, con la función *SSL_CTX_check_private_key*.

5.2.2 Programa servidor

En el lado del servidor solicitante, tenemos la otra parte de esta aplicación distribuida, *clienteCA*, también en C, y que contiene los ficheros *clienteCA.c* (con la función principal), *funcionesServidor.c* (contiene funciones necesarias para el correcto funcionamiento del servidor y su comunicación con la autoridad) y *conexionCifrada.c* (que tiene funciones para establecer la conexión cifrada).

Para la ejecución de este programa y el conjunto de acciones que dispone se ha creado el script *servidor.sh* el cual sirve para la instalación, que descarga e instala en la máquina las librerías necesarias así como compila el código fuente de la aplicación en C, y para el resto de funciones del programa, que son pedir (y configurar) un certificado, ver el certificado configurado, revocarlo o borrarlo del sistema (que incluye la revocación de éste). Al igual que en el script de la autoridad, este tiene dos formas de ser invocado, ya sea mediante comandos independientes o mediante el menú que incluye.

Figura 5-4. Menú principal servidor.sh

5.2.2.1 clienteCA.c

Este archive contiene el main. A diferencia del fichero principal de la autoridad, en este caso solo se incluye la función principal, que será la que haciendo uso de otras funciones de las librerías incluidas realice todo el proceso de comunicación con la autoridad, tanto de solicitud de un certificado firmado, como de revocación. Así mismo se llamarán a los scripts correspondientes para configurar el certificado que nos sea proporcionado en el servidor Apache de forma automática.

Como ya hemos comentado, este programa se centra únicamente en la configuración en el servidor web Apache, por ser el más empleado y motivo de pruebas posteriores que se harán en este trabajo.

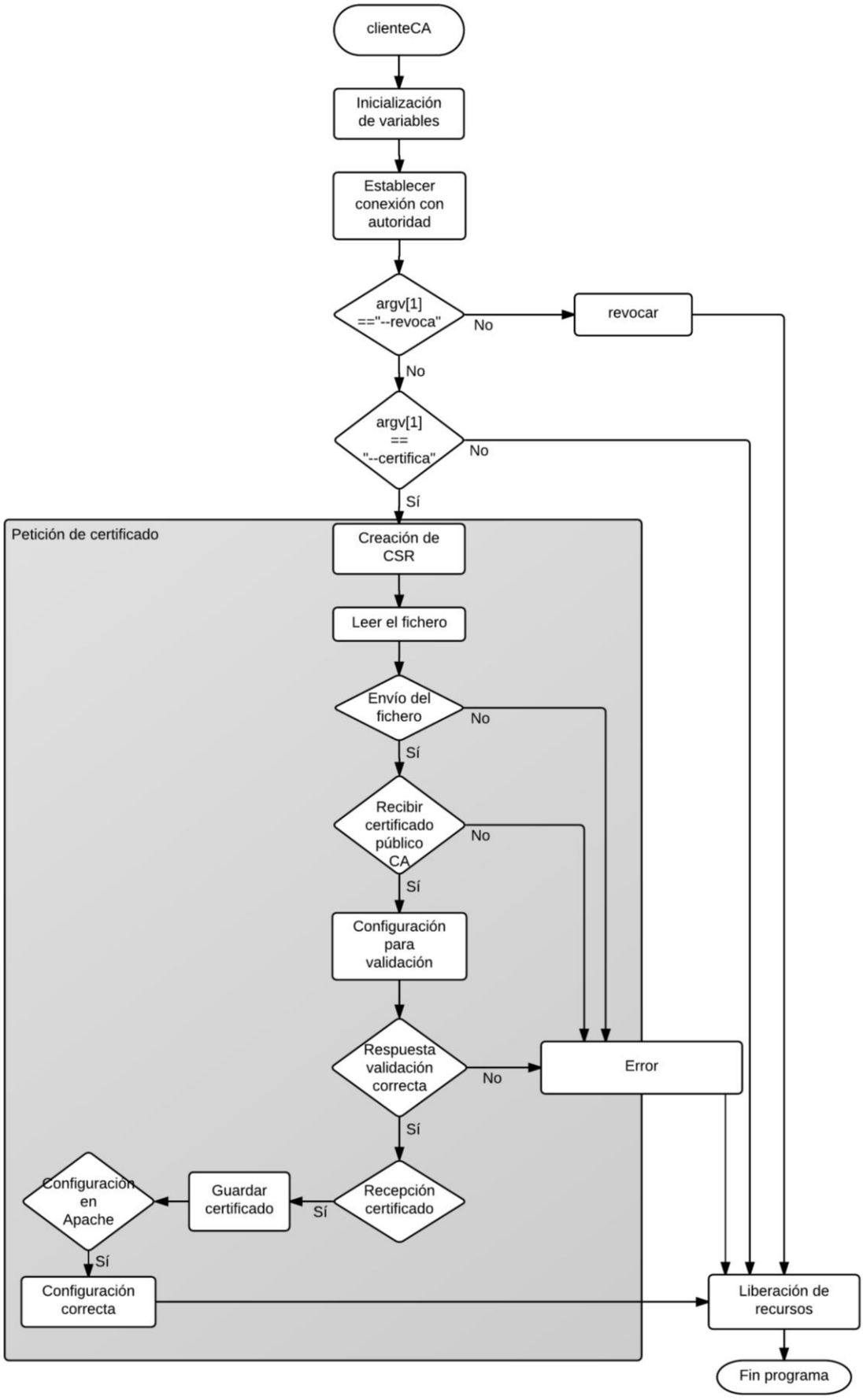

Figura 5-5. Diagrama función main de clienteCA.c

En esta función principal, tras comprobar que el número de argumentos es correcto (Uso: `clienteCA [--revoca]/[--certifica] maquinaCA puerto [maquinapropia]`), la inicialización de variables incluye la reserva de memoria dinámica necesaria para los buffers de lectura y escritura usados en la comunicación.

Para el establecimiento de conexión, se crea el contexto SSL con *iniciaCTX*, se genera un nuevo socket pasándole la dirección y el puerto de la autoridad a la función *conexion*, se crea un nuevo estado de la conexión (*ssl*), se asocia el socket al estado y se desarrolla la conexión cifrada con *SSL_connect*.

Una vez que se han relacionado ambas entidades correctamente, se comprueba si el argumento pasado al programa requiere una revocación (en cuyo caso se llama a la función *revocar*) o requiere una certificación, lo que indica que se debe solicitar y configurar un certificado en la máquina.

En el segundo caso, se crea una petición llamando al script correspondiente, que la generará en un fichero. Dicho archivo será leído y enviado a la autoridad. Si le llega, se espera una respuesta de la CA como es su certificado público, que se guardará con el nombre *cacert.pem* en el directorio en el que nos encontremos.

Una vez recibido, comenzamos con el proceso de validación por parte de la autoridad de que nuestro servidor es correcto y existente. Para ello nos enviará el nombre del fichero que quiere que situemos en él. Si lo recibimos correctamente, se llamará a la función *respuestaValidacion*, que devolverá 1 si todo el proceso se ha realizado con éxito, situando el fichero con el contenido que se nos indique en el servidor Apache y mandando el mensaje de "ok".

Cuando la autoridad lo haya validado, nos enviará el certificado firmado, que lo recibirá este programa y lo guardará según el nombre indicado, haciendo uso de la función *escribeFichero*. Se invoca a un script que comprobará si es correcto el certificado (*compruebaCert.sh*) y si es así pondrá a 1 la bandera que indica una recepción con éxito.

Pero esto no se acaba aquí. Dentro de la funcionalidad de automatización que buscamos se incluye el configurar el certificado en Apache, de forma que se ahorre esto al administrador del sistema, el cual sólo deberá proporcionar permisos de root. Con esto se comprueba si somos este tipo de usuario, y si no lo es, se indica que se introduzca la clave para el acceso. Se invocará al script que va a configurar el certificado en Apache, que devolverá un 0 si todo se ha realizado satisfactoriamente.

Por último, se imprimirán los resultados a modo de resumen, y se liberará la memoria y la conexión establecida.

5.2.2.2 funcionesServidor.c

Se reúnen una serie de funciones para el correcto funcionamiento del servidor (empleados en *clienteCA.c*). El fichero de cabecera asociado es *funcionesServidor.h*.

La función *escribeFichero* es la misma que en *funcionesAutoridad.c* (en la CA), que escribe el contenido de la cadena de caracteres en el archivo con el nombre indicado.

En *imprimeResumen* se muestra por pantalla un extracto de lo realizado, que en este caso incluye una bandera para indicar si se ha hecho correctamente. Además se le pasa como parámetro una cadena que indica la localización del certificado configurado en el caso de que se haya podido hacer. No devuelve nada.

La función *respuestaValidacion* realiza los pasos que exige la CA para que ésta pueda hacer la validación. Tiene como parámetros el estado de la conexión SSL, una cadena que indica el nombre del fichero donde guardar la información y el archivo para escribir la salida de los comandos. Devuelve un entero que dice si se debe continuar (1) o no (0). Tras la declaración de variables y reserva de memoria para el buffer, se recibe de la autoridad el contenido que debe contener el fichero de validación. Lo guardamos e invocamos un script que colocará este fichero en el lugar adecuado para que la autoridad pueda descargárselo. Tras ello, manda a la CA un mensaje de "ok" si el resultado del script es exitoso.

Por último, la función *revocar* se encarga de pedir a la autoridad que el certificado que tiene configurado sea revocado. Como entrada recibe el estado de la conexión SSL, devolviendo un valor entero. Para ello obtiene el número de serie del certificado que está en Apache con el script obtieneSerial.sh, y éste es enviado a la CA para que lo revoque.

5.2.2.3 conexionCifrada.c

Incluye las funciones necesarias para encriptar la conexión desde este lado de la comunicación. Tiene asociado el archivo de cabecera *conexionCifrada.h*.

La función *conexion* recibe de entrada el nombre de la máquina (o dirección IP) y el puerto donde está la autoridad con la que nos queremos conectar. Realiza el proceso de inicialización del socket TCP, devolviendo éste. Obtenemos el nombre de la máquina que se guardará como un tipo *struct hostent* *, y que será posteriormente pasado como dirección de tipo *struct sockaddr_in*. Se crea el socket y se inicia la conexión con el uso de *connect*.

Al igual que en la máquina de la autoridad con la función *iniciaServidorCTX* (recordamos que estamos en un modelo cliente-servidor), en este fichero se encuentra *iniciaCTX*. Esta función inicializa el contexto de la conexión SSL. Es similar a la de la CA, con la diferencia de que el método que se instancia en este caso será mediante la función *TLSv1_2_client_method*.

5.2.3 Scripts en la aplicación

Todo no se basa en esta aplicación distribuida en C. En muchas ocasiones hace uso de scripts para ejecutar funciones del sistema necesarias para la gestión de los certificados o la configuración de Apache. El programa en C se encarga de gestionar e invocar los scripts que son necesarios en el momento adecuado. Caso aparte se pueden considerar los scripts de instalación, que en la máquina de la autoridad son *instalaCA.sh*, *instalaOCSP.sh* (para la instalación del servidor OCSP) y *desinstalaCA.sh*; mientras que en la máquina del servidor es *instala.sh* (ya que lo único que realiza es la descarga de librerías que faltan y compilar el código fuente).

En las siguientes tablas se muestran los scripts que podemos en encontrar en cada una de las entidades.

Tabla 5–1. Scripts de la autoridad

Nombre	Función
instalaCA.sh	Script para la instalación de la CA. Requiere permisos de root.
desinstalaCA.sh	Desinstala la autoridad de certificación.
validaCA.sh	Comprueba el fichero de validación del servidor para ver si éste existe.
generaCertificado.sh	A partir de una petición genera un certificado firmado digitalmete.
revocaCertificado.sh	Revoca un certificado a partir de un número de serie proporcionado.

Tabla 5–2. Scripts del servidor

Nombre	Función
instala.sh	Instala las librerías necesarias y compila. Requiere permisos de root.
creaCSR.sh	Genera la petición del certificado a partir del fichero de configuración.
validacion.sh	Mueve el fichero de validación al directorio del servidor Apache.
soyroot.sh	Comprueba si el usuario que lo ejecuta es root.
compruebaCert.sh	Comprueba que en el certificado recibido coinciden sus claves.
configuraCert.sh	Funciona de puente al script *configCertroot.sh* y pide acceso a root.
configCertroot.sh	Copia certificados en directorio correspondiente y configura Apache.
borraNoUtil.sh	Borra ficheros innecesarios una vez pedido el certificado.
borraCert.sh	Borra del sistema los certificados y deja Apache como estaba.
obtieneSerial.sh	Obtiene serial del certificado configurado en sistema.

Este conjunto de scripts está situado en el directorio *scripts*/ dentro del directorio principal de cada entidad. Además, algunos de éstos hacen uso de una serie de parámetros que están en ficheros de configuración, los cuales pueden ser modificados por el usuario según sus preferencias.

En la CA, el archivo *configCA.conf* incluye los parámetros necesarios para poner en marcha la autoridad, tales como los argumentos para crear el certificado raíz, puerto en el que escucha, ruta de directorios, nombre de certificado a generar y usuario/grupo que van a emplear tanto la autoridad como el servidor OCSP.

El lado del servidor tiene en su directorio principal el fichero *configServidor.conf*, que igualmente permite indicar los parámetros del certificado que se va a solicitar, la ruta del servidor web (donde se situará el fichero de validación), nombre con el que se guardará el certificado solicitado, IP (o dominio) y puerto de la CA.

A continuación se pueden ver los elementos configurables de este fichero en el servidor:

```
#----------------------------
# Parametros de la peticion
#----------------------------
PAIS=ES
PROVINCIA=Sevilla
LOCALIDAD=Sevilla
ORGANIZACION=ESI
UNIDAD_ORG="Web de prueba TFG"

#------------------------
# Ruta del servidor web
#------------------------
RUTA_SERVIDOR=/var/www/

#------------------------
# Dominio o ip de la CA
#------------------------
NOMBRECA=localhost

#---------------
# Puerto de CA
#---------------
PUERTO=61000

#------------------------------
# Nombre certificado a obtener
#------------------------------
CERT="certTFG"
```

En los siguientes apartados se explicarán los diferentes procesos de comunicación que tienen el servidor y la autoridad haciendo uso de los scripts aquí mencionados. Será entonces cuando se detallarán algunos de los más destacados.

5.2.4 Solicitud y generación de un certificado

Hemos analizado el programa por separado en cada una de las entidades. Visto como un conjunto, ambas partes mantienen una comunicación en la que cada uno espera que el otro mande un mensaje con cierta información. Para el caso de que el servidor solicite un nuevo certificado, se manda un CSR y se espera que la autoridad nos mande información de diverso tipo, antes de que nos mande el certificado.

En el siguiente diagrama de paso de mensajes podemos observar cómo es esta comunicación:

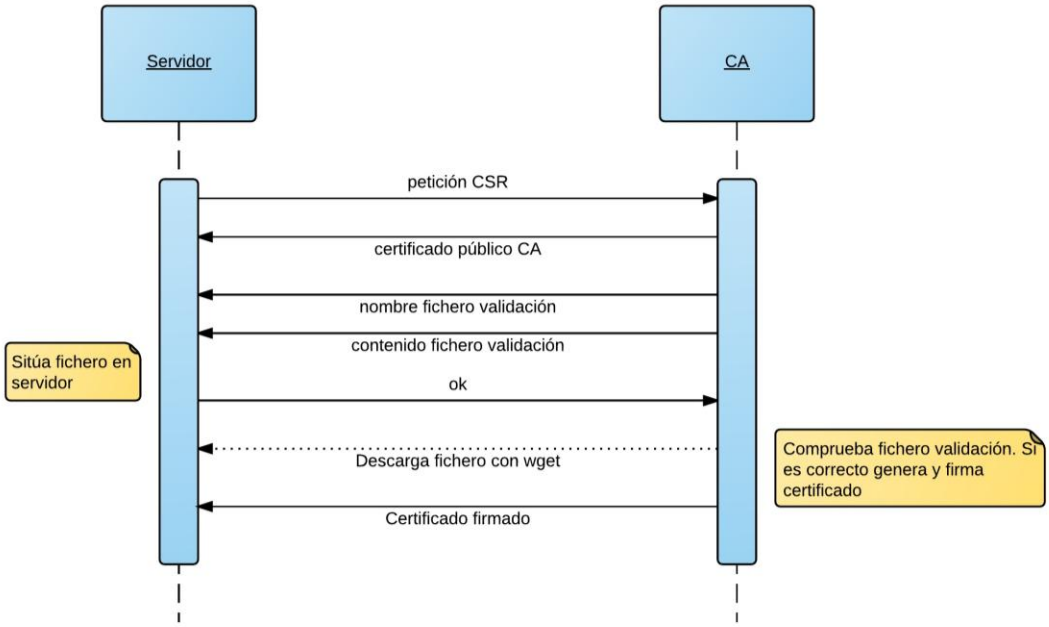

Figura 5-6. Paso de mensajes solicitud de un certificado a la CA

En el lado del servidor, destacar el script *creaCSR.sh*, que es el encargado de realizar la petición de un nuevo certificado a la CA, que genera una clave privada, y con ésta crea el CSR haciendo uso del comando "*openssl req*", en el que se indicarán los campos de la petición tomados del fichero de configuración que está en el directorio principal del servidor.

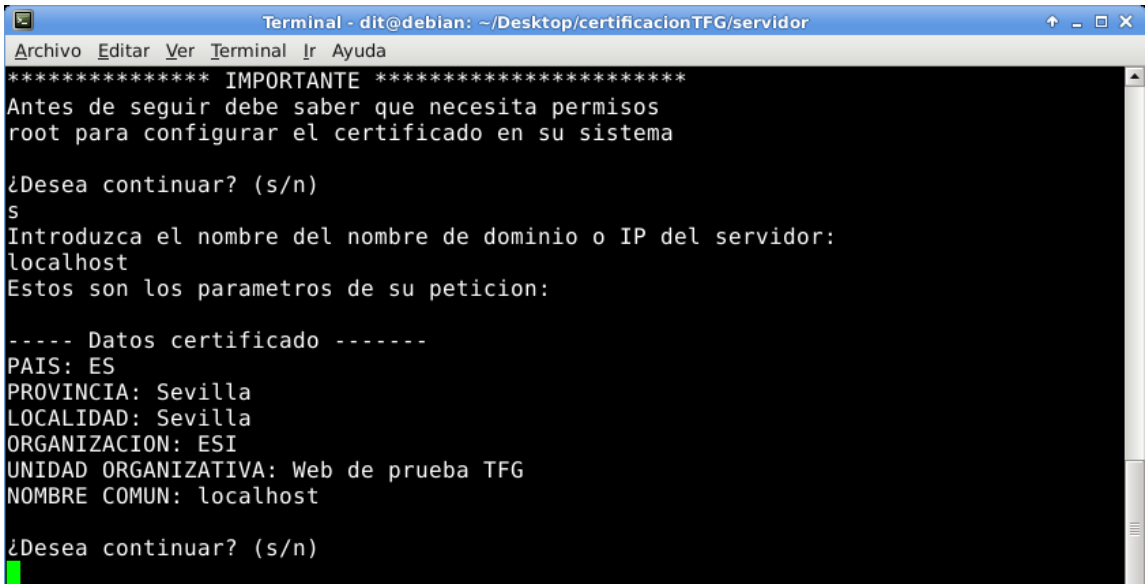

Figura 5-7. Comienzo solicitud de certificado desde menú principal de servidor.sh

Desde el punto de vista de la autoridad, nos centramos en esta ocasión en el script *generaCertificado.sh*. Lo primero que se hace es comprobar si existe la petición y ver si el nombre del certificado indicado en el fichero de instalación existe ya. En ese caso, se añadiría un valor numérico a continuación del nombre.

Después se obtiene de la petición el CN (nombre común) que se solicita. Haciendo uso de la función *valida_ip* incluida dentro del mismo script, se comprueba si es una dirección IP o no. Si lo es, hay que generar el certificado incluyendo en su configuración que el campo *Subject Alternative Name* debe estar relleno con la IP indicada. En caso de que no lo sea, se generará el certificado de forma normal. Todo esto haciendo uso de la herramienta OpenSSL. Una vez generado, se actualiza la CRL en la autoridad y se copia en el servidor OCSP para que éste la tenga disponible.

Pueden verse estos comandos de OpenSSL en la sección 4.2.1.3, donde ya fueron explicados.

A lo largo del proceso se requieren permisos de root para ciertas acciones. Si se ejecuta el programa desde otro usuario, cuando sea necesario se pedirá que se introduzca la contraseña para el acceso al superusuario. Se va a requerir en dos ocasiones, una cuando haya que colocar el fichero de validación en Apache y otra cuando haya que configurar el certificado que ha sido proporcionada por la CA en el sistema y servidor Apache.

Si todo se ha realizado con éxito, debe aparecer algo como la captura que viene a continuación, donde en el resumen de resultados se indica que tanto la recepción como la configuración del certificado se han hecho sin problemas.

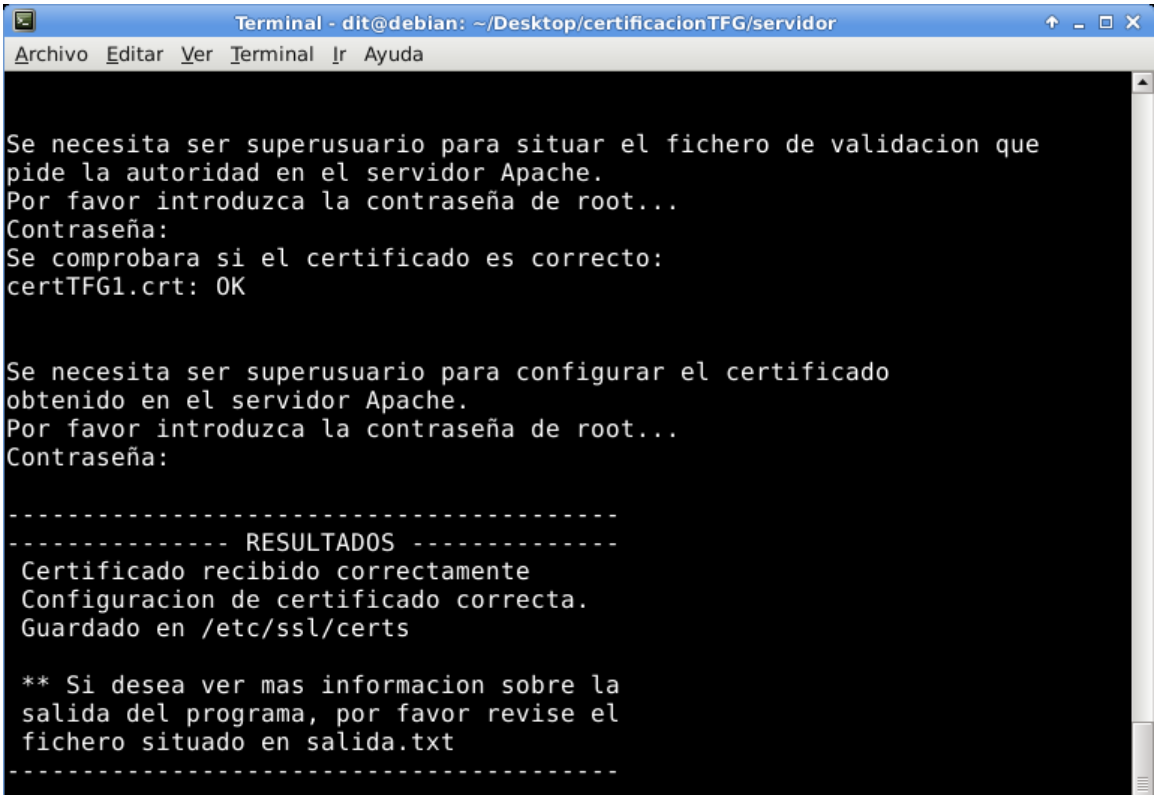

Figura 5-8. Resultado solicitud de certificado correcta

5.2.4.1 Validación del servidor

La autoridad no puede firmar un certificado a cualquiera que se lo solicite, ya que puede ser una petición para un dominio que no existe o para un uso fraudulento. Supongamos que se quiere acceder a *https://ejemplo.com*, y un atacante que se encuentre en medio de la conexión haya solicitado a la autoridad un certificado para dicho dominio. Si la CA se lo da, éste tendrá la capacidad de poder dar al usuario este certificado como si fuera el verdadero servidor, pudiendo obtener y descifrar toda la información.

Para evitar problemas como éste se ha diseñado esta comprobación por parte de la autoridad, que valida al servidor solicitante. La idea surge a partir del proyecto *Let's Encrypt* ya mencionado, donde se pretende poner en marcha una CA de características similares a la de este proyecto. Al no ser presencial, necesita alguna prueba para validar el servidor que pide el certificado. Para ello, a partir del dominio del servidor, la CA permite elegir entre dos pruebas: proporcionar un registro DNS para el dominio o proporcionar un recurso HTTP mediante una URI conocida por la autoridad. Durante estas pruebas, la autoridad le da al servidor una cadena de bits que será usada sólo una vez y que debe firmar con su clave privada, verificando posteriormente la CA dicha firma.

Basándonos en esto, se pretende que la CA exija al servidor que ponga a disposición para su descarga un fichero determinado. Este archivo es generado por la autoridad, que dentro del programa enviará al servidor tanto el contenido como el nombre. El programa en el lado del servidor lo situará en Apache para que pueda

descargarlo la autoridad. Ésta mediante wget intenta su descarga. Si es correcto y coincide con lo que ella ha mandado, lo valida. Si no es correcto o no puede descargarlo, no continuará y no generará ningún certificado. A todo esto hay que añadirle que dicho fichero estará firmado con la clave privada del servidor para que la CA pueda verificarlo con la clave pública.

En la autoridad, el script *validaCA.sh* permite dos usos a la hora de ser invocado. Por un lado, si se le pasa el argumento "generaNomFich", se crea un nombre aleatorio para el fichero a partir de un resumen de un número aleatorio y guardándolo en *nomfich*, a partir de lo siguiente:

```
echo -n $RANDOM | openssl dgst -sha -binary | xxd -p > nomfich
```

Por otro lado, este script permite realizar la validación del servidor, obteniendo el dominio o IP a partir del CN de la petición recibida. Como primer argumento recibirá la petición y como segundo el nombre del fichero que deberá haberse generado anteriormente. Haciendo uso de wget se descargará el fichero del servidor.

Si se ha recibido correctamente, se obtiene de la petición la clave pública del servidor. A partir de ésta podemos validar si este fichero descargado es correcto (validamos su firma).

Desde el punto de vista del servidor, el script *validacion.sh* firma digitalmente con la clave privada del servidor el fichero y lo sitúa en Apache, de forma que cuando la CA se lo descargue pueda verificar su autenticidad. Se necesitan permisos de root, que se pedirán en el proceso de la solicitud si son necesarios como se ve en la Figura 5-8.

Esto sigue la idea de firma digital en su forma más primitiva, tal y como fue descrito en la Figura 3-3. Se emplea el comando "openssl rsautl" para ello, como vemos en la siguiente línea de código:

```
openssl rsautl -inkey fich/server.key -sign -in $1 -out cifrado.txt
```

5.2.4.2 Configuración en servidor

Si se ha recibido correctamente el certificado solicitado a la autoridad, el programa se encargará de configurarlo convenientemente en el servidor Apache que debe estar ejecutándose en la máquina. Ya se ha comentado que la idea se ha desarrollado para este servidor en concreto por ser uno de los más utilizados de código libre.

Como hay que modificar archivos de la configuración de Apache que se encuentra en carpetas del sistema, se necesita tener permisos de root. Como se puede ejecutar el programa desde cualquier usuario, se ha creado un script, *configuraCert.sh*, que sirve de puente entre el ejecutable en C hasta otro script que hará todos los cambios. Con él se comprueban los parámetros y se construye la invocación del nuevo script con todos los argumentos necesarios. Si no se tienen permisos de root, se pide la contraseña para acceder a él.

El script *configCertroot.sh* se divide en tres grandes bloques. En primer lugar se configura el certificado público de la CA, que debe copiarse al directorio de los certificados en el sistema, por defecto */etc/ssl/certs/*. Además, es necesario crear un enlace simbólico al certificado que tenga como nombre el hash de éste, que se obtiene haciendo uso de la utilidad OpenSSL. Esto es necesario para convertirla en una entidad certificadora de confianza para nuestro servidor web.

El segundo bloque es la configuración del certificado obtenido para el servidor. Lo único que hay que hacer es copiar el certificado al directorio */etc/ssl/certs/* y la clave privada a */etc/ssl/private/*.

Por último hay que configurar Apache para que haga uso del cifrado ssl a partir del certificado obtenido. Lo primero que se realiza es hacer una copia de seguridad de los ficheros de configuración que se van a modificar para volver al estado original si es necesario. Se habilita el módulo *ssl* de Apache y reemplazamos el fichero ports.conf por uno que contiene la configuración necesaria para esto.

En el fichero *default-ssl* de Apache se incluye el certificado y la clave privada que van a ser utilizadas a partir de ahora. Se activa el fichero modificado y se reinicia el servidor, teniendo ya operativo. Si todo se ha realizado con éxito, debe salir algo similar a lo de la Figura 5-8.

5.2.5 Revocación de un certificado

Revocar un certificado permite anular su validez antes de la fecha de caducidad del mismo. Suele emplearse cuando haya sido víctima de algún ataque, habiéndose extraviado o copiado el certificado, o si se ha obtenido su clave privada. Cualquier firma realizada con el certificado tras su revocación hace que dicha firma no tenga validez.

Se ha querido que en el servidor sólo haya que ejecutar también un comando y que el programa también se encargue de todo. Para ello se emplea el mismo programa que para la solicitud, sólo que se indica como argumento que se desea revocar el certificado. Entonces se le enviará a la CA el número de serie del certificado que se ha configurado en el sistema, haciendo uso del script obtieneSerial.sh.

Cuando la autoridad recibe este serial, se invoca al script *revocaCertificado.sh*, que recorre todos sus certificados generados en la CA hasta que coincida con el recibido. Cuando eso ocurre, se revoca el certificado con OpenSSL, y se actualiza la CRL para que el servidor OCSP pueda leer la información. Los comandos empleados para estas acciones se expusieron en la sección 4.2.1.3.

Esto tiene un inconveniente, y es que la autoridad no verifica al servidor que pide la revocación, lo que puede provocar que alguien pueda solicitar revocaciones para diferentes seriales, dejando inválidos estos certificados. Sin embargo, no se ha implementado ya que excede el alcance de este trabajo.

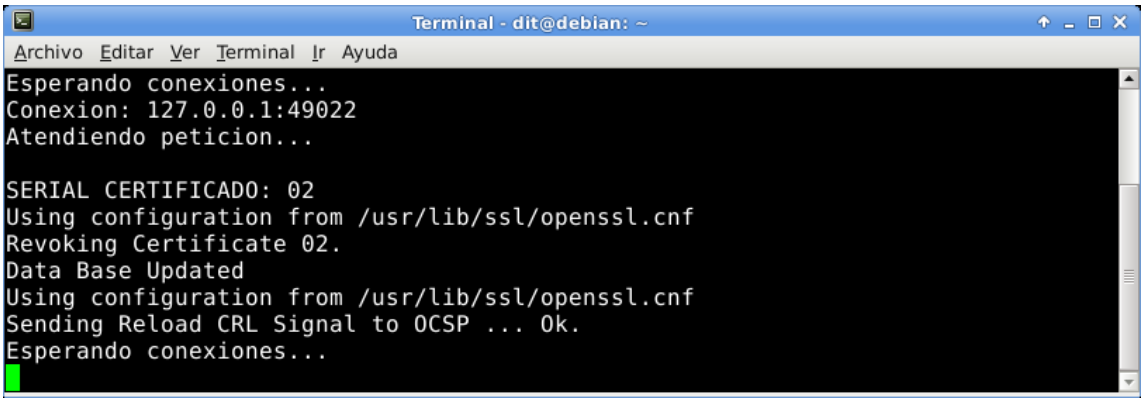

Figura 5-9. Revocación de un certificado en la autoridad

5.2.6 Borrado de un certificado

La idea que se mantiene en el proyecto es la de facilitar la tarea al administrador, de forma que se ha dado la opción de a la vez que se revoca el certificado, eliminarlo del sistema. Puede que parezca peligroso, pero si se revoca se hace con la intención de solicitar un nuevo certificado a la autoridad, y más siendo gratuita la solución ofrecida y de forma automática. El proceso de borrado está pensado para volver a dejar la máquina del servidor en el estado en el que estaba antes de configurar el certificado para SSL/TLS.

El servidor realiza una petición de revocación al igual que en el proceso de revocación, pero además en el propio servidor se realizan las acciones necesarias para el borrado del certificado. Es por ello que en la autoridad el proceso es idéntico al de revocación del apartado anterior.

Para eliminar un certificado del sistema, hay que volver a modificar ficheros de Apache, por lo que se necesitan permisos de root para la ejecución del script que hace las tareas pertinentes. La opción de borrado, al igual que la de revocación, puede ser invocada desde el script *./servidor.sh* (ver anexos). Ésta en particular necesita de él especialmente ya que el borrado se realiza de forma independiente al programa de C, ya que no es invocado por éste.

El script se denomina *borraCert.sh*, y al igual que el empleado para la configuración, tiene tres bloques. El primero de ellos borra del sistema el certificado público de la CA y su enlace simbólico. En el segundo se borra tanto el certificado como la clave privada situados en el directorio por defecto */etc/ssl/certs/*. En el tercer bloque se restauran las copias de seguridad de los ficheros de Apache, se deshabilita el módulo SSL y el fichero *default-ssl* utilizado y por último se reinicia el servidor Apache.

6 SERVIDOR OCSP

Inteligencia es la habilidad de adaptarse a los cambios.

- Stephen Hawking -

Ya tenemos nuestro servidor web configurado para que los clientes que se conecten a él puedan establecer una comunicación cifrada bajo SSL/TLS. Pero no siempre el certificado configurado en el servidor va a tener validez, ya que puede ser que por algún motivo se haya revocado.

El servidor OCSP va a permitir a los clientes comprobar el estado del certificado que ofrece un servidor, para ver si está correcto o no. En este capítulo se va a exponer cómo se ha configurado la utilidad empleada para integrarla en el proyecto junto a la autoridad de certificación.

6.1 Integración de OCSPD en el proyecto

Aunque en el estudio de las tecnologías vimos la autoridad de validación como una entidad independiente de la CA, en este caso vamos a integrar en la misma máquina el servidor OCSP junto con la autoridad certificadora. El motivo que lleva a esto es que queremos algo simple y sencillo, por lo que se ha abordado la idea de que ambas entidades convivan en la misma máquina, teniendo la misma instalación y evitando generar una nueva aplicación distribuida que conecte ambas partes. Además, el hecho de que se encuentren en máquinas separadas no influye en nuestro estudio, ya que lo que se pretende analizar es la comunicación entre servidor y cliente para ver si es segura, sin importar si el servidor OCSP está en la misma máquina que la autoridad o no.

A pesar de ello, la instalación del respondedor OCSP está en un script aparte, que se invocará de forma conjunta con el de la instalación de la CA en *autoridad.sh* pero que podría ser ejecutado en otra máquina siempre que se modificara para que se compartan correctamente los archivos necesarios entre la autoridad y servidor OCSP.

6.1.1 instalaOCSP.sh

Como ya se ha comentado en 4.2.2, la solución elegida para que se pueda consultar el estado mediante el protocolo OCSP es OpenCA OCSPD. En el script *instalaOCSP.sh*, situado en *scripts/OCSP/* dentro del directorio principal donde se está ejecutando el programa, se instala y configura esta herramienta para integrarla junto con la autoridad creada.

Este script comienza con la descarga de librerías necesarias para la correcta instalación de OCSPD, que son: *build-essential*, *libldap-dev*, *libxslt-dev*, *libxml2-dev*.

La utilidad OCSPD se obtiene a partir del código fuente de OCSPD que viene comprimido en el mismo directorio del script, en el fichero *ocspdfuente.zip*. En un primer momento se pensó la idea de que se descargara directamente del sitio web, pero se puso finalmente el archivo comprimido para evitar problemas en la instalación por cambios en futuras versiones.

Se requiere la instalación de LibPKI, también de OpenCA y que incluye una serie de librerías necesarias para OCSPD. Una vez hecho esto, se procede a la instalación del OCSPD, usando el siguiente comando:

```
cd openca-ocspd && ./configure --prefix=/usr && make && make install
```

Si se ha instalado el respondedor correctamente, se configura lo necesario para que pueda ser arrancado. Se generará una petición para solicitar un certificado a la autoridad, que se crea ejecutando un script que proporciona la herramienta OCSPD (*ocspd-genreq.sh*).

Se proporcionan los permisos necesarios de forma que un único usuario que acceda a este servidor, que va a ser en este caso *tfgca* (creado también para ejecutar la CA). En el fichero de configuración de OCSPD (*ocspd.xml*) también hay que indicar el usuario y grupo en las etiquetas <*pki:user*> y <*pki:group*> respectivamente.

Los ficheros de configuración para las CAs soportadas que vienen de ejemplo se eliminarán para añadir un nuevo archivo que incluye la configuración para la CA que se ha creado y que está instalada en la misma máquina (*tfgca.xml*).

Lo siguiente a realizar está diseñado en el script para el caso en el que la autoridad y el servidor OCSP comparten sistema, ya que desde este mismo script se genera el certificado firmado para OCSPD, el cual posteriormente es copiado al directorio del servidor OCSP (por defecto */usr/etc/ocspd/*), así como el certificado público de la CA y la CRL desde la cual recogerá la información necesaria. Insistir en que esto sólo es válido para el caso de que compartan máquina, y que se da por supuesto que el script de instalación se ejecuta de forma conjunta al de la CA, permitiendo ejecutar el comando para generar un certificado.

6.1.2 Inclusión en certificados

Los clientes deben conocer la URI donde se encuentra el servidor OCSP para comprobar el estado de un certificado. Esta ruta está determinada por OCSPD en el fichero *ocspd.xml* y el puerto en el que escucha por defecto es el 2560, pudiéndose modificar también desde ese fichero de configuración.

Para saber esta URI, los clientes deben de obtenerla de alguna forma. Para ello, los certificados tienen una opción que permite incluir esta URI para que sea obtenida por los clientes. En el script *instalaCA.sh* se añaden las siguientes líneas al final del fichero *openssl.conf*.

```
[ usr_cert ]
authorityInfoAccess = OCSP;URI:http://ocsp.tfg.gitt:2560

[ v3_OCSP ]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
extendedKeyUsage = OCSPSigning
```

La clave está en el valor *authorityInfoAccess*, que permite indicar que el acceso se realice mediante OCSP y con la dirección indicada (debe coincidir con la configurada en *ocspd.xml*). Además, se añade la extensión v3_OCSP que incluye nuevos parámetros como son el no repudio, firma digital o el cifrado de claves en las firmas de OCSP. En la generación del certificado que necesita OCSPD para funcionar bien es necesario indicar esta extensión:

```
openssl ca -batch -policy policy_anything -keyfile private/cakey.pem -out
certs/cert.pem -in $RUTA_OCSPD/req.pem -extensions v3_OCSP
```

6.2 Puesta en marcha y pruebas

6.2.1 Arranque del servidor

Tras la configuración correcta del servidor OCSP, el propio script *autoridad.sh* se encarga de ponerlo en marcha, con la invocación del comando:

```
/usr/etc/init.d/ocspd start
```

Si no hay ningún problema, el servidor se estará ejecutando en segundo plano mientras se ejecuta a la vez la autoridad certificadora. En el capítulo anterior se ha comentado varias veces que tanto cuando se genera un nuevo certificado como cuando se revoca hay que recargar la CRL y copiarlas en el directorio del servidor OCSP. Esto es necesario ya que OpenCA OCSPD hace uso de las listas de revocación para la consulta de la información acerca de los certificados.

Esta copia en el directorio de OCSPD hace que se actualice casi al instante, pudiendo considerar tener la información casi en tiempo real. Se dice "casi" ya que esto va a depender del tamaño de la CRL. Si es muy grande, tardará más en copiarse (aunque el tiempo no debe ser muy desorbitado al estar en la misma máquina) y por tanto si se solicita información en ese periodo de tiempo en la copia es posible que no se dé una información exacta. Quitando esta excepción, la automatización de todo el proceso permite que se disponga del estado actualizado en un corto periodo de tiempo.

6.2.2 Comprobación con Firefox

Los navegadores web incluyen opciones para que se valide el certificado mediante OCSP si viene el certificado preparado para ello, es decir, que contenga la URI que marca la dirección donde se encuentra el respondedor.

Para comprobar que el servidor OCSP puesto en marcha funciona lo hemos probado con Iceweasel (Firefox) en otra máquina con Debian 7. Lo primero es entrar en *Preferencias* (desde la pestaña *Editor*), y en *Avanzado* seleccionar la pestaña *Certificados*. Si hacemos clic en Validación se mostrará un cuadro de diálogo como el de la imagen, donde deben marcarse ambas opciones para sacar todo el provecho de OCSP.

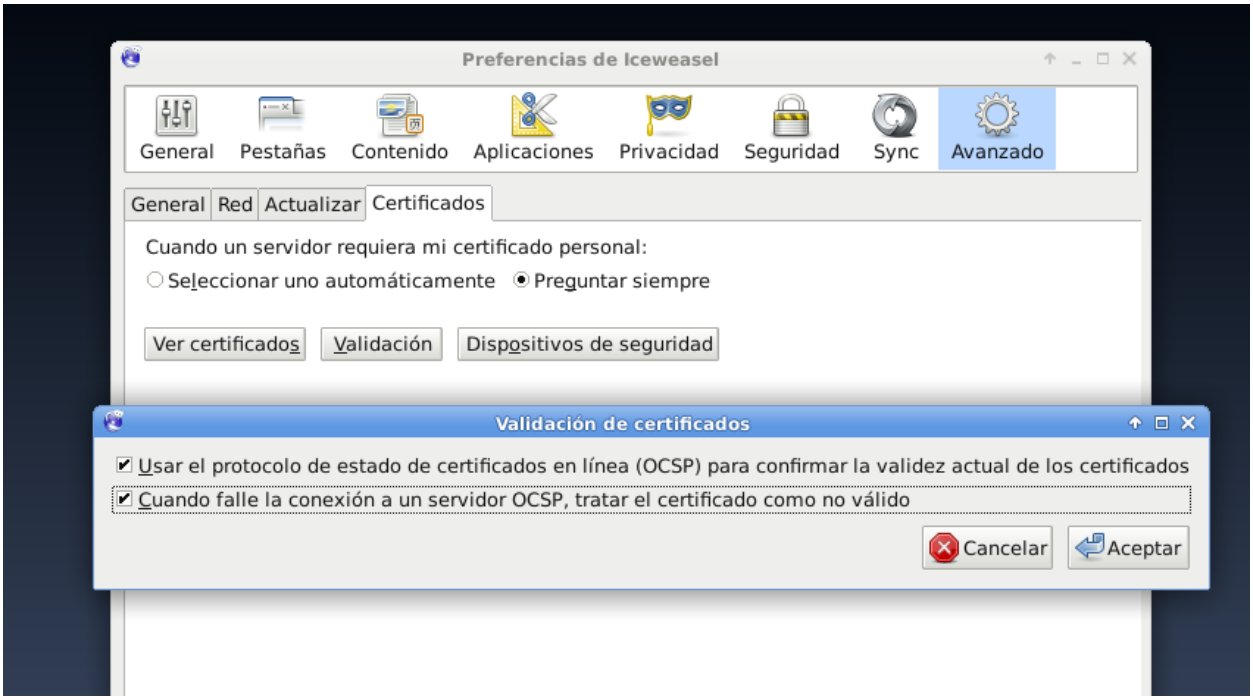

Figura 6-1. Opciones de validación en Iceweasel

Además, es necesario indicarle al navegador la autoridad de confianza que certifica al servidor. Por ello, y suponiendo que ya tenemos descargado en nuestra máquina el certificado público de la CA, en la ventana anterior de *Certificados*, pulsar sobre *Ver certificado*. En la nueva ventana se muestra un listado de los certificados, servidores y autoridades de confianza para el navegador.

Si se hace clic sobre *Importar* de la pestaña Autoridades se puede añadir una nueva autoridad certificadora de confianza. Se mostrará un cuadro de diálogo donde vienen diversos propósitos para los que se puede confiar. Vamos a ponerlo en principio para páginas web.

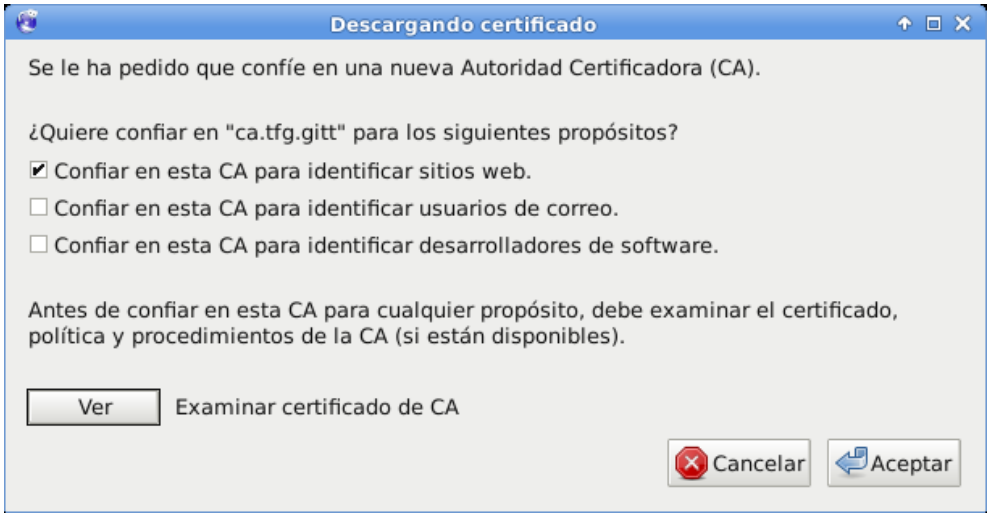

Figura 6-2. Opciones de confianza de nueva CA en Iceweasel

Una vez importado, deberá de aparecer en la lista junto con el resto de autoridades. En esta, si le damos a ver el certificado de la nueva autoridad, deberá verse algo parecido a esto:

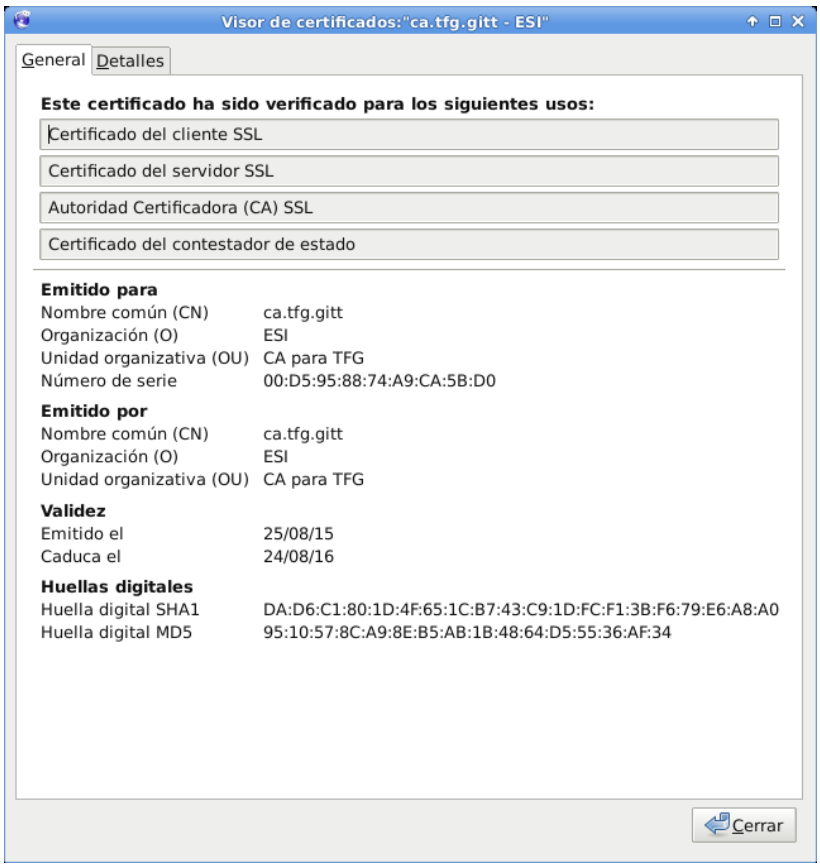

Figura 6-3. Certificado raíz de CA creada visto en Iceweasel

Cuando Firefox confíe en nuestra autoridad, entonces es el momento de acceder al servidor. La validación la realizará internamente el navegador y si es correcta, nos dejará acceder al sitio web sin ningún problema. En el caso de que se encuentre con un certificado revocado, mostrará un mensaje como el de la captura.

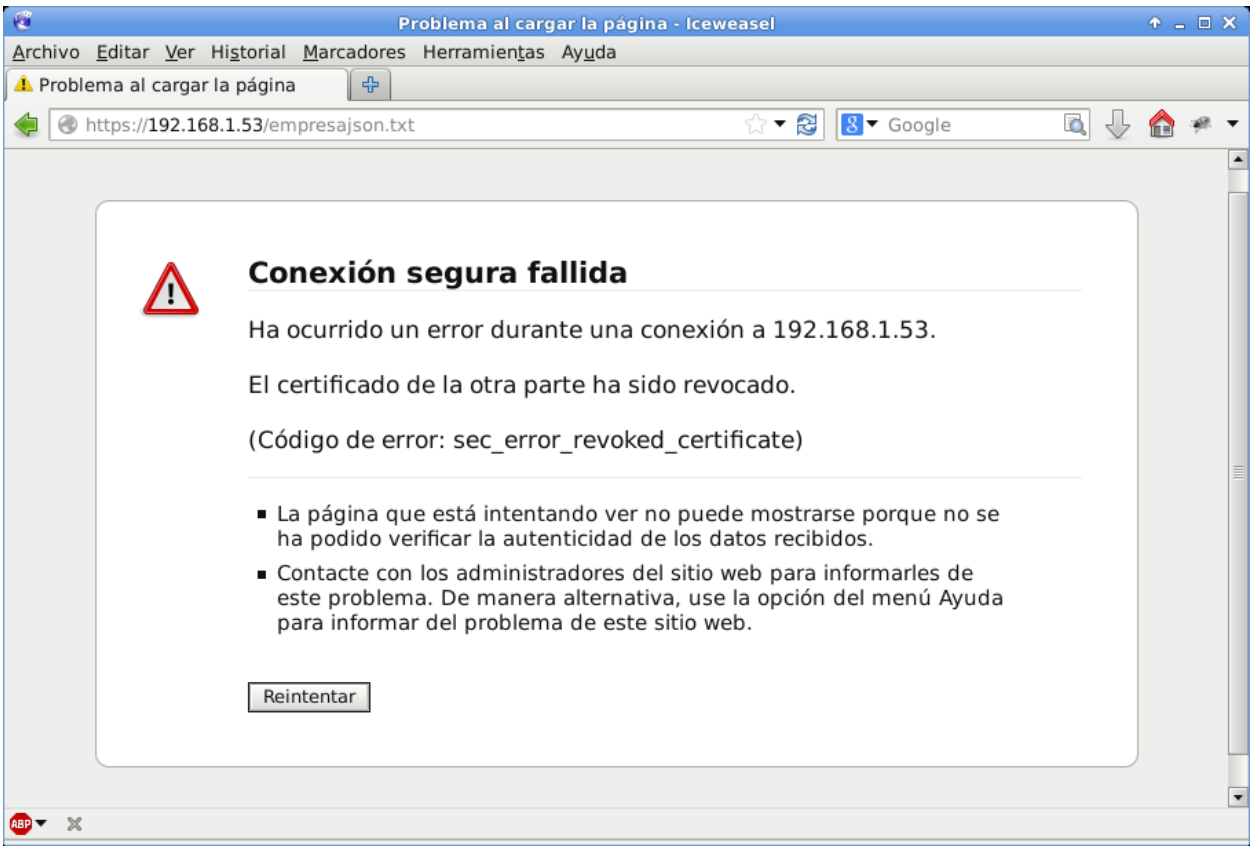

Figura 6-4. Mensaje de error con certificado revocado Iceweasel

6.2.3 Comprobación con OpenSSL

La herramienta OpenSSL para línea de comandos también permite validar un certificado conectándose al servidor OCSP. Además, como ya se vio en la comparativa de soluciones para OCSP, permite poner en marcha su propio servidor. Pero ahora nos centraremos en la validación haciendo uso de la utilidad ya configurada OCSPD. Esto puede resultar de gran ayuda para conexiones que se realicen directamente en scripts, o pruebas en las que es necesario comprobar el estado de un certificado obtenido.

El inconveniente es que es necesario tener descargado en la máquina del cliente que va a realizar esta comprobación el certificado en el directorio desde el que se ejecute el comando. Para esta prueba, se va a realizar desde la máquina del servidor, ya que sólo la vamos a utilizar para comprobar que el servidor que hemos integrado funciona sin problemas. El comando utilizado en esta ocasión es:

```
  openssl  ocsp  -CAfile  cacert.pem  -issuer  cacert.pem  -cert  certTFG.crt  -url
http://ocsp.tfg.gitt:2560
```

La respuesta va a ser satisfactoria si el certificado es correcto, y si posteriormente se revoca veremos que nos indica dicho estado. En la Figura 6-5 se puede observar dos peticiones OCSP, en una de ellas se verá que el estado es bueno y en el otro, tras haber revocado el certificado, se ve dicha revocación y la hora en la que se hizo.

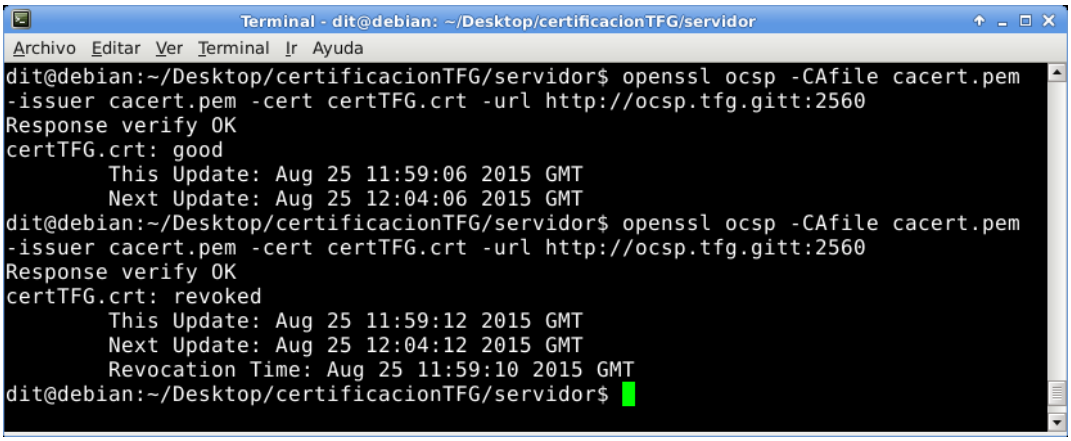

Figura 6-5. Respuestas OCSP con OpenSSL

Con esto se demuestra la rápida actualización del servidor, ya que se copia la CRL en el directorio del servidor OCSP automáticamente tras cada cambio en los certificados de la autoridad. Esto mitiga en la medida de lo posible la carencia de OCSPD que no tiene funciones de tiempo real, ya que obtiene los datos de una lista de revocación que va a estar en la autoridad, y que ésta debe encargarse de pasar al respondedor OCSP.

7 CLIENTE

Vive como si fueras a morir mañana. Aprende como si fueras a vivir siempre.

- Mahatma Gandhi -

El sistema ya está preparado para ponerse a funcionar. Nuestro servidor está configurado correctamente para que escuchar peticiones y establecer conexiones seguras. También tenemos activo un servidor OCSP que dará información a los usuarios sobre el estado del certificado que ofrece el servidor web.

Ahora lo que falta es un cliente, que también debe ser seguro y realizar las comprobaciones necesarias antes de conectarse al servidor. Para ello, haciendo uso del lenguaje de programación Java, se ha diseñado un cliente básico que tiene todas las funciones para la comunicación cifrada y que puede servir como base para construir cualquier otra aplicación más compleja.

7.1 Aplicación Cliente en Java

7.1.1 Clase ClienteHttps

Esta clase conforma un cliente que se conecta a un servidor de forma segura mediante HTTPS. Para ello tiene diversos métodos entre ellos el *main* para poder establecer la comunicación con el servidor. La idea básica que se ha tomado es que en el servidor se haya situado un fichero JSON, que el cliente leerá y mostrará por pantalla en forma de tabla con todos los campos del JSON.

Se ha elegido la lectura de un fichero JSON con la idea de acceder a un archivo que contiene información confidencial que debe transmitirse de forma segura. Esto puede ser un fichero de claves o con datos importantes por ejemplo para una empresa.

El fichero JSON debe estar colocado en el servidor Apache para el acceso desde el cliente. En este caso se ha creado el fichero *empresajson.txt*, que simula la relación de trabajadores de una empresa, que incluye sus DNI y otros datos relevantes.

7.1.1.1 Método principal

En el método *main* lo primero que se hace es comprobar que el número de argumentos mínimo es correcto. Se debe indicar al menos la dirección del servidor al que se conecta y la ruta donde se encuentra el certificado público de la CA (se tendrá que descargar anteriormente en la máquina del cliente).

Además, se le pueden pasar más argumentos que servirán para indicar diversos modos para las pruebas que

haremos posteriormente en la conexión. Si alguna de estas opciones ha sido indicada, se pondrá la variable correspondiente de tipo booleano a *true*.

Estas opciones (junto con una breve descripción de cada una de ellas) son las que siguen:

- --*novalida*: El certificado del servidor no será validado por medio de OCSP
- --*http*: Se fuerza la utilización de http, siendo una comunicación no cifrada
- --*nocompruebacert*: Elimina la comprobación del servidor (que corresponda con la CA indicada y otras pruebas que realiza Java) y toma por válido cualquier certificado.
- --*noTLS*: Fuerza el uso del protocolo SSL

Aclarar que el uso de estas preferencias sólo está indicado para este proyecto por y para el estudio que se va a realizar para ver si la comunicación es segura. Una vez realizadas las pruebas, y si se va a emplear para otro tipo de acciones, estas opciones deberían ser desactivadas.

Para establecer la conexión se utiliza la clase *HttpsURLConnection* (o *HttpURLConnection* en caso de forzarse el uso de HTTP). Si no se ha elegido la opción de desactivar TLS, se configurarán las propiedades de Java para que se utilice éste, ya que actualmente su cifrado es menos vulnerable.

Si se ha elegido la opción de no comprobar el certificado, hay que desactivar las comprobaciones que Java hace por defecto cada vez que se conecta de forma segura a un servidor. Tras ello, se crea el contexto necesario y tras terminar de configurarla se abre la conexión, asociando dicha conexión a un *InputStream*.

En el caso de no seleccionar dicha opción, el método llamará al método *importaCA* que como su propio nombre indica obtiene el certificado de la autoridad que estará en el sistema y lo añadirá para que la aplicación la considere como autoridad de confianza. Tras ello, se abre la conexión añadiéndole el resultado del método anterior, que habrá devuelto un tipo *SSLSocketFactory*. Igualmente, ahora se asocia la conexión a un *InputStream*, desde el que se tomarán los datos.

Luego se validará el certificado (a no ser que se haya indicado el argumento necesario para no hacerlo, en cuyo caso se saltará este paso) que se obtendrá de la conexión establecida. Para ello se llamará al método *validacionOCSP* dentro de la clase *Valida*.

Tras ello se invocará al método *leerJSONdesdeConexion*, que leerá el fichero JSON desde el flujo de entrada que se le pasa por parámetros. Del objeto *JSONObject* que devuelve se obtendrá una cadena que se pasará a *imprimeTodo*, que mostrará todo el JSON en pantalla en forma de tabla.

Figura 7-1. Ejecución normal del cliente

7.1.1.2 Método importaCA

La aplicación java no tiene entre sus autoridades de confianza la que nosotros hemos creado a menos que se lo indiquemos. Se podría realizar de forma externa, es decir, que el usuario antes de ejecutar el cliente lo añada a Java en el sistema, haciendo uso de la utilidad *keytool*. Sin embargo, aquí se busca que todo sea lo más simple y automatizado posible, evitando configuraciones para el usuario. Es por eso que realizamos el reconocimiento de confianza en el propio código, requiriendo únicamente que se le diga dónde está el certificado de la CA.

Este método toma el certificado raíz de la autoridad que estará descargado en el sistema y lo guarda en un almacén de claves. Además genera un socket para establecer conexiones con servidores que hayan sido generados por esta CA. De entrada recibe como parámetro un *String* que será la ruta del certificado raíz. Devuelve una factoría de sockets (*SSLSocketFactory*) con el almacén de claves que contiene el certificado de la autoridad.

Lo primero que se hace es crear un objeto de la clase *KeyStore* (almacén de claves) donde se cargará el certificado de la CA. Una vez añadido, se guardan los cambios en el keystore que se almacenará en el directorio donde se esté ejecutando la aplicación.

Después se genera el socket que hace falta para cargar el keystore creado a la conexión HTTPS, empleando para ello la clase *TrustManagerFactory* para la inicialización de un contexto, con el cual se obtendrá la factoría de sockets que se devolverá como resultado.

7.1.1.3 Métodos leerJSONdesdeConexion, leerTodo e imprimeTodo

El método *leerJSONdesdeConexion* obtiene un objeto JSON a partir de un flujo de una conexión. Como parámetro tiene un flujo de entrada de clase *InputStream* y devuelve un objeto del tipo *JSONObject*.

Convierte el flujo de entrada en un objeto *BufferedReader*, que será la entrada al método *leerTodo*, el cual devolverá un *String*. Con esto se crea un nuevo objeto JSON que será el que se devolverá al método principal.

En *leerTodo* lo que se hace es recorrer todo el buffer en un bucle y se va leyendo cada carácter con *read* y se va añadiendo a una cadena que será el resultado del método.

Cuando se invoca a *imprimeTodo* se le debe pasar como parámetro un tipo *JSONArray*, que interpretará haciendo uso de *getJSONObject*, e irá mostrando por pantalla en forma de tabla, como se ve en la Figura 7-1.

7.1.2 Clase Valida

Esta clase sirve como complemento para una conexión segura. Contiene los métodos necesarios para la validación de un certificado mediante un servidor OCSP.

7.1.2.1 Método validacionOCSP

Realiza todo el proceso de comunicación con el servidor OCSP, a partir del certificado a validar y del de la autoridad certificadora. Como entrada recibe un conjunto de certificados del tipo *Certificate[]* para validar (por lo general solo uno) y un *String* con la ruta donde está el certificado de la CA. Devuelve un valor booleano indicando si se ha terminado con éxito o no la validación.

En un bucle se recorre la tabla de certificados recibido por parámetros. Se comprueba la fecha de validez del certificado (comprobando si no ha expirado). Mediante el método *obtieneOcspUrl* se coge del certificado la dirección donde se encuentra el servidor OCSP.

Se carga el proveedor necesario para realizar la petición OCSP (este proveedor proviene de la librería externa Bouncy Castle) y generamos el id del certificado (número de serie) que servirá para que sea encontrado por el servidor OCSP. Se genera la petición una vez se ha añadido el certificado que se quiere verificar y se establece la conexión con el servidor.

Cuando se recibe la respuesta, se analiza mediante el método *tratarRespuestaOK*, que devolverá un entero que será 0 si el estado es bueno. Este método será invocado siempre que el estado obtenido sea satisfactorio, en caso contrario se devolverá un mensaje de error según el estado que indique el respondedor OCSP.

7.1.2.2 Métodos obtieneOcspURL, tratarRespuestaOK y fullStream

El método *obtieneOcspURL* obtiene de un certificado la dirección URL donde se tiene que validar el certificado mediante OCSP. Se le pasa como parámetro el certificado X.509 que contendrá dicha información y devolverá una cadena con la URL conseguida.

Para ello se busca en el certificado el valor de la extensión correspondiente a *authorityInfoAccess*, que contendrá la información que buscamos.

Con *tratarRespuestaOK* se lee la respuesta del servidor OCSP mediante *getResponseObject* y en función del resultado, se devolverá un valor entero que servirá para indicar los tres estados posibles que se pueden obtener: *good* (devolverá 0), *revoked* (devolverá 1), *unknown* (devolverá 2).

Por último, *fullStream* es un método que devuelve un flujo *InputStream* a partir de una cadena que se le pasa como parámetro. Será utilizado en el método *obtieneOcspURL*.

7.2 Script cliente.sh

Es un script que se ha utilizado para realizar pruebas durante el diseño del cliente (ya que se ha modificado varias veces) y facilita la ejecución de la aplicación. No es obligatorio su uso, además de tener reducidas sus funciones, no permitiendo la mayoría de opciones que tiene el cliente para poder indicárselas mediante línea de comandos.

Lo primero que hace es comprobar que el certificado público de la autoridad está en el mismo directorio desde el que se está ejecutando este script. Si no es así no se podrá continuar.

Luego busca el fichero .jar donde está comprimido el proyecto del cliente (ambas clases anteriormente expuestas). Si no lo encuentra, lo descomprime ya que normalmente estará comprimido a su vez en un zip.

Comprueba si se ha indicado como parámetro la dirección a la que se debe conectar (donde se encuentre el servidor). Si no se ha indicado nada se conectará a la por defecto, que estará en localhost.

Con esto invoca al ejecutable, de la siguiente manera:

```
java -jar ClienteHttps.jar $URL_SERVIDOR $CACERT
```

Esta invocación es la misma que si se hace de forma independiente al script, especificando la dirección del servidor y la ruta del certificado raíz de la CA, así como la posibilidad de añadir las opciones comentadas en 7.1.1.1.

8 PRUEBAS DE SEGURIDAD Y ATAQUES MITM

Una computadora puede ser llamada "inteligente" si logra engañar a una persona haciéndole creer que es un humano.

- Alan Turing -

Una vez que hemos montado toda la infraestructura completa, incluyendo el cliente que solicitará los servicios del servidor, se ha de comprobar que la comunicación entre estas dos entidades es segura.

La mejor forma de hacer esto es a prueba de fallos, haciéndole cambios y pasándole una herramienta que demuestre que no es vulnerable. En este capítulo se detallarán las pruebas a realizar así como las diferentes vulnerabilidades que se intentan evitar.

8.1 Validación OCSP del cliente

En el capítulo 6 se vio cómo se podía validar el certificado del servidor mediante Firefox y OpenSSL. En esta ocasión nos centramos en el cliente diseñado, que como ya hemos comentado tiene funciones para comprobar el estado del certificado.

Todo se basa en la clase *Valida* del cliente, que contiene la función que tomará del certificado la URL para comprobar el estado de éste. Se ha diseñado de forma que si se le pasa como argumento la opción *--novalida*, se salta esta validación, quedando el certificado sin comprobar por el cliente.

El problema que acarrea que no se valide es que si el administrador del servidor ha revocado el certificado por cualquier motivo (puede ser que hayan sido comprometidas sus claves), el cliente seguirá teniendo ese certificado y si no se valida accederá al servidor de forma insegura. Por ejemplo si las claves han sido robadas, el atacante podría descifrar la información.

En esta prueba se van a realizar dos casos diferentes. Uno en el que se va a validar el servidor haciendo uso del funcionamiento normal de la aplicación cliente. El otro caso será con la opción para no validarlo, viendo su vulnerabilidad. En ambos, se probará con un certificado en buen estado y otro revocado.

Si se sigue el curso normal de la ejecución del programa, en el caso de que el certificado esté en buen estado no va a dar ningún problema y se mostrará la tabla del fichero JSON que lee. En el caso de usar *--novalida* también se va a mostrar dicha tabla. En este caso no pasa nada ya que el certificado es correcto.

Pero si se revoca el certificado, como ya hemos dicho, si no se valida correctamente puede convertirse en un blanco fácil para los atacantes. Con la ejecución en la que se valida con OCSP, no se accede al servidor al detectar que está revocado, como se puede observar en la Figura 8-1.

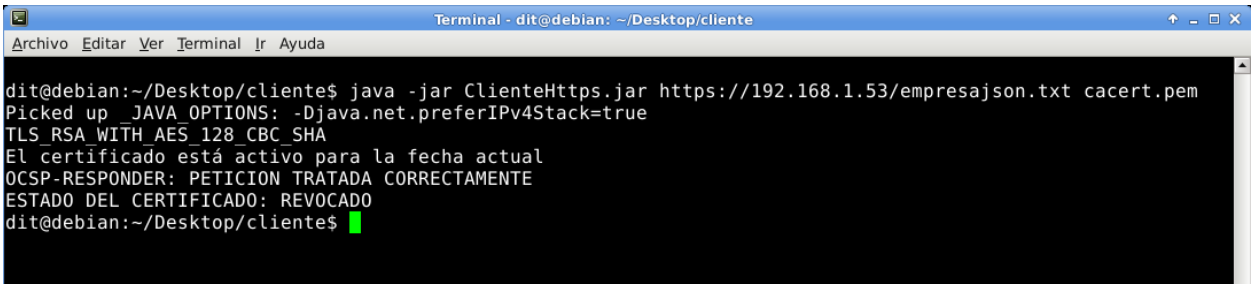

Figura 8-1. Ejecución cliente con certificado revocado

Para cuando añadimos --*novalida* se ve que se accede al servidor mostrando la tabla. Esto no debería ser así al estar revocado, porque significa que el administrador del sistema por algún motivo no quiere que se utilice ese certificado para autenticar al servidor.

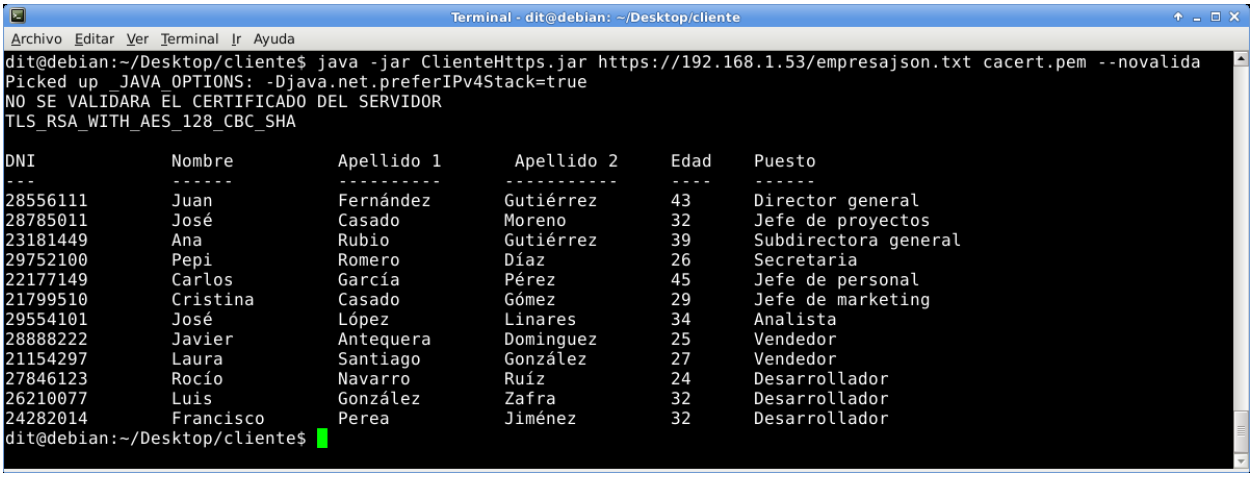

Figura 8-2. Ejecución cliente opción --*novalida* con certificado revocado

8.2 Ataques MITM con nogotofail

Nogotofail es una utilidad que Google ha puesto a disposición de los desarrolladores para que se pueda comprobar si la conexión que se está creando es segura. Para ello, como ya se expuso en la sección 4.3, esta herramienta se coloca en el medio de la comunicación entre cliente y servidor e intenta encontrar vulnerabilidades haciendo diversos ataques.

Se trata de poner a funcionar el servidor, y para cada petición del cliente, introduce aleatoriamente un ataque. También permite configurarlo para que se ejecute un ataque en concreto. Esto es lo que se va a hacer ahora para analizar cada uno de los posibles ataques y ver si es vulnerable.

En el ejemplo el cliente va a tener la IP 192.168.164.140 y la del servidor va a ser 192.168.1.50. Por tanto, la URL a la que se accederá es *https://192.168.1.50/empresajson.txt*.

Se van a emplear dos aplicaciones clientes diferentes para las pruebas. Por un lado el cliente que hemos creado en Java, que accede al servidor para leer el JSON e interpretarlo, y por otro lado mediante wget, que va a ser un cliente que al conectarse al servidor descargará el fichero JSON en la máquina que lo solicite. Además, para demostrar que aún existen clientes vulnerables, se estudiará el caso del ataque *heartbleed* para curl.

Esto se hace para comprobar que el servidor es seguro con distintos clientes que acceden a él. Aun así, que el servidor sea seguro no significa que la conexión no sea vulnerable, ya que intervienen ambos extremos de la comunicación.

El servidor *nogotofail* requiere una instalación que se puede ver en el Anexo B. Se ha instalado en una máquina Ubuntu y se pondrá en modo proxy, de forma que todo lo que salga o vaya hacia el cliente debe pasar por él, simulando el ataque MITM.

Vamos a detallar cómo se deben ejecutar las diferentes pruebas, analizando los resultados obtenidos en el próximo capítulo.

- **droptls**: Tira las conexiones TLS pero deja las SSLv3.

  Con este ataque se desactiva las conexiones TLS, lo que hace que se llegue a la versión 3 de SSL. Esta versión es vulnerable a numerosos ataques, como por ejemplo POODLE, que es capaz de llegar a descifrar la información con esta versión, y que supuso la obsolescencia de SSLv3.

  La ejecución del servidor *nogotofail* con esto es:

```
  python –m nogotofail.mitm --mode socks --port 8080 -A droptls -p 1 --serverssl
server.crt -l log.txt
```

- **selfsigned**: Intenta un MITM usando un certificado auto-firmado para el dominio requerido.

  Un certificado auto-firmado no puede considerarse de confianza, a no ser que lo incluyamos como servidor de confianza. En este caso, como se ha construido la autoridad que firma los certificados a las máquinas que lo soliciten, sólo se va a configurar la autoridad como de confianza. Para que un certificado de un servidor pueda ser aceptado tiene que estar firmado por dicha CA, que anteriormente se habrá encargado de confirmar que el solicitante es quien dice ser.

  Si nos viene un certificado auto-firmado no podemos asegurar que sea de dicho servidor, ya que podemos ser víctimas de un ataque MITM que haya firmado su propio certificado y nos diga que él es el servidor, pudiendo descifrar toda la comunicación.

  Existe una incompatibilidad entre *nogotofail* y la aplicación Java, que reside en que el servidor atacante genera el certificado en formato PEM en el que se incluyen en el mismo archivo tanto la clave privada como el certificado en sí, mientras que Java espera un formato DER con archivos separados.

  Es por ello que en *nogotofail* se ha sustituido el certificado generado con la herramienta (situado en */tmp/*) por un certificado propio generado con OpenSSL. En el cliente creado se ha añadido la opción *--noCompruebaCert*, con la cual la aplicación no verifica que el certificado pertenece a una autoridad de confianza, dando por bueno cualquier certificado. La creación del certificado para sustituir al de *nogotofail* se realiza de la siguiente manera:

```
  openssl req -x509 -nodes -newkey rsa:2048 -keyout clave.pem -out cert.pem -days 365
```

  También se ha probado con el cliente wget. Para ello se comprueba su buen funcionamiento que impide la conexión al servidor si se ejecuta normalmente, y que logra acceder si no se comprueba el certificado. Para hacer esto hay que añadirle la opción *--no-check-certificate*.

  Su puesta en marcha es la siguiente:

```
  python –m nogotofail.mitm --mode socks --port 8080 -A selfsigned -p 1 --serverssl
server.crt -l log.txt
```

- **clientheartbleed**: Envía un mensaje heartbleed al cliente durante el *handshake* de SSL.

  Heartbleed es un agujero de seguridad en OpenSSL que permite a un atacante leer la memoria de un servidor o cliente (es el que se comprueba en este caso). La vulnerabilidad se encuentra en una función encargada de gestionar mensajes *heartbeat*, denominados *keep-alive*, es decir, se usan para informar al servidor de que seguimos "vivos" para que no cierre la conexión. Estos mensajes podían ser modificados, mintiendo en la longitud, lo que permitía que se rellenara el paquete con más información de la cuenta, que se encuentre en posiciones de memoria posteriores a la que nos encontramos. Podría ser que nos encontráramos con claves privadas o usuarios y contraseñas que están usando el servidor.

Al enviar *nogotofail* un mensaje de este tipo al cliente comprueba si realmente es capaz de aceptarlo sin detectar el problema, poniendo en grave riesgo su seguridad.

La ejecución para el servidor *nogotofail* es:

```
  python –m nogotofail.mitm --mode socks --port 8080 -A clientheartbleed -p 1 --
serverssl server.crt -l log.txt
```

- **serverkeyreplace**: Prueba para clientes vulnerables a la sustitución de la clave SSL del servidor.

  Con esta sustitución se puede realizar un típico ataque "Man In The Middle" donde el atacante actúa como cliente para el servidor web, y actúa como servidor para el cliente original. Así es capaz de interceptar toda la conexión, ya que ha cambiado la clave del servidor por una suya propia.

  Su puesta en marcha es:

```
  python –m nogotofail.mitm --mode socks --port 8080 -A serverkeyreplace -p 1 --
serverssl server.crt -l log.txt
```

- **anonserver**: Intenta una conexión MITM que acepta un servidor anónimo o no autorizado.

  Su funcionamiento en la máquina de *nogotofail* es:

```
  python –m nogotofail.mitm --mode socks --port 8080 -A anonserver -p 1 --serverssl
server.crt -l log.txt
```

- **dropssl**: Tira las conexiones SSL.

  Al igual que el ataque *droptls*, tira las conexiones de este tipo, pero en este caso no deja ninguna disponible como ocurría con TLS.

  El arranque del servidor nogotofail para esto es:

```
  python –m nogotofail.mitm --mode socks --port 8080 -A dropssl -p 1 --serverssl
server.crt -l log.txt
```

- **superfishmitm**: Intenta un MITM usando la CA comprometida Superfish.

  Superfish es un componente de software que ha sido calificado como un "ataque de intromisión" que introduce numerosas vulnerabilidades en los clientes. Por ejemplo, Lenovo lo ha permitido preinstalar en numerosos equipos, lo cual supuso un gran problema de seguridad. El problema es que está preinstalado con autoridad de certificado raíz, lo que permite suplantar cualquier certificado de seguridad de los servidores.

  Con esto se genera un certificado cuya CA es ésta, para comprobar si se permite o no su acceso.

  Sigue la misma idea que el ataque *selfsigned* al proporcionar un certificado generado por *nogotofail*. Por tanto, se puede ver correctamente el fallo con wget y su argumento *--no-check-certificate*, pero no se va a poder observar la vulnerabilidad explotada con el cliente java por la incompatibilidad ya mencionada.

  Se ejecuta de la siguiente forma:

```
  python –m nogotofail.mitm --mode socks --port 8080 -A superfishmitm -p 1 --serverssl
server.crt -l log.txt
```

- **invalidhostname**: Intenta un MITM usando un certificado válido para otro dominio.

  Para este ataque es necesario tener en el directorio del servidor *nogotofail* un certificado válido y su clave privada (almacenados de forma conjunta en *trusted-cert.pem*). Este certificado tendrá que haber sido firmado por una autoridad de confianza (en este caso se ha firmado con la que se ha creado).

  Se pone en marcha el servidor de la siguiente manera:

```
  python –m nogotofail.mitm --mode socks --port 8080 -A invalidhostname -p 1 --
serverssl server.crt -l log.txt
```

- **earlycss**: Prueba la vulnerabilidad early CSS de OpenSSL (CVE-2014-0224).

  El problema se encuentra en el mensaje de cambio de la especificación de cifrado (CCS) y genera datos de claves en un tiempo inapropiado que puede ser utilizado para obtener los datos.

  Para su funcionamiento, ejecutar en la máquina de *nogotofail* lo siguiente:

```
  python –m nogotofail.mitm --mode socks --port 8080 -A earlycss -p 1 --serverssl
server.crt -l log.txt
```

Además de estos ataques, *nogotofail* tiene una serie de manejadores (*handlers*) que analizan el tráfico de datos en busca de tráfico vulnerable. En la siguiente tabla se recogen todos los que incluye.

Tabla 8–1. Manejadores de nogotofail que buscan vulnerabilidades en tráfico

Nombre	Función
imapstarttlsstrip	Suprime STARTTLS en IMAP
httpauthdetection	Detecta cabeceras de autorización en peticiones HTTP
imagereplace	Reemplaza respuestas con Content-Type de image/* por ./replace.png
customrequest	Detecta cliente que especifica expresiones regulares en peticiones
weaktlsversiondetection	Detecta versiones de los protocolos TLS/SSL que se sabe que son débiles
insecurecipherdetection	Detecta cifrados inseguros en Client Hellos de TLS
blockhttp	Bloquea el tráfico HTTP
sisablecdcpencryption	Desactiva Chrome Data Compression Proxy encryption
sslstrip	Ejecuta sslstrip en tráfico HTP. Detecta cuándo se visitan las urls de sslstrip
httpdetection	Detectan peticiones HTTP en texto plano y advierte sobre ellas
xmppstarttlsstrip	Suprime STARTTLS en flujos XMPP
rawlogger	Registra tráfico raw
androidwebviewjsrce	Detecta Android Webview Javascript RCE
xmppauthdetection	Detecta credenciales de autenticación en tráfico XMPP
smtpstarttlsstrip	Suprime STARTTLS en SMTP
smtpauthdetection	Detecta credenciales de autenticación en tráfico SMTP
imapauthdetection	Detecta credenciales de autenticación en tráfico IMAP

De todos estos, destacar el manejador encargado de la detección de HTTP, ya que se ha incluido una opción en el cliente Java que permite solicitar la petición en HTTP, de forma que se compruebe que esto es vulnerable y detectable para cualquier atacante que se encuentre en el medio.

Lo mismo ocurre con el protocolo TLS, el cual con la opción *--noTLS* puede ser desactivado, dejando tanto un cifrado como un protocolo asociado a SSL. Esto es lo que por defecto tiene Java, lo cual ha tenido que ser

modificado para que se use TLS si no se indica nada en los argumentos. En este proyecto se ha usado la siguiente configuración:

```
System.setProperty("https.protocols", "TLSv1");
System.setProperty("https.cipherSuites", "TLS_RSA_WITH_AES_128_CBC_SHA");
```

Ya se ha comentado que TLS es actualmente más seguro que SSL, ya que de este último han sido publicadas numerosas vulnerabilidades. Algunas, como por ejemplo el ataque POODLE, ha supuesto la derrota total de alguna de sus versiones.

En *nogotofail* no es necesario seleccionar el manejador que quiere detectar ya que por defecto analiza el tráfico buscando todas las vulnerabilidades posibles. Si se desea estudiar alguna en concreto, se puede seleccionar el manejador de datos con la opción -D y puede ser combinado con cualquiera de los ataques anteriormente analizados. A continuación se muestra un ejemplo para detectar el tráfico HTTP existente en la conexión.

```
python –m nogotofail.mitm --mode socks --port 8080 -D httpdetection -p 1 --serverssl
server.crt -l log.txt
```

9 RESULTADOS

Hago las cosas lo mejor que puedo, lo mejor que sé, y así será hasta el final.

- Abraham Lincoln -

Tras la puesta en marcha de toda la infraestructura y el estudio de la conexión segura entre un cliente y un servidor web a partir de una serie de pruebas realizadas, se han de analizar los resultados obtenidos de éstas, que servirán para determinar si se ha alcanzado el nivel de seguridad deseado, es decir, que no sea vulnerable a las amenazas ya conocidas actualmente.

La prueba referente a la validación OCSP no se va a recoger aquí ya que se han mostrado las capturas correspondientes en el capítulo anterior, observándose que si no se valida correctamente puede accederse con un certificado revocado, lo que supondría un problema de seguridad. Pero el cliente java está configurado para realizar sin problemas esta verificación, impidiendo el acceso si el estado del certificado no es bueno.

Para el resto de pruebas, se va a hacer un repaso por todas comentando los resultados obtenidos para cada una de ellas.

9.1 Respuestas ante ataques

Se verán las capturas desde el punto de vista del cliente, verificando si nos deja acceso libre y vulnerable o tiene algún bloqueo ante un ataque. Cuando hacemos uso de la herramienta *nogotofail*, su servidor MITM también proporciona una serie de información, tanto por pantalla como un registro en un fichero si se le indica. Este registro puede verse en el Anexo D.

Se va a seguir el orden de ataques para *nogotofail* que se vio en la sección 8.2.

9.1.1 Exclusión de conexiones TLS

Si probamos con el cliente java el ataque *droptls* de la herramienta, observamos que salta una excepción indicando error interno. Esto se debe a que se encuentra desactivada la versión SSLv3 del programa, evitando que se exploten vulnerabilidades por conectarse con esta versión.

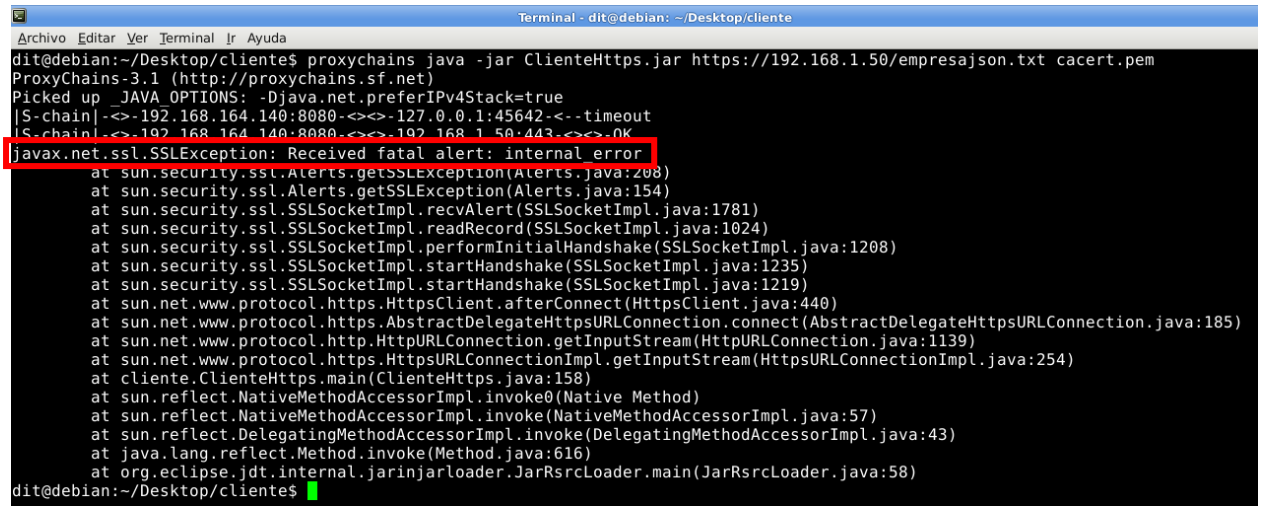

Figura 9-1. Ejecución cliente java ante ataque “droptls”

Lo mismo ocurre si se intenta acceder al servidor mediante wget. Recordar que hay que decirle a wget que utilice el certificado de la autoridad que hemos creado (*cacert.pem*) para que reconozca el certificado del servidor.

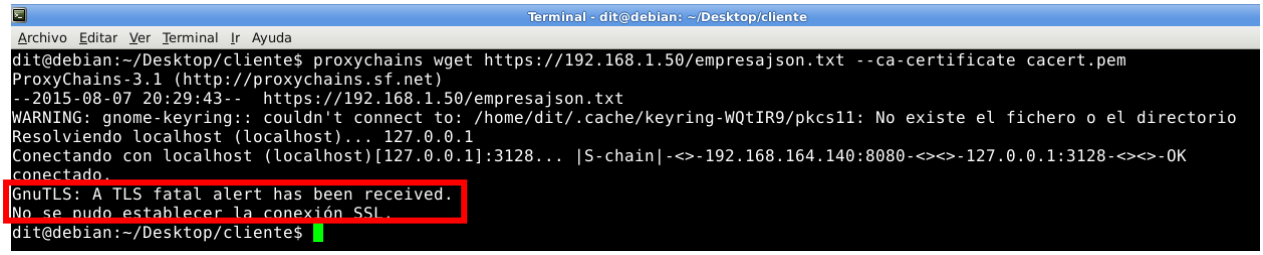

Figura 9-2. Ejecución wget ante ataque “droptls”

Podemos determinar que la conexión con el servidor está segura ante este tipo de ataques, ya que los clientes están configurados para no permitir el acceso por SSLv3. Hay que tener cuidado, porque esto depende del cliente, por lo que es necesario que si se conectara otro cliente diferente éste también rechazara esta versión.

9.1.2 MITM con certificado auto-firmado

Como ya se ha comentado, si se realiza este ataque al cliente java con el certificado que genera automáticamente *nogotofail* (para la opción *selfsigned)* encontramos un problema de compatibilidad en los formatos de lectura y escritura de los certificados entre ambas aplicaciones. Se obtiene este error en el cliente:

```
  Javax.net.ssl.SSLProtocolException: java.io.IOException: Parse Generalized time,
invalid offset
```

La solución a este problema ya se ha expuesto en el capítulo anterior, que es sustituyendo el certificado autofirmado por *nogotofail* por otro que generemos nosotros directamente con OpenSSL. El resultado de la ejecución es que si se llama al cliente de forma normal da un error al no reconocer la autenticación del servidor.

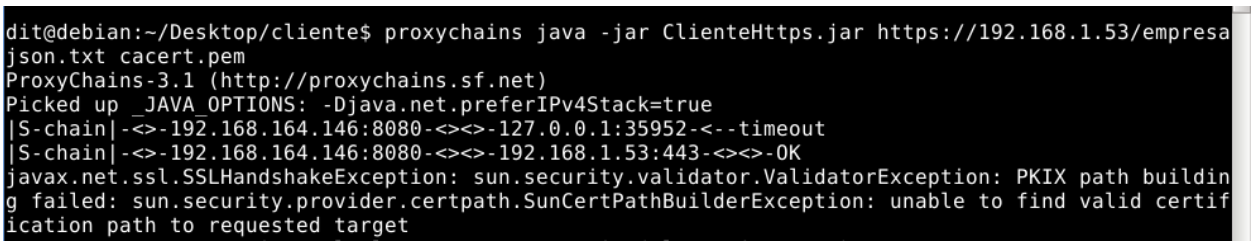

Figura 9-3. Ejecución normal cliente java ante ataque “selfsigned”

Si se ejecuta con la opción --nocompruebaCert va a dejar pasar cualquier certificado. Como vemos en la captura, el servidor atacante ha sido capaz de realizar el ataque MITM y obtener todos los datos de la conexión, recibiendo el cliente los datos del servidor sin darse cuenta de que ha sido víctima de un ataque. El ataque quedará reflejado en *nogotofail*, que mostrará un error crítico en la conexión (ver Anexo D).

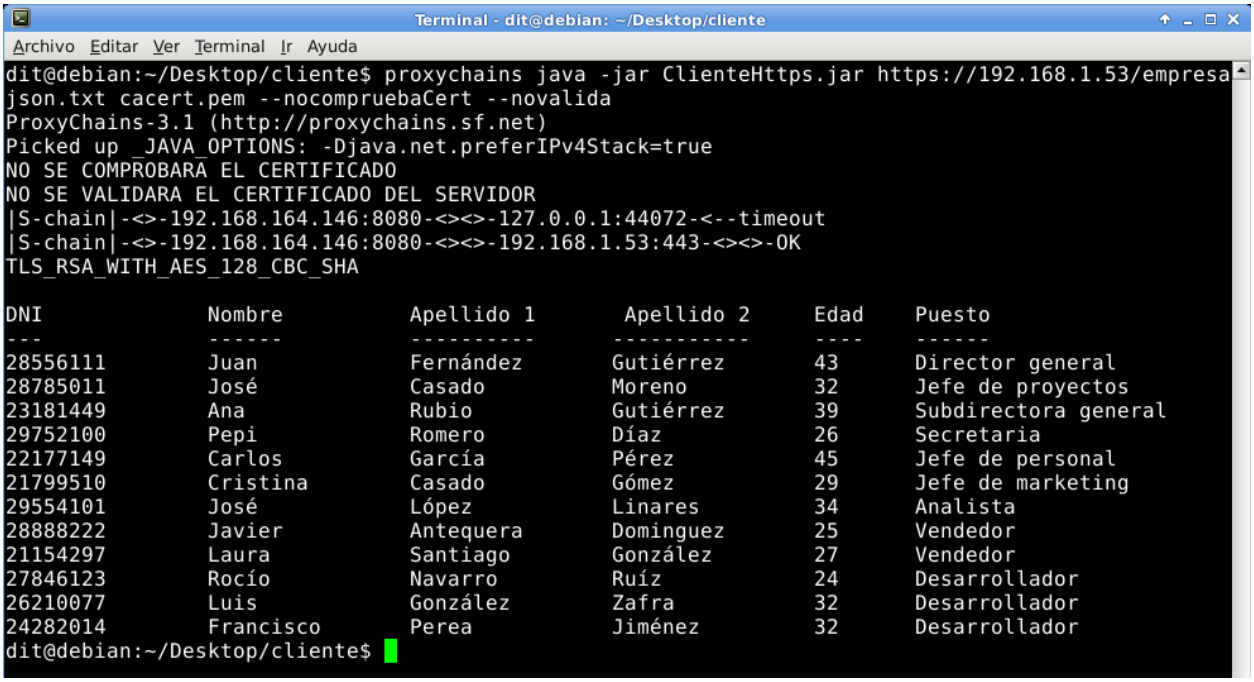

Figura 9-4. Ejecución cliente java con --nocompruebaCert ante ataque "selfsigned"

Se ha pasado la herramienta también haciendo una petición con wget. Este no da problemas con el certificado propio generado por *nogotofail*, pudiendo ver con el certificado de la herramienta tanto su ejecución correcta (que da un error al no poder verificar el certificado) y su ejecución forzada para que no compruebe el certificado, viendo cómo se realiza el ataque sin problemas. En esta última podemos observar que a pesar de que se fuerza para que no se compruebe, el cliente te avisa de que el certificado no es confiable.

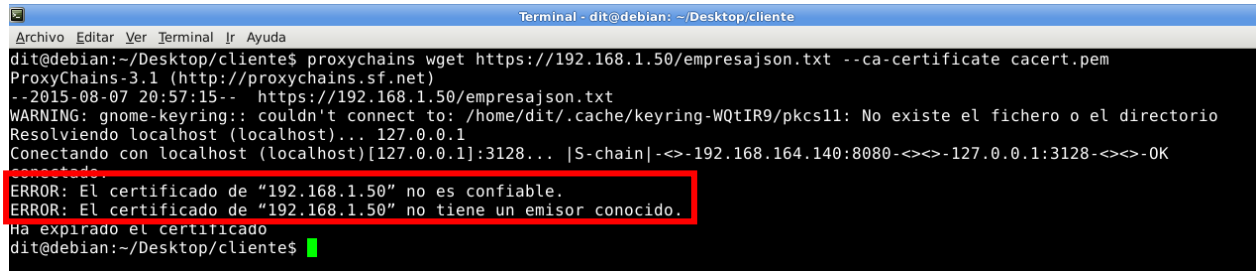

Figura 9-5. Ejecución wget ante ataque "selfsigned"

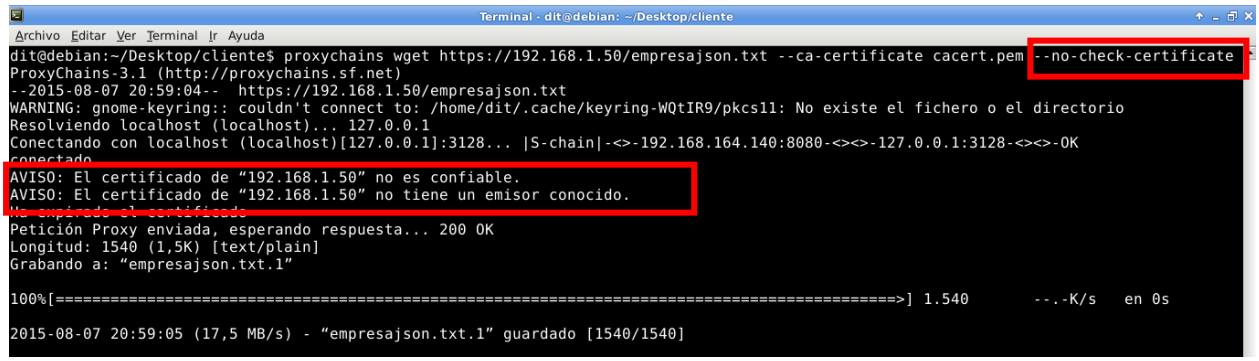

Figura 9-6. Ejecución wget con --no-check-certificate ante ataque "selfsigned"

9.1.3 Heartbleed para cliente

Cuando se realiza el ataque *heartbleed* para el cliente, llama la atención que en el creado en Java no hay problemas a la hora de ejecutarlo. Buscando en la documentación de Oracle, se indica que Java SE no es vulnerable a este ataque, ya que no incluye ni utiliza OpenSSL. Aun así, se sacaron en su momento una serie de parches para Java de forma que sus programas fueran inmunes a esta vulnerabilidad.

Para el caso de wget, es capaz de detenerlo. En esta ocasión, cuando se ejecuta el cliente para acceder al servidor, detecta un mensaje no válido, lo que hace que se detenga la conexión. Desde el punto de vista de la seguridad, esto es mejor debido a que no se establece ningún tipo de conexión, evitando cualquier otro problema (por ejemplo el ataque MITM, que no llega a ser detectado en Java con este ataque).

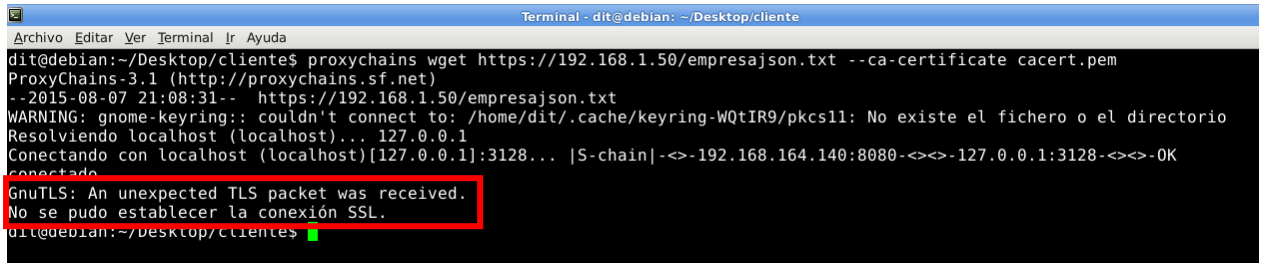

Figura 9-7. Ejecución wget ante ataque "clientheartbleed"

9.1.4 Reemplazo clave de servidor

El ataque *serverkeyreplace* es detectado por ambos clientes, siendo detenida la conexión con el servidor cuando se detecta una clave distinta. En el cliente java se indica que se ha recibido un parámetro ilegal (que va a ser la clave del servidor que no coincide).

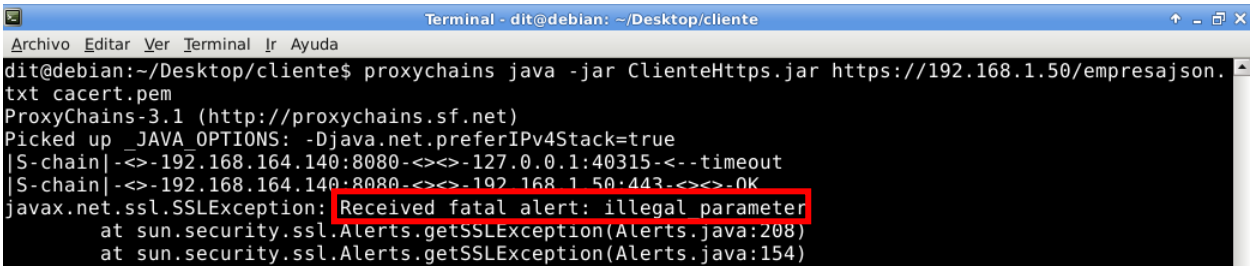

Figura 9-8. Ejecución cliente java ante ataque "serverkeyreplace"

Para wget también se obtiene un error, que dice que la verificación de la firma de la clave pública ha fallado, lo que impide que se establezca la conexión con el servidor.

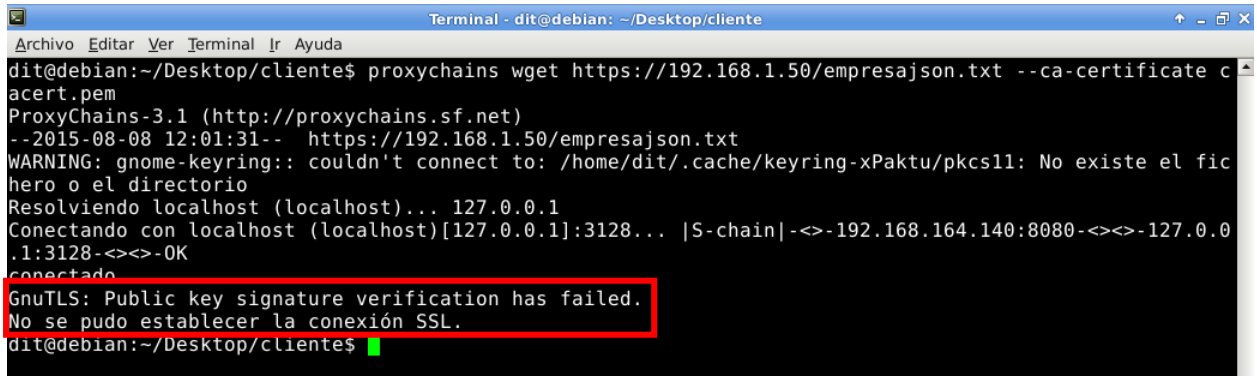

Figura 9-9. Ejecución wget ante ataque "serverkeyreplace"

9.1.5 MITM para servidor anónimo

Si se indica un servidor anónimo con *anonserver* en *nogotofail* también va a ser detectado por las ejecuciones normales de los dos clientes que están siendo probados.

Los mensajes obtenidos para los clientes coinciden con los que se muestran cuando se les pasa el ataque *droptls*, indicando una alerta fatal de la conexión (Figura 9-1 y Figura 9-2).

9.1.6 Exclusión de conexiones SSL

Lo mismo ocurre con el ataque *dropssl*. Los clientes bloquean la conexión con el servidor cuando se les ataca con esto, impidiendo que se conecten de forma vulnerable. De nuevo, los mensajes obtenidos por los dos clientes indican una alerta fatal como en *droptls* y en *anonserver*.

9.1.7 MITM con certificado de Superfish

Cuando se intenta un ataque MITM con la firma de la CA de Superfish (*superfishmitm*), nos encontramos en el cliente java con el mismo problema que con los certificados auto-firmados, ya que es un certificado que viene con la herramienta *nogotofail*, con el problema de compatibilidad ya mencionado. Si realizamos la misma solución que antes (sustituyendo en *nogotofail* el certificado por uno nuevo) se puede ver que la respuesta del cliente va a ser la misma que en los certificados auto-firmados, deteniendo la conexión en la ejecución normal (Figura 9-3) y dejando obtener los datos del servidor en la ejecución con la opción --nocompruebaCert (Figura 9-4). Por tanto, si esta CA no se encuentra entre nuestras autoridades de confianza el cliente no debería a dejar conectarse con el servidor.

Igualmente lo vemos con wget. Si se ejecuta de forma normal no se va a poder acceder al servidor porque no puede autenticarlo. Si se arranca wget con la opción --no-check-certificate, entonces sí se realizará el ataque MITM satisfactoriamente, dejando al cliente vulnerable.

Las capturas también van a coincidir con las de *selfsigned* para wget. En funcionamiento normal detiene la conexión porque el certificado no es confiable (Figura 9-5) y sin la comprobación es capaz de descargar el fichero JSON (Figura 9-6).

9.1.8 Dominio inválido

El ataque *invalidhostname* también va a ser paralizado por ambos clientes. Al detectar que el dominio que indica el certificado no coincide con el que se quiere conectar, saltan las excepciones correspondientes.

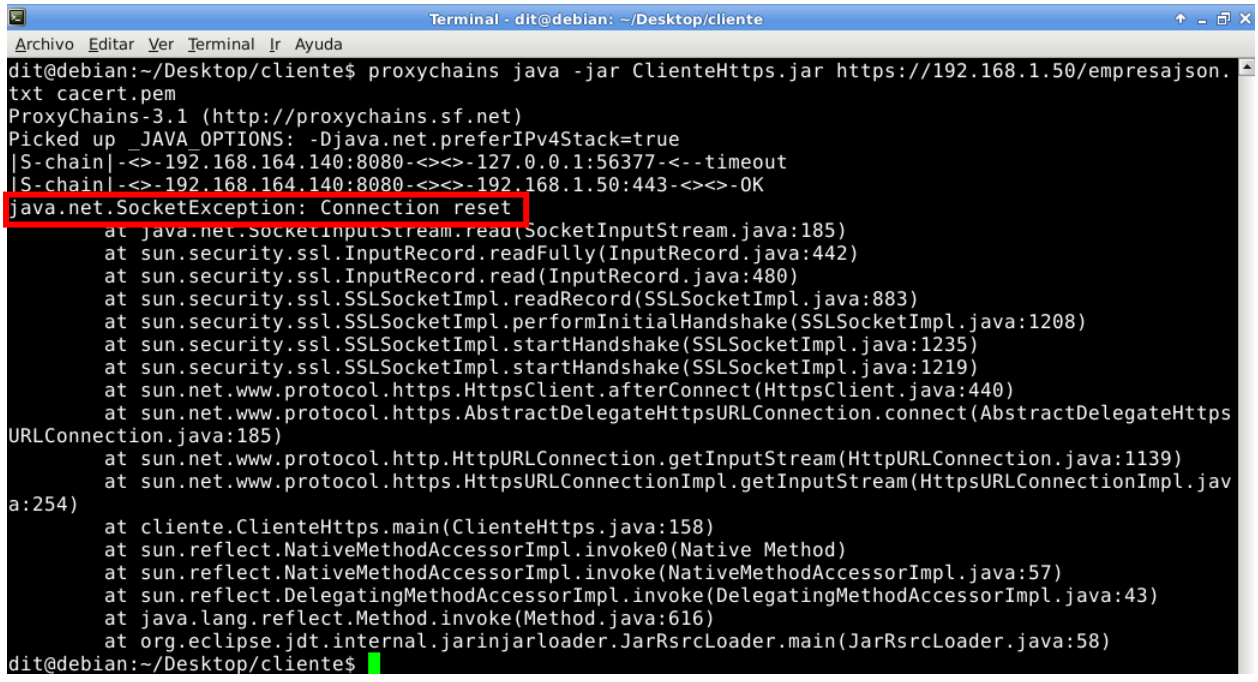

Figura 9-10. Ejecución cliente java ante ataque “invalidhostname”

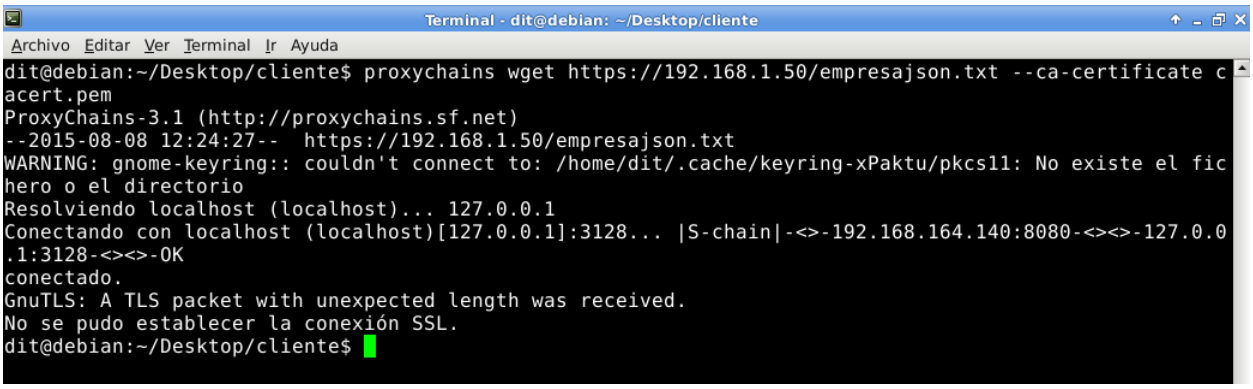

Figura 9-11. Ejecución wget ante ataque “invalidhostname”

9.1.9 Prueba early CCS de OpenSSL

Al realizar *earlyccs* de la utilidad *nogotofail* se está comprobando si es vulnerable a este agujero de seguridad de OpenSSL.

En el cliente java salta una excepción referida a la pérdida del algoritmo de cifrado (recordamos que esta vulnerabilidad estaba en los mensajes de cambio a la especificación del cifrado).

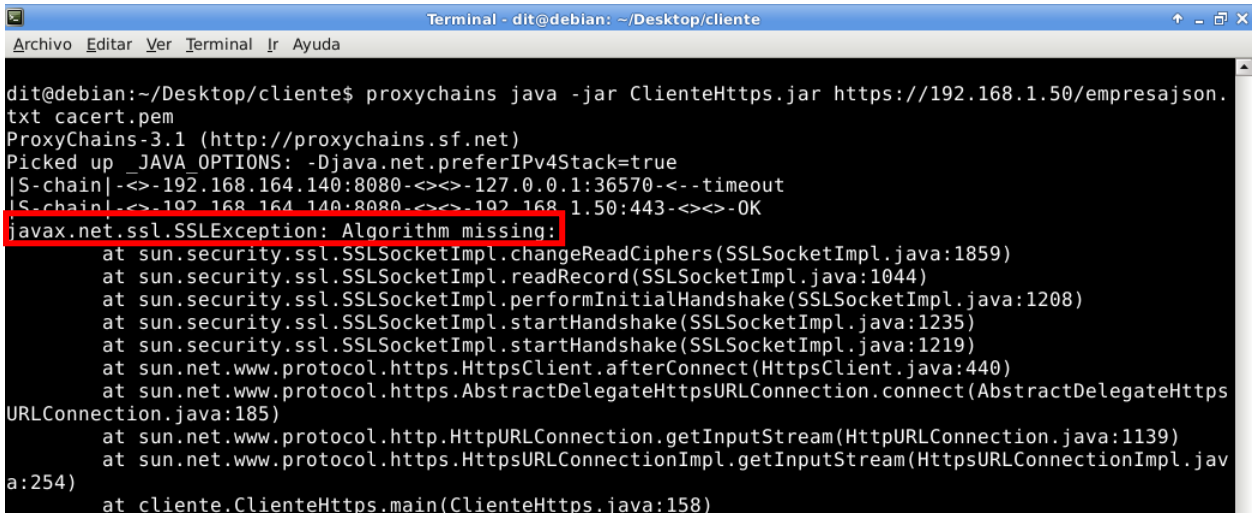

Figura 9-12. Ejecución cliente java ante ataque “earlyccs”

Para wget también se paraliza la conexión con el servidor, mostrando el error de que el paquete TLS no es el esperado.

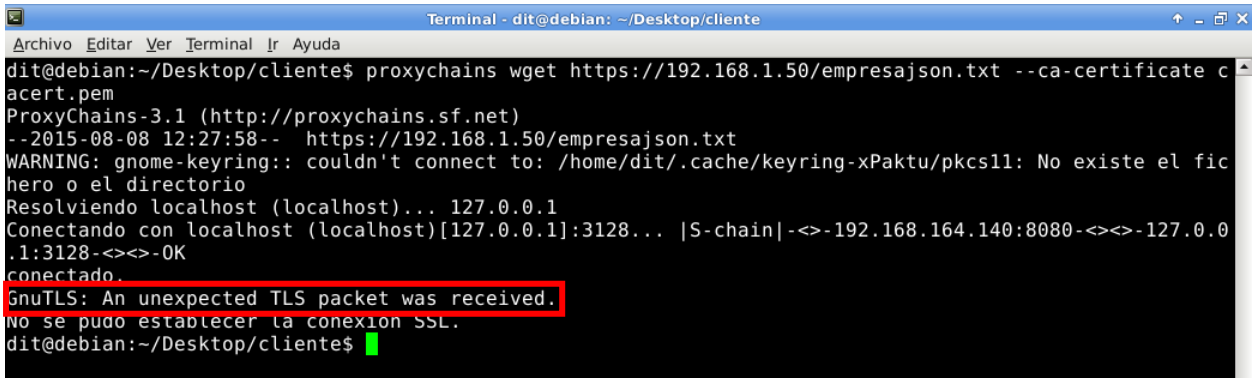

Figura 9-13. Ejecución wget ante ataque “earlyccs”

9.2 Análisis del tráfico

Como ya se ha comentado, *nogotofail* realiza un análisis de los datos de la conexión en busca de tráfico vulnerable. Conforme se han ido realizando las pruebas se han detectado algunos casos vulnerables que han ido siendo resueltos. Aun así, y como forma de exponerlo para esta memoria, se han incluido como opciones en los argumentos del cliente java estas vulnerabilidades de manera forzada, para poder ver la diferencia y cómo *nogotofail* es capaz de detectarlo.

El primero de ellos es la detección de HTTP, que se trata de información en texto plano sin encriptar. Si forzamos el cliente con la opción *--http*, va a emplear este protocolo. Se puede comprobar que el servidor MITM va a detectar este tráfico, determinándolo como un error en la conexión.

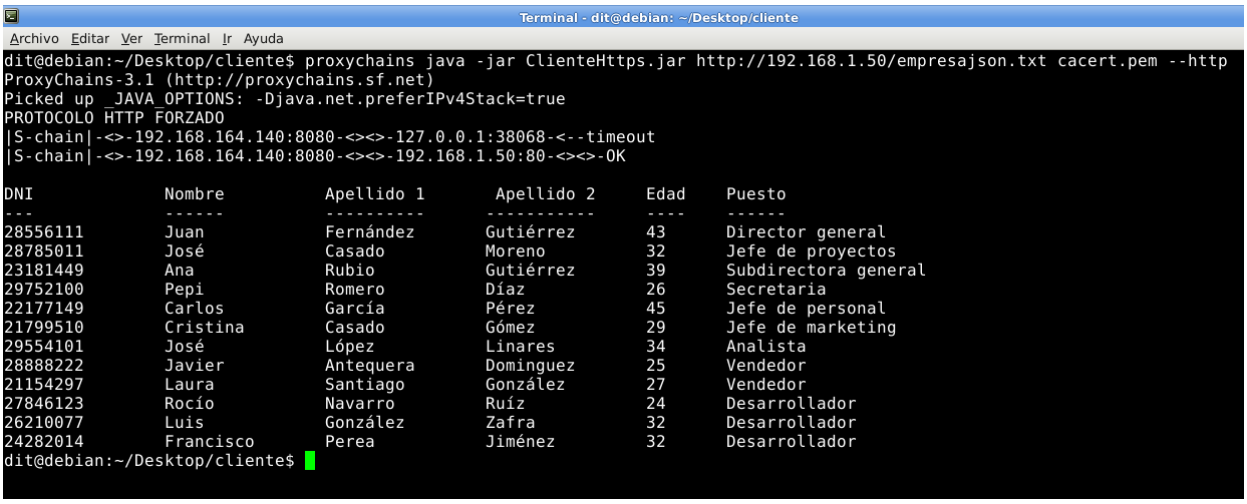

Figura 9-14. Ejecución cliente java con forzado HTTP

Con una ejecución normal también puede obtenerse el error de la detección de http. Esto se debe porque la validación que realiza el cliente sobre el estado del certificado mediante OCSP se hace mediante texto plano. Esto es así ya que el servidor OCSP se basa en respuestas HTTP GET. Sin embargo, esto no llega a ser un problema, ya que se está verificando una información de un certificado que es público, por lo que no existe riesgo en que sea leído por una tercera persona.

Otro de los asuntos a destacar es que java por defecto utiliza el protocolo SSL para la conexión con un servidor. Esto también es reflejado por *nogotofail*, y podemos verlo usando la opción *--noTLS* del cliente. En un funcionamiento normal se fuerza en el propio código de la aplicación a que se emplee TLS.

Además el servidor MITM también va a detectar un cifrado inseguro, que es el que usa Java por defecto. También ha sido resuelto, pero puede verse indicando la opción *--noTLS*, que engloba ambas vulnerabilidades.

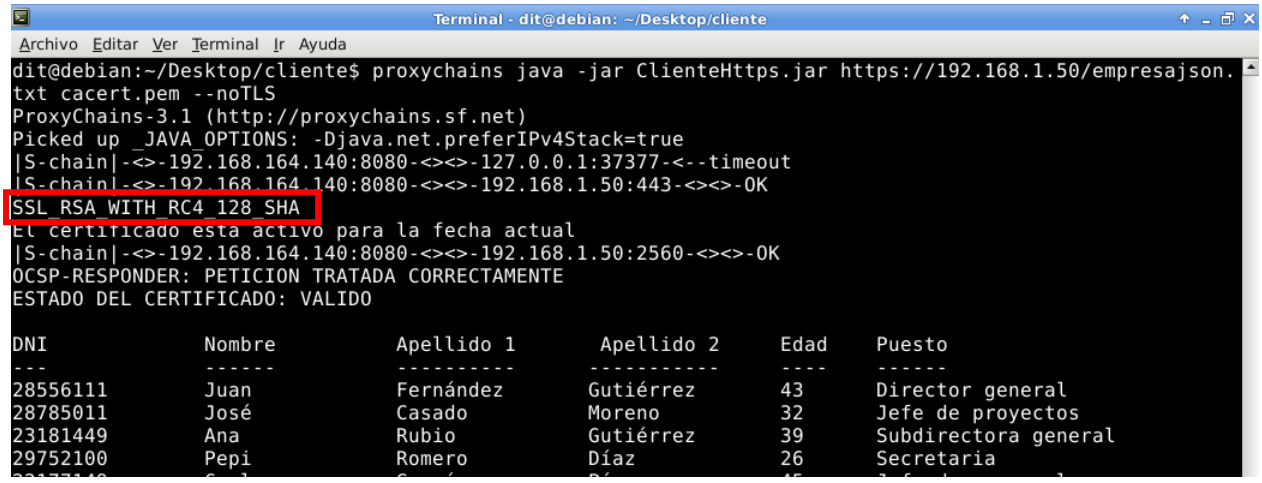

Figura 9-15. Ejecución cliente java con forzado no TLS

9.3 Caso de cliente vulnerable

A día de hoy, a pesar de que los clientes más utilizados poseen herramientas para evitar los ataques que han sido mostrados, todavía existen programas que tienen agujeros de seguridad ante estos. Vamos a estudiar en este apartado un caso para el ataque *heartbleed*.

Se ha buscado a través de la red y se ha dado con el cliente cURL. Esta herramienta tiene características similares a wget, mostrando el resultado por pantalla. A pesar de no dejar pasar otro tipo de ataques (por ejemplo MITM a partir de certificado autofirmado, POODLE, etc), en el caso de heartbleed se muestra vulnerable.

Cuando se le pasa la herramienta *nogotofail*, el cliente recibe el mensaje *heartbeat* modificado, enviando una respuesta, siendo recibida por el atacante y pudiendo contener en ella información sensible. Tras la ejecución con nogotofail, el cliente se encuentra con una respuesta vacía, pero lo importante es que en el servidor *nogotofail* se ha recibido la respuesta al ataque.

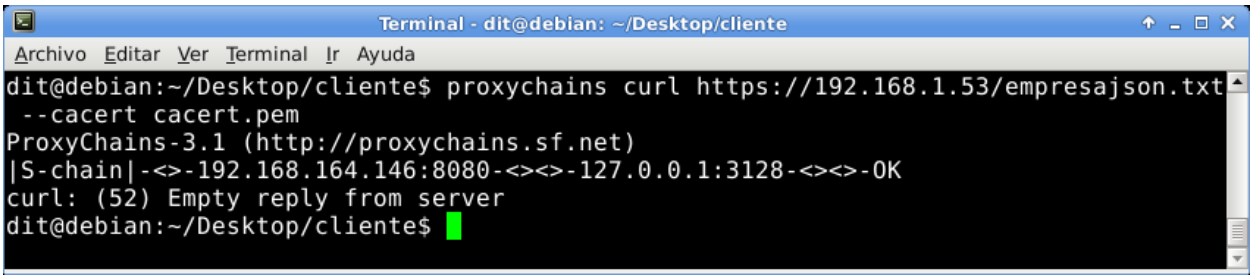

Figura 9-16. Ejecución curl ante ataque heartbleed

La herramienta *nogotofail* nos informa con un error crítico tras detectar que ha sido capaz de realizar el ataque con éxito.

```
  2015-09-04 13:56:44,771 [CRITICAL] [192.168.164.147:47423<=>127.0.0.1:3128 683b23ef-
ab8a-48fd-80d9-272c606e6aff clientheartbleed](Unknown) Heartbleed response received
```

Con este ejemplo se pretende demostrar que no sólo hay que asegurar el servidor. Ambos extremos de la conexión deben ser seguros, para evitar problemas como este. En la actualidad, son muchos los clientes que siguen siendo vulnerables a múltiples ataques, lo que nos indica que no siempre por tener una conexión HTTPS podemos decir que es segura.

10 CONCLUSIONES

Lo importante es no dejar de hacerse preguntas.

- Albert Einstein -

El proyecto llevado a cabo nos da una idea de cómo tiene que ser una arquitectura completa para tener un sistema seguro.

Con la creación de la autoridad certificadora se ha intentado cumplir los objetivos para que se pueda solicitar la firma de un certificado para un servidor de forma sencilla y automática. Se ha estudiado la teoría la criptografía y los sistemas libres existentes para ponerla en marcha y lograr un programa que se pueda emplear desde Linux para la solicitud.

Además, se ha buscado crear una conexión cliente-servidor que tape los agujeros de seguridad posibles gracias a la comprobación con el servidor OCSP configurado y a la herramienta *nogotofail* para detectar vulnerabilidades ante ataques. Recordar que el cliente se ha diseñado para realizar las diferentes pruebas con opciones vulnerables. Para un uso genérico habría que desactivar dichas opciones. Con esto podríamos desarrollar cualquier aplicación más compleja, incluso sentar la idea para sistemas Android, ya que éste basa su programación en Java.

Con los resultados obtenidos se ha visto cómo siempre hay cosas que se pueden escapar a la hora de diseñar un nuevo sistema, como ocurrió con la conexión que java lo realizaba usando por defecto SSL y no TLS, o con el cliente cURL vulnerable a ataques heartbleed. Esto ha sido de ayuda para comprobar la seguridad y aprender acerca de los ataques existentes y las vulnerabilidades a las que se enfrentan las aplicaciones hoy en día.

Aun así, y como siempre se dice, la seguridad total no existe, por lo que estos resultados sólo nos llevan a un análisis momentáneo, que puede verse afectado por cualquier agujero de seguridad que sea encontrado en un futuro no muy lejano.

10.1 Líneas futuras

Ya se ha comentado que este proyecto sólo es la base para la construcción de otros de mayor envergadura. Por el lado de la autoridad, sería necesario ponerla en un servidor real con acceso a todo el mundo. Además habría que proporcionar el certificado público, publicándolo por ejemplo en Internet para su descarga.

La autoridad debería poder atender varias peticiones al mismo tiempo. Eso se escapa a los requisitos de este proyecto en el que se buscaba la puesta en marcha de toda la infraestructura. Pero si se quisiera realizar esto habría que añadir hilos en la programación en C para que cada hilo sea capaz de atender una solicitud del servidor.

También se debería añadir la mejora de seguridad a la hora de revocar un certificado. La CA debería comprobar que el que está pidiendo la revocación sea el que contenga dicho certificado que se indica en la petición.

Otra mejora posible es hacer que se configuren los certificados para otros servidores, ya que actualmente sólo lo hace para Apache.

El hecho de que el servidor OCSP no funcione totalmente en tiempo real es una de las tareas pendientes, ya que toma los datos de las CRLs generadas por la CA. Se ha actualice casi al instante, pero esto va a ser así mientras haya CRLs pequeñas, cuando sean de mayor tamaño se incrementará el tiempo de actualización.

Respecto a la conexión cliente-servidor, recordar que habrá que ir comprobando nuevos posibles ataques a los que puedan ser vulnerables.

El cliente java se ha diseñado para las pruebas de este proyecto, pero una futura mejora sería la creación de una aplicación mayor que realice funciones específicas que requieran la conexión segura con el servidor.

ANEXO A: GUÍA DE INSTALACIÓN Y USO

La máquina donde se está ejecutando es Debian 7. Es posible que no funcione correctamente si se ejecuta en una máquina con sistema distinto. Se necesitan instalar una serie de librerías para que funcionen tanto autoridad como servidor. El gestor de instalaciones debe estar actualizado, por lo que se recomienda ejecutar: `apt-get update`*.*

Autoridad

a) Instalación y puesta en marcha

- Descomprimir fichero `certificacionTFG.zip` en el sistema. En este caso se ha realizado en `$HOME/Desktop`, que a partir de ahora denominaremos `$DIR`.
- Necesitaremos permisos de root para instalar la autoridad. Para ello entramos con:

```
su -
```

- Situarse en el directorio descomprimido anteriormente, y dentro de él en su directorio autoridad:

```
cd $DIR/certificacionTFG/autoridad
```

- Necesitamos dar permisos de ejecución a los scripts, ejecutando por ejemplo este comando (estando situados en el directorio anteriormente citado):

```
chmod  u+x autoridad.sh scripts/*
```

- Antes de instalar, se pueden modificar algunos parámetros de la autoridad, cuya configuración se encuentra en el fichero `configCA.conf`, tales como la ruta donde se instala la autoridad, los parámetros de su certificado, o el usuario y grupo desde el que se va a ejecutar.

- Se puede modificar también la dirección del servidor OCSP, desde el fichero situado en `scripts/OCSP/ficheros/config_openssl_ocsp.cnf`, que por defecto viene en localhost, pero que puede modificarse al dominio o ip de la máquina donde se esté instalando.

- Ejecutamos el siguiente script para instalar la autoridad, que automáticamente realizará también la compilación de los ficheros para ejecutarla:

```
./autoridad.sh --instala
```

- A lo largo del proceso de instalación, es necesario instalar una serie de librerías, que se descargarán en el caso de que no estén en el sistema. Las librerías a utilizar son:
    - Para CA: `openssl, libssl-dev`
    - Para OCSP: `build-essential, libldap-dev, libxml2-dev`

  Al ejecutar los comandos de instalación con `apt-get install`, es posible que se instalen algunas librerías adicionales que necesitan las arriba mencionadas.

  Además, se instalarán también el servidor OCSP (se ha empleado OpenCA's OCSP Responder - v3.1.2) y la librería LibPKI necesaria para que este servidor funcione. Ambos se encuentran comprimidos en el fichero `certificacionTFG.zip` (en el directorio `autoridad/scripts/ocspdfuente.zip`) y que se descomprimirán e instalarán automáticamente durante la ejecución del script de instalación.

- Durante su instalación del servidor OCSP, se realizarán algunas preguntas, donde en la primera se deberá introducir `'CN=OCSP Server, C=ES'`, y en las siguientes dejarlas tal cual, pulsando Enter. Debería de quedar algo tal que así:

```
Please Enter the Server's Subject (eg., CN=OCSP Server, O=OpenCA, C=US):
CN=OCSP Server, C=ES
Please Enter the Algorithm (default: RSA-SHA256):

Please Enter the Key Size (default: 2048):
```

- Si todo ha sido correcto, ya tendremos lista la autoridad para ponerla a funcionar, ejecutando simplemente el siguiente comando:

```
./autoridad.sh --ejecuta
```

  Este comando, aunque se invoca desde root, es un script que activará el servidor OCSP (necesita estos permisos para ser iniciado) y que iniciará la autoridad de certificación desde un usuario nuevo creado en la instalación (por defecto *tfgca*).

b) Desinstalación

- En el caso de que quisiéramos desinstalar la autoridad, con permisos de root habría que ejecutar el siguiente comando (situándonos en el mismo directorio):

```
./autoridad.sh --desinstala
```

- Si deseamos desinstalar la autoridad y el servidor OCSP instalado, habría que indicar esta opción (también como superusuario):

```
./autoridad.sh --desinstalaConOCSP
```

c) Script general

- El script que hemos utilizado tanto para instalar, desinstalar como para poner en marcha contiene un menú más intuitivo que se obtiene si ejecutamos sin parámetros:

```
./autoridad.sh
```

Servidor

a) Instalación

- Descomprimir fichero `certificacionTFG.zip` en el sistema si no lo hemos hecho ya. En este caso se ha realizado en `$HOME/Desktop`, que a partir de ahora denominaremos `$DIR`.
- Necesitaremos permisos de root para instalar el servidor por primera vez. Para ello entramos con:

```
su -
```

- Situarse en el directorio descomprimido anteriormente, y dentro de él en su directorio servidor:

```
cd $DIR/certificacionTFG/servidor
```

- Necesitamos dar permisos de ejecución a los scripts, ejecutando por ejemplo este comando (estando situados en el directorio anteriormente citado):

```
chmod  u+x servidor.sh scripts/*
```

- Ejecutamos el siguiente script para instalar los paquetes necesarios para que funcione el servidor, que también compilará los ficheros para ponerlo en marcha:

```
./servidor.sh --instala
```

- A lo largo del proceso de instalación, es necesario instalar una serie de librerías, que se descargarán en el caso de que no estén en el sistema. Las librerías a utilizar son: `apache2, openssl, libssl-dev`

  Al ejecutar los comandos de instalación con `apt-get install`, es posible que se instalen algunas librerías adicionales que necesitan las arriba mencionadas.

b) Ejecución

- Una vez compilado todo sin problema, procedemos a ejecutar el programa para solicitar un certificado a la autoridad. Antes que nada, hay que tener en cuenta que el script está configurado para una dirección IP de la autoridad que es la que ha sido usada para su programación y pruebas. Por tanto habrá que cambiarla por la dirección o dominio donde esté instalada la autoridad.

  Para ello, abrimos el fichero de configuración `configServidor.conf` y cambiamos la siguiente línea, poniendo la dirección que sea:

```
NOMBRECA=192.168.1.37
```

  Además de esto, podemos cambiar en este fichero otros parámetros respecto al servidor, tales como los parámetros de la petición, la ruta del servidor, etc.

- Procedemos a solicitar un certificado a la autoridad (colocando lo que queremos certificar donde pone IP o dominio) y posterior configuración en el servidor Apache:

```
./servidor.sh --certifica <IP o DOMINIO>
```

- Para borrar el certificado (y revocarlo también), ejecutamos lo siguiente:

```
./servidor.sh --borra
```

- Para revocar el certificado, sin llegar a borrarlo del servidor, ejecutamos:

```
./servidor.sh --revoca
```

Los dos últimos comandos indicados revocarán el certificado que se encuentra configurado en Apache.

- Para ejecutar posteriormente el cliente, se debe situar el fichero empresajson.txt (que se encuentra en `$DIR/certificacionTFG/servidor/fich`) en el directorio del servidor `/var/www/`. Este fichero será el que el cliente lea en el programa que hemos realizado para probar el sistema.

c) Script general

- El script que hemos utilizado para pedir, revocar o borrar un certificado contiene un menú más intuitivo que se obtiene si ejecutamos sin parámetros:

```
./servidor.sh
```

Cliente

a) Ejecución

- Descomprimir fichero `certificacionTFG.zip` en el sistema si no lo hemos hecho ya. Situarse en el directorio descomprimido, y dentro de él en su directorio cliente:

```
cd $DIR/certificacionTFG/servidor
```

- Antes de ejecutar el cliente, es necesario tener descargado el certificado público de la autoridad (*`cacert.pem`*).

- Se ha creado un script para ejecutar el cliente de manera mucho más sencilla, que accede al certificado de la autoridad (que tiene que estar situado en el directorio del cliente) y a la dirección del servidor (*https://localhost/empresajson.txt*). La forma de ejecutar esto es:

```
./cliente.sh
./cliente.sh --novalida (en el caso de no querer validar el certificado)
```

También se puede añadir como argumento la dirección a la que nos conectamos:

```
./cliente.sh https://192.168.1.53/empresajson.txt
```

- Necesitamos dar permisos de ejecución al script en caso de usarlo, ejecutando por ejemplo este comando (estando situados en el directorio anteriormente citado):

```
chmod  u+x cliente.sh
```

- Este script se ha creado sólo para ejecutar las pruebas de forma más sencilla. Si se desea ejecutar el cliente sin utilizar el script, y con más opciones, realizar lo siguiente:
    - Descomprimir fichero *ClienteHttps.jar.tar.gz*:

```
tar xzvf ClienteHttps.jar.tar.gz
```

- Ejecutar programa cliente empaquetado en el jar descomprimido, siendo <URL_SERVIDOR> la dirección web donde se accede al fichero json del servidor, y <CACERT> la ruta hasta el certificado de la autoridad:

```
java -jar ClienteHttps.jar <URL_SERVIDOR> <CACERT>
```

- Si no se quiere validar el certificado del servidor:

```
java -jar ClienteHttps.jar <URL_SERVIDOR> <CACERT> --novalida
```

- Si no se desea comprobar el certificado:

```
java -jar ClienteHttps.jar <URL_SERVIDOR> <CACERT> --nocompruebaCert
```

- Si se desea ejecutar en una página utilizando solo HTTP (sin seguridad):

```
java -jar ClienteHttps.jar <(http://) URL_SERVIDOR> <CACERT> --http
```

- Si no se quiere utilizar el protocolo TLS:

```
java -jar ClienteHttps.jar <URL_SERVIDOR> <CACERT> --noTLS
```

Todas estas opciones son acumulables y se pueden indicar una tras otra (excepto la de http, que con ella se debe indicar la url con http:// y no soporta las opciones de seguridad de los protocolos TLS o SSL).

ANEXO B: INSTALACIÓN DE NOGOTOFAIL

Antes de detallar los pasos para la instalación, mencionar que lo que se indica a continuación se ha incluido en un script (`nogotofail/configura_nogotofail.sh`) para facilitar la tarea de instalación para las pruebas a realizar en este proyecto. Este script sólo es funcional para este uso, sin garantizar su funcionamiento para otro tipo de configuraciones.

Para la instalación de *nogotofail*, ha sido necesario la descarga de la herramienta de su sitio web: *https://github.com/google/nogotofail*. Como está desarrollado en github, se ha empleado *git* para su clonación en un directorio de la máquina. Se ha utilizado para ponerlo en marcha un sistema Ubuntu, que funcionará como el equipo que se situará "en el medio".

```
git clone https://github.com/google/nogotofail.git
```

Una vez descargado en el directorio correspondiente, hay que descargar las librerías necesarias para que funcione (si no estaban descargadas ya en el equipo).

```
apt-get install python python-openssl python-psutil python-setuptools
```

Además, será necesario generar con *openssl* un certificado para encriptar la conexión. Esto es indicado en la guía *Getting Started* que se proporciona junto al código. El comando utilizado para generar dicho certificado es:

```
openssl req -x509 -newkey rsa:2048 -sha256 -subj "/CN=mitm.nogotofail/" -nodes -keyout server.crt -out server.crt
```

También se dice en la guía que para el ataque de hostname inválido es necesario tener un certificado de una autoridad de confianza pero con un nombre diferente al del servidor al que nos vayamos a conectar. Este certificado se debe llamar `trusted-cert.pem` y debe estar situado en el directorio donde se ha descargado *nogotofail*.

Para ello, se ha generado en nuestra autoridad un certificado con estas características con hostname *evilprueba.com* y se ha comprobado que es lo que queremos con el siguiente comando, obteniendo OK en la salida:

```
openssl verify -CApath /etc/ssl/certs/ -untrusted trusted-cert.pem trusted-cert.pem
```

Para facilitar esta tarea, se ha creado en la autoridad un script (`autoridad/scripts/pruebas/trustCertNogotofail.sh`) que genera este certificado, teniendo que copiarlo a la máquina donde se está instalando *nogotofail*. En ella, será necesario copiar el

fichero al directorio *nogotofail*, así como configurar la clave pública de la autoridad a la que pertenece. Para ello, tenemos un nuevo script (`nogotofail/configCA.sh`) que ejecutado con permisos especiales guarda esta clave en `/etc/ssl/certs`. Esta clave tiene que estar almacenada en el mismo directorio que donde se ejecuta el script y con nombre cacert.pem para que funcione el script correctamente.

Una vez configurado, el comando básico para poner el servidor nogotofail en marcha es:

```
python -m nogotofail.mitm --mode socks --port 8080 --serverssl server.crt
```

Este comando generará algunos ataques o se mantiene sin atacar de manera aleatoria. Posteriormente se verán otras ejecuciones para ver los ataques concretos. *Nogotofail* contiene también un cliente opcional para el caso de querer ver la información en otro equipo diferente al del servidor instalado. En este caso no es necesario, por tanto no se va a utilizar.

Este servidor se va a comportar como un proxy, por tanto debemos indicar al cliente que debe pasar por él. Todo esto es una simulación y por eso lo hacemos así, en un caso real el atacante se situaría en el medio siendo el cliente ejecutado de forma normal.

En el cliente se ha instalado la herramienta *proxychains*:

```
apt-get install proxychains
```

Antes de ejecutarlo, hay que configurar *proxychains* para que acceda a donde tenemos instalado nogotofail, añadiendo la siguiente línea al fichero `/etc/proxychains.conf` en la sección `[Proxy List]` (siendo la IP la de la máquina donde se encuentre el servidor de *nogotofail*):

```
socks5 192.168.164.140 8080
```

Para ejecutar el cliente ahora se hace añadiendo *proxychains* a la ejecución normal:

```
proxychains java -jar ClienteHttps.jar https://192.168.1.50/empresajson.txt
cacert.pem
```

ANEXO C: CÓDIGO

El código diseñado para este proyecto no se ha incluido en esta memoria debido a su extensión. Va adjunto en el disco a entregar junto con la memoria. Está comprimido en el directorio `certificacionTFG`, el cual contiene los siguientes directorios (cada uno para cada entidad):

- **autoridad**: Todo el código fuente referido a la CA y el servidor OCSP (ya que ambos se han instalado en la misma máquina). En el directorio `fuente/` está el código del programa en C y en el directorio `scripts/` se encuentran todos los scripts creados, a excepción del script `autoridad.sh` que se encuentra en el principal.
- **servidor:** Incluye código para el programa del servidor web que solicitará el certificado y lo configurará en su máquina. Al igual que en la autoridad, están los directorios `fuente/` y `scripts/`, y el script `servidor.sh` que accederá a todas las funciones que se encuentra en el directorio principal del servidor.
- **cliente:** Código java para la aplicación del cliente creada. Se encontrará el fichero comprimido `ClienteHttps.jar.tar.gz` (que habrá que descomprimir para obtener el jar) y el script `cliente.sh`.
- **nogotofail:** Fichero comprimido *nogotofail* con el código fuente descargado y los scripts `configura_nogotofail.sh` y `configCA.sh` que facilitan la configuración para este proyecto en concreto.

A pesar de que todo vaya comprimido en un solo directorio, cada una de las carpetas corresponde a entidades que se pueden instalar en máquinas diferentes, teniendo en cuenta las configuraciones indicadas en el anexo A.

ANEXO D: REGISTRO DE PRUEBAS NOGOTOFAIL

Log de los ataques y las detecciones del tráfico de datos desde el servidor de *nogotofail* para cada una de las pruebas realizadas. Se muestran en el mismo orden en que fueron expuestas en los capítulos 8 y 9.

droptls

- **cliente java**

```
2015-08-07 20:25:21,003 [INFO] Starting...
2015-08-07 20:25:28,366 [INFO] [192.168.164.139:38170<=>127.0.0.1:33485 11d5fc20-3654-
4c10-bdc8-261bfa0178ea logging](Unknown) Selected for connection
2015-08-07 20:25:28,368 [INFO] Failed to connect to endpoint 127.0.0.1:33485 errno 111
2015-08-07 20:25:28,369 [INFO] [192.168.164.139:38170<=>127.0.0.1:33485 11d5fc20-3654-
4c10-bdc8-261bfa0178ea logging](Unknown) Connection closed
2015-08-07 20:25:28,447 [INFO] [192.168.164.139:38171<=>192.168.1.50:443 afa8d53b-
16db-4507-95d2-d95d5a18829d logging](Unknown) Selected for connection
2015-08-07 20:25:28,449 [INFO] [192.168.164.139:38171<=>192.168.1.50:443 afa8d53b-
16db-4507-95d2-d95d5a18829d logging](Unknown) Connection established
2015-08-07 20:25:28,456 [INFO] [192.168.164.139:38171<=>192.168.1.50:443 afa8d53b-
16db-4507-95d2-d95d5a18829d logging](Unknown) Handler being removed
2015-08-07 20:25:28,456 [INFO] [192.168.164.139:38171<=>192.168.1.50:443 afa8d53b-
16db-4507-95d2-d95d5a18829d droptls](Unknown) Selected for connection
2015-08-07 20:25:28,458 [INFO] [192.168.164.139:38171<=>192.168.1.50:443 afa8d53b-
16db-4507-95d2-d95d5a18829d droptls](Unknown) Connection closed by handler
```

- **wget**

```
2015-08-07 20:29:44,081 [INFO] [192.168.164.139:38183<=>127.0.0.1:3128 551e9602-984a-
479f-98b4-950d435a52c8 logging](Unknown) Selected for connection
2015-08-07 20:29:44,083 [INFO] [192.168.164.139:38183<=>127.0.0.1:3128 551e9602-984a-
479f-98b4-950d435a52c8 logging](Unknown) Connection established
2015-08-07 20:29:44,086 [ERROR] [192.168.164.139:38183<=>127.0.0.1:3128 551e9602-984a-
479f-98b4-950d435a52c8 httpdetection](Unknown) HTTP request CONNECT
127.0.0.1192.168.1.50:443
2015-08-07 20:29:44,481 [INFO] [192.168.164.139:38183<=>127.0.0.1:3128 551e9602-984a-
479f-98b4-950d435a52c8 logging](Unknown) Handler being removed
2015-08-07 20:29:44,482 [INFO] [192.168.164.139:38183<=>127.0.0.1:3128 551e9602-984a-
479f-98b4-950d435a52c8 droptls](Unknown) Selected for connection
2015-08-07 20:29:44,483 [INFO] [192.168.164.139:38183<=>127.0.0.1:3128 551e9602-984a-
479f-98b4-950d435a52c8 droptls](Unknown) Connection closed by handler
```

selfsigned

- **cliente java (con comprobación de certificado)**

```
2015-09-05 13:03:54,627 [INFO] Starting...
2015-09-05 13:04:39,368 [INFO] [192.168.164.147:55422<=>127.0.0.1:39398 9960bcef-dc61-
40d6-a1ef-096b4c20d8e8 logging](Unknown) Selected for connection
2015-09-05 13:04:39,369 [INFO] Failed to connect to endpoint 127.0.0.1:39398 errno 111
2015-09-05 13:04:39,369 [INFO] [192.168.164.147:55422<=>127.0.0.1:39398 9960bcef-dc61-
40d6-a1ef-096b4c20d8e8 logging](Unknown) Connection closed
2015-09-05 13:04:39,500 [INFO] [192.168.164.147:55423<=>192.168.1.53:443 dd4c8197-
e1de-4f62-9178-d01337c9f04f logging](Unknown) Selected for connection
2015-09-05 13:04:39,503 [INFO] [192.168.164.147:55423<=>192.168.1.53:443 dd4c8197-
e1de-4f62-9178-d01337c9f04f logging](Unknown) Connection established
2015-09-05 13:04:39,543 [INFO] [192.168.164.147:55423<=>192.168.1.53:443 dd4c8197-
e1de-4f62-9178-d01337c9f04f logging](Unknown) Handler being removed
2015-09-05 13:04:39,543 [INFO] [192.168.164.147:55423<=>192.168.1.53:443 dd4c8197-
e1de-4f62-9178-d01337c9f04f selfsigned](Unknown) Selected for connection
2015-09-05 13:04:40,151 [INFO] [192.168.164.147:55423<=>192.168.1.53:443 dd4c8197-
e1de-4f62-9178-d01337c9f04f selfsigned](Unknown) Connection closed
```

- **cliente java (sin comprobación de certificado)**

```
2015-09-05 13:04:59,363 [INFO] Starting...
2015-09-05 13:05:21,223 [INFO] [192.168.164.147:55424<=>127.0.0.1:53510 cde08758-2665-
4a2e-bedb-85ebd49f32a3 logging](Unknown) Selected for connection
2015-09-05 13:05:21,224 [INFO] Failed to connect to endpoint 127.0.0.1:53510 errno 111
2015-09-05 13:05:21,224 [INFO] [192.168.164.147:55424<=>127.0.0.1:53510 cde08758-2665-
4a2e-bedb-85ebd49f32a3 logging](Unknown) Connection closed
2015-09-05 13:05:21,307 [INFO] [192.168.164.147:55425<=>192.168.1.53:443 727b347c-
a9a2-4442-bf6c-5c59400de58f logging](Unknown) Selected for connection
2015-09-05 13:05:21,309 [INFO] [192.168.164.147:55425<=>192.168.1.53:443 727b347c-
a9a2-4442-bf6c-5c59400de58f logging](Unknown) Connection established
2015-09-05 13:05:21,317 [INFO] [192.168.164.147:55425<=>192.168.1.53:443 727b347c-
a9a2-4442-bf6c-5c59400de58f logging](Unknown) Handler being removed
2015-09-05 13:05:21,323 [INFO] [192.168.164.147:55425<=>192.168.1.53:443 727b347c-
a9a2-4442-bf6c-5c59400de58f selfsigned](Unknown) Selected for connection
2015-09-05 13:05:21,645 [INFO] [192.168.164.147:55425<=>192.168.1.53:443 727b347c-
a9a2-4442-bf6c-5c59400de58f selfsigned](Unknown) SSL connection established
2015-09-05 13:05:21,650 [CRITICAL] [192.168.164.147:55425<=>192.168.1.53:443 727b347c-
a9a2-4442-bf6c-5c59400de58f selfsigned](Unknown) MITM Success! Cert file:
/tmp/._cert_ca.pem_-413061719.pem
2015-09-05 13:05:21,652 [ERROR] [192.168.164.147:55425<=>192.168.1.53:443 727b347c-
a9a2-4442-bf6c-5c59400de58f httpdetection](Unknown) HTTP request GET
192.168.1.53/empresajson.txt
2015-09-05 13:05:21,797 [INFO] [192.168.164.147:55425<=>192.168.1.53:443 727b347c-
a9a2-4442-bf6c-5c59400de58f selfsigned](Unknown) Connection closed
```

- **wget (con comprobación de certificado)**

```
2015-08-07 20:57:16,199 [INFO] [192.168.164.139:38189<=>127.0.0.1:3128 74071fda-b265-
4ef2-b35b-b1bb60705522 logging](Unknown) Selected for connection
2015-08-07 20:57:16,200 [INFO] [192.168.164.139:38189<=>127.0.0.1:3128 74071fda-b265-
4ef2-b35b-b1bb60705522 logging](Unknown) Connection established
2015-08-07 20:57:16,228 [ERROR] [192.168.164.139:38189<=>127.0.0.1:3128 74071fda-b265-
4ef2-b35b-b1bb60705522 httpdetection](Unknown) HTTP request CONNECT
127.0.0.1192.168.1.50:443
2015-08-07 20:57:16,263 [INFO] [192.168.164.139:38189<=>127.0.0.1:3128 74071fda-b265-
4ef2-b35b-b1bb60705522 logging](Unknown) Handler being removed
2015-08-07 20:57:16,264 [INFO] [192.168.164.139:38189<=>127.0.0.1:3128 74071fda-b265-
4ef2-b35b-b1bb60705522 selfsigned](Unknown) Selected for connection
2015-08-07 20:57:16,532 [INFO] [192.168.164.139:38189<=>127.0.0.1:3128 74071fda-b265-
4ef2-b35b-b1bb60705522 selfsigned](Unknown) SSL connection established
2015-08-07 20:57:16,839 [INFO] [192.168.164.139:38189<=>127.0.0.1:3128 74071fda-b265-
4ef2-b35b-b1bb60705522 selfsigned](Unknown) Connection closed
```

- **wget (sin comprobación de certificado)**

```
2015-08-07 20:59:05,359 [INFO] [192.168.164.139:38190<=>127.0.0.1:3128 b6a74ff2-5e2c-
4506-b482-119b64829945 logging](Unknown) Selected for connection
2015-08-07 20:59:05,361 [INFO] [192.168.164.139:38190<=>127.0.0.1:3128 b6a74ff2-5e2c-
4506-b482-119b64829945 logging](Unknown) Connection established
2015-08-07 20:59:05,362 [ERROR] [192.168.164.139:38190<=>127.0.0.1:3128 b6a74ff2-5e2c-
4506-b482-119b64829945 httpdetection](Unknown) HTTP request CONNECT
127.0.0.1192.168.1.50:443
2015-08-07 20:59:05,367 [INFO] [192.168.164.139:38190<=>127.0.0.1:3128 b6a74ff2-5e2c-
4506-b482-119b64829945 logging](Unknown) Handler being removed
2015-08-07 20:59:05,367 [INFO] [192.168.164.139:38190<=>127.0.0.1:3128 b6a74ff2-5e2c-
4506-b482-119b64829945 selfsigned](Unknown) Selected for connection
2015-08-07 20:59:05,538 [INFO] [192.168.164.139:38190<=>127.0.0.1:3128 b6a74ff2-5e2c-
4506-b482-119b64829945 selfsigned](Unknown) SSL connection established
2015-08-07 20:59:05,852 [CRITICAL] [192.168.164.139:38190<=>127.0.0.1:3128 b6a74ff2-
5e2c-4506-b482-119b64829945 selfsigned](Unknown) MITM Success! Cert file:
/tmp/._cert_ca.pem_242500049.pem
2015-08-07 20:59:05,854 [ERROR] [192.168.164.139:38190<=>127.0.0.1:3128 b6a74ff2-5e2c-
4506-b482-119b64829945 httpdetection](Unknown) HTTP request GET
192.168.1.50/empresajson.txt
2015-08-07 20:59:05,859 [INFO] [192.168.164.139:38190<=>127.0.0.1:3128 b6a74ff2-5e2c-
4506-b482-119b64829945 selfsigned](Unknown) Connection closed
```

clienthearbleed

- **cliente java**

```
2015-08-07 21:07:37,246 [INFO] [192.168.164.139:38199<=>127.0.0.1:60640 23d0f27e-9ab8-
4874-93c4-7605adced4f2 logging](Unknown) Selected for connection
2015-08-07 21:07:37,247 [INFO] Failed to connect to endpoint 127.0.0.1:60640 errno 111
2015-08-07 21:07:37,247 [INFO] [192.168.164.139:38199<=>127.0.0.1:60640 23d0f27e-9ab8-
4874-93c4-7605adced4f2 logging](Unknown) Connection closed
2015-08-07 21:07:37,333 [INFO] [192.168.164.139:38200<=>192.168.1.50:443 6617f79b-
230c-4819-8ddc-3255647118c1 logging](Unknown) Selected for connection
2015-08-07 21:07:37,336 [INFO] [192.168.164.139:38200<=>192.168.1.50:443 6617f79b-
230c-4819-8ddc-3255647118c1 logging](Unknown) Connection established
2015-08-07 21:07:37,343 [INFO] [192.168.164.139:38200<=>192.168.1.50:443 6617f79b-
230c-4819-8ddc-3255647118c1 logging](Unknown) Handler being removed
2015-08-07 21:07:37,343 [INFO] [192.168.164.139:38200<=>192.168.1.50:443 6617f79b-
230c-4819-8ddc-3255647118c1 clientheartbleed](Unknown) Selected for connection
2015-08-07 21:07:38,025 [INFO] [192.168.164.139:38201<=>192.168.1.50:2560 bc0733dd-
f54a-45ab-828b-3fd2410b3433 logging](Unknown) Selected for connection
2015-08-07 21:07:38,027 [INFO] [192.168.164.139:38201<=>192.168.1.50:2560 bc0733dd-
f54a-45ab-828b-3fd2410b3433 logging](Unknown) Connection established
2015-08-07 21:07:38,034 [ERROR] [192.168.164.139:38201<=>192.168.1.50:2560 bc0733dd-
f54a-45ab-828b-3fd2410b3433 httpdetection](Unknown) HTTP request POST
192.168.1.50:2560/
2015-08-07 21:07:38,050 [INFO] [192.168.164.139:38201<=>192.168.1.50:2560 bc0733dd-
f54a-45ab-828b-3fd2410b3433 logging](Unknown) Connection closed
```

- **wget**

```
2015-08-07 21:09:29,680 [INFO] [192.168.164.139:38204<=>127.0.0.1:3128 23e54e2b-e9fe-
4b6f-b1b9-e8b40e544fb5 logging](Unknown) Selected for connection
2015-08-07 21:09:29,681 [INFO] [192.168.164.139:38204<=>127.0.0.1:3128 23e54e2b-e9fe-
4b6f-b1b9-e8b40e544fb5 logging](Unknown) Connection established
2015-08-07 21:09:29,682 [ERROR] [192.168.164.139:38204<=>127.0.0.1:3128 23e54e2b-e9fe-
4b6f-b1b9-e8b40e544fb5 httpdetection](Unknown) HTTP request CONNECT
127.0.0.1192.168.1.50:443
2015-08-07 21:09:29,686 [INFO] [192.168.164.139:38204<=>127.0.0.1:3128 23e54e2b-e9fe-
4b6f-b1b9-e8b40e544fb5 logging](Unknown) Handler being removed
2015-08-07 21:09:29,687 [INFO] [192.168.164.139:38204<=>127.0.0.1:3128 23e54e2b-e9fe-
4b6f-b1b9-e8b40e544fb5 clientheartbleed](Unknown) Selected for connection
```

```
2015-08-07 21:09:29,785 [INFO] [192.168.164.139:38204<=>127.0.0.1:3128 23e54e2b-e9fe-
4b6f-b1b9-e8b40e544fb5 clientheartbleed](Unknown) Connection closed
```

serverkeyreplace

- **cliente java**

```
2015-08-08 11:58:04,204 [INFO] Starting...
2015-08-08 11:58:56,063 [INFO] [192.168.164.139:42531<=>127.0.0.1:40315 d198c403-1dfc-
473f-863b-738134cd323b logging](Unknown) Selected for connection
2015-08-08 11:58:56,064 [INFO] Failed to connect to endpoint 127.0.0.1:40315 errno 111
2015-08-08 11:58:56,066 [INFO] [192.168.164.139:42531<=>127.0.0.1:40315 d198c403-1dfc-
473f-863b-738134cd323b logging](Unknown) Connection closed
2015-08-08 11:58:56,418 [INFO] [192.168.164.139:42532<=>192.168.1.50:443 f9c19af8-
5553-425e-b57d-5ae2ee01aa57 logging](Unknown) Selected for connection
2015-08-08 11:58:56,420 [INFO] [192.168.164.139:42532<=>192.168.1.50:443 f9c19af8-
5553-425e-b57d-5ae2ee01aa57 logging](Unknown) Connection established
2015-08-08 11:58:56,439 [INFO] [192.168.164.139:42532<=>192.168.1.50:443 f9c19af8-
5553-425e-b57d-5ae2ee01aa57 logging](Unknown) Handler being removed
2015-08-08 11:58:56,440 [INFO] [192.168.164.139:42532<=>192.168.1.50:443 f9c19af8-
5553-425e-b57d-5ae2ee01aa57 serverkeyreplace](Unknown) Selected for connection
2015-08-08 11:58:56,592 [INFO] [192.168.164.139:42532<=>192.168.1.50:443 f9c19af8-
5553-425e-b57d-5ae2ee01aa57 serverkeyreplace](Unknown) Connection closed
```

- **wget**

```
2015-08-08 12:01:37,519 [INFO] [192.168.164.139:42537<=>127.0.0.1:3128 74f033ab-8a6e-
4a95-9103-290073cb2f69 logging](Unknown) Selected for connection
2015-08-08 12:01:37,521 [INFO] [192.168.164.139:42537<=>127.0.0.1:3128 74f033ab-8a6e-
4a95-9103-290073cb2f69 logging](Unknown) Connection established
2015-08-08 12:01:37,523 [ERROR] [192.168.164.139:42537<=>127.0.0.1:3128 74f033ab-8a6e-
4a95-9103-290073cb2f69 httpdetection](Unknown) HTTP request CONNECT
127.0.0.1192.168.1.50:443
2015-08-08 12:01:37,600 [INFO] [192.168.164.139:42537<=>127.0.0.1:3128 74f033ab-8a6e-
4a95-9103-290073cb2f69 logging](Unknown) Handler being removed
2015-08-08 12:01:37,601 [INFO] [192.168.164.139:42537<=>127.0.0.1:3128 74f033ab-8a6e-
4a95-9103-290073cb2f69 serverkeyreplace](Unknown) Selected for connection
2015-08-08 12:01:37,773 [INFO] [192.168.164.139:42537<=>127.0.0.1:3128 74f033ab-8a6e-
4a95-9103-290073cb2f69 serverkeyreplace](Unknown) Connection closed
```

anonserver

- **cliente java**

```
2015-08-08 12:10:15,307 [INFO] Starting...
2015-08-08 12:10:39,133 [INFO] [192.168.164.139:42538<=>127.0.0.1:43132 a14e53ea-5c8d-
4614-8295-dd0a915f65e4 logging](Unknown) Selected for connection
2015-08-08 12:10:39,134 [INFO] Failed to connect to endpoint 127.0.0.1:43132 errno 111
2015-08-08 12:10:39,135 [INFO] [192.168.164.139:42538<=>127.0.0.1:43132 a14e53ea-5c8d-
4614-8295-dd0a915f65e4 logging](Unknown) Connection closed
2015-08-08 12:10:39,208 [INFO] [192.168.164.139:42539<=>192.168.1.50:443 cc8861b1-
c0a7-4b6a-ae8f-aabdd557377d logging](Unknown) Selected for connection
2015-08-08 12:10:39,210 [INFO] [192.168.164.139:42539<=>192.168.1.50:443 cc8861b1-
c0a7-4b6a-ae8f-aabdd557377d logging](Unknown) Connection established
2015-08-08 12:10:39,219 [INFO] [192.168.164.139:42539<=>192.168.1.50:443 cc8861b1-
c0a7-4b6a-ae8f-aabdd557377d logging](Unknown) Handler being removed
2015-08-08 12:10:39,220 [INFO] [192.168.164.139:42539<=>192.168.1.50:443 cc8861b1-
c0a7-4b6a-ae8f-aabdd557377d anonserver](Unknown) Selected for connection
2015-08-08 12:10:39,457 [INFO] [192.168.164.139:42539<=>192.168.1.50:443 cc8861b1-
c0a7-4b6a-ae8f-aabdd557377d anonserver](Unknown) Connection closed
```

- **wget**

```
2015-08-08 12:12:10,996 [INFO] [192.168.164.139:42544<=>127.0.0.1:3128 19f5a617-5e4e-
4c7a-afb2-2e07fa0e0e1d logging](Unknown) Selected for connection
2015-08-08 12:12:10,997 [INFO] [192.168.164.139:42544<=>127.0.0.1:3128 19f5a617-5e4e-
4c7a-afb2-2e07fa0e0e1d logging](Unknown) Connection established
2015-08-08 12:12:11,007 [ERROR] [192.168.164.139:42544<=>127.0.0.1:3128 19f5a617-5e4e-
4c7a-afb2-2e07fa0e0e1d httpdetection](Unknown) HTTP request CONNECT
127.0.0.1192.168.1.50:443
2015-08-08 12:12:11,018 [INFO] [192.168.164.139:42544<=>127.0.0.1:3128 19f5a617-5e4e-
4c7a-afb2-2e07fa0e0e1d logging](Unknown) Handler being removed
2015-08-08 12:12:11,018 [INFO] [192.168.164.139:42544<=>127.0.0.1:3128 19f5a617-5e4e-
4c7a-afb2-2e07fa0e0e1d anonserver](Unknown) Selected for connection
2015-08-08 12:12:11,133 [INFO] [192.168.164.139:42544<=>127.0.0.1:3128 19f5a617-5e4e-
4c7a-afb2-2e07fa0e0e1d anonserver](Unknown) Connection closed
```

dropssl

- **cliente java**

```
2015-08-08 12:13:08,617 [INFO] Starting...
2015-08-08 12:13:15,573 [INFO] [192.168.164.139:42545<=>127.0.0.1:50647 ae873145-2db3-
4f57-bca0-12fc9b9220bd logging](Unknown) Selected for connection
2015-08-08 12:13:15,573 [INFO] Failed to connect to endpoint 127.0.0.1:50647 errno 111
2015-08-08 12:13:15,574 [INFO] [192.168.164.139:42545<=>127.0.0.1:50647 ae873145-2db3-
4f57-bca0-12fc9b9220bd logging](Unknown) Connection closed
2015-08-08 12:13:15,641 [INFO] [192.168.164.139:42546<=>192.168.1.50:443 19ea3e5a-
16be-4281-91ce-9cd8348d67df logging](Unknown) Selected for connection
2015-08-08 12:13:15,644 [INFO] [192.168.164.139:42546<=>192.168.1.50:443 19ea3e5a-
16be-4281-91ce-9cd8348d67df logging](Unknown) Connection established
2015-08-08 12:13:15,656 [INFO] [192.168.164.139:42546<=>192.168.1.50:443 19ea3e5a-
16be-4281-91ce-9cd8348d67df logging](Unknown) Handler being removed
2015-08-08 12:13:15,657 [INFO] [192.168.164.139:42546<=>192.168.1.50:443 19ea3e5a-
16be-4281-91ce-9cd8348d67df dropssl](Unknown) Selected for connection
2015-08-08 12:13:15,658 [INFO] [192.168.164.139:42546<=>192.168.1.50:443 19ea3e5a-
16be-4281-91ce-9cd8348d67df dropssl](Unknown) Connection closed by handler
```

- **wget**

```
2015-08-08 12:14:05,719 [INFO] [192.168.164.139:42547<=>127.0.0.1:3128 0d92b667-9718-
4410-9b4b-09e8a6865516 logging](Unknown) Selected for connection
2015-08-08 12:14:05,720 [INFO] [192.168.164.139:42547<=>127.0.0.1:3128 0d92b667-9718-
4410-9b4b-09e8a6865516 logging](Unknown) Connection established
2015-08-08 12:14:05,722 [ERROR] [192.168.164.139:42547<=>127.0.0.1:3128 0d92b667-9718-
4410-9b4b-09e8a6865516 httpdetection](Unknown) HTTP request CONNECT
127.0.0.1192.168.1.50:443
2015-08-08 12:14:05,727 [INFO] [192.168.164.139:42547<=>127.0.0.1:3128 0d92b667-9718-
4410-9b4b-09e8a6865516 logging](Unknown) Handler being removed
2015-08-08 12:14:05,727 [INFO] [192.168.164.139:42547<=>127.0.0.1:3128 0d92b667-9718-
4410-9b4b-09e8a6865516 dropssl](Unknown) Selected for connection
2015-08-08 12:14:05,728 [INFO] [192.168.164.139:42547<=>127.0.0.1:3128 0d92b667-9718-
4410-9b4b-09e8a6865516 dropssl](Unknown) Connection closed by handler
```

superfishmitm

- **cliente java (con comprobación de certificado)**

```
2015-09-05 13:05:58,010 [INFO] Starting...
2015-09-05 13:06:05,955 [INFO] [192.168.164.147:55426<=>127.0.0.1:40338 0eeae400-a2d2-
4135-b738-162087741110 logging](Unknown) Selected for connection
```

```
2015-09-05 13:06:05,983 [INFO] Failed to connect to endpoint 127.0.0.1:40338 errno 111
2015-09-05 13:06:05,983 [INFO] [192.168.164.147:55426<=>127.0.0.1:40338 0eeae400-a2d2-
4135-b738-162087741110 logging](Unknown) Connection closed
2015-09-05 13:06:06,046 [INFO] [192.168.164.147:55427<=>192.168.1.53:443 ca1bda51-
84dd-452a-b7bd-b961a2c34079 logging](Unknown) Selected for connection
2015-09-05 13:06:06,048 [INFO] [192.168.164.147:55427<=>192.168.1.53:443 ca1bda51-
84dd-452a-b7bd-b961a2c34079 logging](Unknown) Connection established
2015-09-05 13:06:06,057 [INFO] [192.168.164.147:55427<=>192.168.1.53:443 ca1bda51-
84dd-452a-b7bd-b961a2c34079 logging](Unknown) Handler being removed
2015-09-05 13:06:06,062 [INFO] [192.168.164.147:55427<=>192.168.1.53:443 ca1bda51-
84dd-452a-b7bd-b961a2c34079 superfishmitm](Unknown) Selected for connection
2015-09-05 13:06:06,229 [INFO] [192.168.164.147:55427<=>192.168.1.53:443 ca1bda51-
84dd-452a-b7bd-b961a2c34079 superfishmitm](Unknown) Connection closed
```

- **cliente java (sin comprobación de certificado)**

```
2015-09-05 13:06:16,448 [INFO] Starting...
2015-09-05 13:06:24,722 [INFO] [192.168.164.147:55428<=>127.0.0.1:40754 286eab12-41e3-
4f60-a2f3-97dc2498c71d logging](Unknown) Selected for connection
2015-09-05 13:06:24,723 [INFO] Failed to connect to endpoint 127.0.0.1:40754 errno 111
2015-09-05 13:06:24,723 [INFO] [192.168.164.147:55428<=>127.0.0.1:40754 286eab12-41e3-
4f60-a2f3-97dc2498c71d logging](Unknown) Connection closed
2015-09-05 13:06:24,788 [INFO] [192.168.164.147:55429<=>192.168.1.53:443 43aab392-
ae6e-49f0-b540-afdb4259b4ed logging](Unknown) Selected for connection
2015-09-05 13:06:24,791 [INFO] [192.168.164.147:55429<=>192.168.1.53:443 43aab392-
ae6e-49f0-b540-afdb4259b4ed logging](Unknown) Connection established
2015-09-05 13:06:24,802 [INFO] [192.168.164.147:55429<=>192.168.1.53:443 43aab392-
ae6e-49f0-b540-afdb4259b4ed logging](Unknown) Handler being removed
2015-09-05 13:06:24,802 [INFO] [192.168.164.147:55429<=>192.168.1.53:443 43aab392-
ae6e-49f0-b540-afdb4259b4ed superfishmitm](Unknown) Selected for connection
2015-09-05 13:06:24,989 [INFO] [192.168.164.147:55429<=>192.168.1.53:443 43aab392-
ae6e-49f0-b540-afdb4259b4ed superfishmitm](Unknown) SSL connection established
2015-09-05 13:06:24,994 [CRITICAL] [192.168.164.147:55429<=>192.168.1.53:443 43aab392-
ae6e-49f0-b540-afdb4259b4ed superfishmitm](Unknown) MITM Success! Cert file:
/tmp/._cert_superfish.pem_-413061719.pem
2015-09-05 13:06:24,996 [ERROR] [192.168.164.147:55429<=>192.168.1.53:443 43aab392-
ae6e-49f0-b540-afdb4259b4ed httpdetection](Unknown) HTTP request GET
192.168.1.53/empresajson.txt
2015-09-05 13:06:25,135 [INFO] [192.168.164.147:55429<=>192.168.1.53:443 43aab392-
ae6e-49f0-b540-afdb4259b4ed superfishmitm](Unknown) Connection closed
```

- **wget (con comprobación de certificado)**

```
2015-08-08 12:17:45,639 [INFO] [192.168.164.139:42550<=>127.0.0.1:3128 e0bf858f-d429-
4713-a6bd-a0ee941e8046 logging](Unknown) Selected for connection
2015-08-08 12:17:45,640 [INFO] [192.168.164.139:42550<=>127.0.0.1:3128 e0bf858f-d429-
4713-a6bd-a0ee941e8046 logging](Unknown) Connection established
2015-08-08 12:17:45,644 [ERROR] [192.168.164.139:42550<=>127.0.0.1:3128 e0bf858f-d429-
4713-a6bd-a0ee941e8046 httpdetection](Unknown) HTTP request CONNECT
127.0.0.1192.168.1.50:443
2015-08-08 12:17:45,649 [INFO] [192.168.164.139:42550<=>127.0.0.1:3128 e0bf858f-d429-
4713-a6bd-a0ee941e8046 logging](Unknown) Handler being removed
2015-08-08 12:17:45,650 [INFO] [192.168.164.139:42550<=>127.0.0.1:3128 e0bf858f-d429-
4713-a6bd-a0ee941e8046 superfishmitm](Unknown) Selected for connection
2015-08-08 12:17:45,831 [INFO] [192.168.164.139:42550<=>127.0.0.1:3128 e0bf858f-d429-
4713-a6bd-a0ee941e8046 superfishmitm](Unknown) SSL connection established
2015-08-08 12:17:46,141 [INFO] [192.168.164.139:42550<=>127.0.0.1:3128 e0bf858f-d429-
4713-a6bd-a0ee941e8046 superfishmitm](Unknown) Connection closed
```

- **wget (sin comprobación de certificado)**

```
2015-08-08 12:18:57,580 [INFO] [192.168.164.139:42551<=>127.0.0.1:3128 238d7f19-38b0-
4c4a-87c0-d0a2d0fa6576 logging](Unknown) Selected for connection
2015-08-08 12:18:57,581 [INFO] [192.168.164.139:42551<=>127.0.0.1:3128 238d7f19-38b0-
4c4a-87c0-d0a2d0fa6576 logging](Unknown) Connection established
```

```
2015-08-08 12:18:57,583 [ERROR] [192.168.164.139:42551<=>127.0.0.1:3128 238d7f19-38b0-
4c4a-87c0-d0a2d0fa6576 httpdetection](Unknown) HTTP request CONNECT
127.0.0.1192.168.1.50:443
2015-08-08 12:18:57,587 [INFO] [192.168.164.139:42551<=>127.0.0.1:3128 238d7f19-38b0-
4c4a-87c0-d0a2d0fa6576 logging](Unknown) Handler being removed
2015-08-08 12:18:57,588 [INFO] [192.168.164.139:42551<=>127.0.0.1:3128 238d7f19-38b0-
4c4a-87c0-d0a2d0fa6576 superfishmitm](Unknown) Selected for connection
2015-08-08 12:18:57,738 [INFO] [192.168.164.139:42551<=>127.0.0.1:3128 238d7f19-38b0-
4c4a-87c0-d0a2d0fa6576 superfishmitm](Unknown) SSL connection established
2015-08-08 12:18:58,072 [CRITICAL] [192.168.164.139:42551<=>127.0.0.1:3128 238d7f19-
38b0-4c4a-87c0-d0a2d0fa6576 superfishmitm](Unknown) MITM Success! Cert file:
/tmp/._cert_superfish.pem_242500049.pem
2015-08-08 12:18:58,073 [ERROR] [192.168.164.139:42551<=>127.0.0.1:3128 238d7f19-38b0-
4c4a-87c0-d0a2d0fa6576 httpdetection](Unknown) HTTP request GET
192.168.1.50/empresajson.txt
2015-08-08 12:18:58,164 [INFO] [192.168.164.139:42551<=>127.0.0.1:3128 238d7f19-38b0-
4c4a-87c0-d0a2d0fa6576 superfishmitm](Unknown) Connection closed
```

invalidhostname

- **cliente java**

```
2015-08-08 12:20:31,449 [INFO] Starting...
2015-08-08 12:22:48,144 [INFO] [192.168.164.139:42552<=>127.0.0.1:56377 e86557e9-9a66-
4c00-937c-d659db47a041 logging](Unknown) Selected for connection
2015-08-08 12:22:48,145 [INFO] Failed to connect to endpoint 127.0.0.1:56377 errno 111
2015-08-08 12:22:48,145 [INFO] [192.168.164.139:42552<=>127.0.0.1:56377 e86557e9-9a66-
4c00-937c-d659db47a041 logging](Unknown) Connection closed
2015-08-08 12:22:48,199 [INFO] [192.168.164.139:42553<=>192.168.1.50:443 bd9e0fa6-
b38d-4e63-a1fd-8d42d34db91f logging](Unknown) Selected for connection
2015-08-08 12:22:48,201 [INFO] [192.168.164.139:42553<=>192.168.1.50:443 bd9e0fa6-
b38d-4e63-a1fd-8d42d34db91f logging](Unknown) Connection established
2015-08-08 12:22:48,237 [INFO] [192.168.164.139:42553<=>192.168.1.50:443 bd9e0fa6-
b38d-4e63-a1fd-8d42d34db91f logging](Unknown) Handler being removed
2015-08-08 12:22:48,237 [INFO] [192.168.164.139:42553<=>192.168.1.50:443 bd9e0fa6-
b38d-4e63-a1fd-8d42d34db91f invalidhostname](Unknown) Selected for connection
2015-08-08 12:22:48,348 [INFO] [192.168.164.139:42553<=>192.168.1.50:443 bd9e0fa6-
b38d-4e63-a1fd-8d42d34db91f invalidhostname](Unknown) Connection closed
```

- **wget**

```
2015-08-08 12:24:32,564 [INFO] [192.168.164.139:42556<=>127.0.0.1:3128 454386ed-fec7-
417f-bf6f-ff587599f094 logging](Unknown) Selected for connection
2015-08-08 12:24:32,565 [INFO] [192.168.164.139:42556<=>127.0.0.1:3128 454386ed-fec7-
417f-bf6f-ff587599f094 logging](Unknown) Connection established
2015-08-08 12:24:32,568 [ERROR] [192.168.164.139:42556<=>127.0.0.1:3128 454386ed-fec7-
417f-bf6f-ff587599f094 httpdetection](Unknown) HTTP request CONNECT
127.0.0.1192.168.1.50:443
2015-08-08 12:24:32,572 [INFO] [192.168.164.139:42556<=>127.0.0.1:3128 454386ed-fec7-
417f-bf6f-ff587599f094 logging](Unknown) Handler being removed
2015-08-08 12:24:32,573 [INFO] [192.168.164.139:42556<=>127.0.0.1:3128 454386ed-fec7-
417f-bf6f-ff587599f094 invalidhostname](Unknown) Selected for connection
2015-08-08 12:24:32,678 [INFO] [192.168.164.139:42556<=>127.0.0.1:3128 454386ed-fec7-
417f-bf6f-ff587599f094 invalidhostname](Unknown) Connection closed
```

earlyccs

- **cliente java**

```
2015-08-08 12:25:43,480 [INFO] Starting...
2015-08-08 12:25:53,845 [INFO] [192.168.164.139:42557<=>127.0.0.1:36570 7c2cfb8f-6fd7-
411b-b79d-036096ef72fa logging](Unknown) Selected for connection
```

```
2015-08-08 12:25:53,845 [INFO] Failed to connect to endpoint 127.0.0.1:36570 errno 111
2015-08-08 12:25:53,846 [INFO] [192.168.164.139:42557<=>127.0.0.1:36570 7c2cfb8f-6fd7-
411b-b79d-036096ef72fa logging](Unknown) Connection closed
2015-08-08 12:25:53,898 [INFO] [192.168.164.139:42558<=>192.168.1.50:443 6234cb82-
4a43-4f5d-9070-0b8a9d0cb56e logging](Unknown) Selected for connection
2015-08-08 12:25:53,900 [INFO] [192.168.164.139:42558<=>192.168.1.50:443 6234cb82-
4a43-4f5d-9070-0b8a9d0cb56e logging](Unknown) Connection established
2015-08-08 12:25:53,914 [INFO] [192.168.164.139:42558<=>192.168.1.50:443 6234cb82-
4a43-4f5d-9070-0b8a9d0cb56e logging](Unknown) Handler being removed
2015-08-08 12:25:53,914 [INFO] [192.168.164.139:42558<=>192.168.1.50:443 6234cb82-
4a43-4f5d-9070-0b8a9d0cb56e earlyccs](Unknown) Selected for connection
2015-08-08 12:25:53,922 [INFO] [192.168.164.139:42558<=>192.168.1.50:443 6234cb82-
4a43-4f5d-9070-0b8a9d0cb56e earlyccs](Unknown) Connection closed
```

- **wget**

```
2015-08-08 12:28:03,894 [INFO] [192.168.164.139:42560<=>127.0.0.1:3128 72717044-c852-
4b5e-9c52-35d9443ce410 logging](Unknown) Selected for connection
2015-08-08 12:28:03,895 [INFO] [192.168.164.139:42560<=>127.0.0.1:3128 72717044-c852-
4b5e-9c52-35d9443ce410 logging](Unknown) Connection established
2015-08-08 12:28:03,896 [ERROR] [192.168.164.139:42560<=>127.0.0.1:3128 72717044-c852-
4b5e-9c52-35d9443ce410 httpdetection](Unknown) HTTP request CONNECT
127.0.0.1192.168.1.50:443
2015-08-08 12:28:03,900 [INFO] [192.168.164.139:42560<=>127.0.0.1:3128 72717044-c852-
4b5e-9c52-35d9443ce410 logging](Unknown) Handler being removed
2015-08-08 12:28:03,901 [INFO] [192.168.164.139:42560<=>127.0.0.1:3128 72717044-c852-
4b5e-9c52-35d9443ce410 earlyccs](Unknown) Selected for connection
2015-08-08 12:28:03,992 [INFO] [192.168.164.139:42560<=>127.0.0.1:3128 72717044-c852-
4b5e-9c52-35d9443ce410 earlyccs](Unknown) Connection closed
```

HTTP (texto plano): Opción --http en cliente java

```
2015-08-08 12:38:50,793 [INFO] [192.168.164.139:42604<=>127.0.0.1:38068 7ad540ca-69eb-
4efa-89a8-700acb4659d6 logging](Unknown) Selected for connection
2015-08-08 12:38:50,794 [INFO] Failed to connect to endpoint 127.0.0.1:38068 errno 111
2015-08-08 12:38:50,794 [INFO] [192.168.164.139:42604<=>127.0.0.1:38068 7ad540ca-69eb-
4efa-89a8-700acb4659d6 logging](Unknown) Connection closed
2015-08-08 12:38:50,800 [INFO] [192.168.164.139:42605<=>192.168.1.50:80 6e7b2e05-2b64-
41c0-ba02-14a4e1513da0 logging](Unknown) Selected for connection
2015-08-08 12:38:50,804 [INFO] [192.168.164.139:42605<=>192.168.1.50:80 6e7b2e05-2b64-
41c0-ba02-14a4e1513da0 logging](Unknown) Connection established
2015-08-08 12:38:50,811 [ERROR] [192.168.164.139:42605<=>192.168.1.50:80 6e7b2e05-
2b64-41c0-ba02-14a4e1513da0 httpdetection](Unknown) HTTP request GET
192.168.1.50/empresajson.txt
2015-08-08 12:38:50,850 [INFO] [192.168.164.139:42605<=>192.168.1.50:80 6e7b2e05-2b64-
41c0-ba02-14a4e1513da0 logging](Unknown) Connection closed
```

Cifrado no TLS: Opción --noTLS en cliente java

```
2015-08-08 12:30:42,896 [INFO] Starting...
2015-08-08 12:30:47,950 [INFO] [192.168.164.139:42568<=>127.0.0.1:37377 00cdfb44-1e30-
451a-9494-a32fa63e23ae logging](Unknown) Selected for connection
2015-08-08 12:30:47,951 [INFO] Failed to connect to endpoint 127.0.0.1:37377 errno 111
2015-08-08 12:30:47,952 [INFO] [192.168.164.139:42568<=>127.0.0.1:37377 00cdfb44-1e30-
451a-9494-a32fa63e23ae logging](Unknown) Connection closed
2015-08-08 12:30:48,004 [INFO] [192.168.164.139:42569<=>192.168.1.50:443 44b4583a-
9e01-4387-a4b9-302b47f735b9 logging](Unknown) Selected for connection
2015-08-08 12:30:48,006 [INFO] [192.168.164.139:42569<=>192.168.1.50:443 44b4583a-
9e01-4387-a4b9-302b47f735b9 logging](Unknown) Connection established
```

```
2015-08-08 12:30:48,020 [ERROR] [192.168.164.139:42569<=>192.168.1.50:443 44b4583a-
9e01-4387-a4b9-302b47f735b9 insecurecipherdetection](Unknown) Client enabled export
TLS/SSL cipher suites SSL2_RC4_128_EXPORT40_WITH_MD5, TLS_RSA_EXPORT_WITH_RC4_40_MD5,
SSL2_RC4_128_EXPORT40_WITH_MD5, TLS_RSA_EXPORT_WITH_DES40_CBC_SHA,
TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA, TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
2015-08-08 12:30:48,023 [ERROR] [192.168.164.139:42569<=>192.168.1.50:443 44b4583a-
9e01-4387-a4b9-302b47f735b9 weaktlsversiondetection](Unknown) Client enabled SSLv2
protocol
2015-08-08 12:30:49,799 [INFO] [192.168.164.139:42570<=>192.168.1.50:2560 5e131045-
b7dd-46d9-81a8-1c9950d83639 logging](Unknown) Selected for connection
2015-08-08 12:30:49,803 [INFO] [192.168.164.139:42570<=>192.168.1.50:2560 5e131045-
b7dd-46d9-81a8-1c9950d83639 logging](Unknown) Connection established
2015-08-08 12:30:49,810 [ERROR] [192.168.164.139:42570<=>192.168.1.50:2560 5e131045-
b7dd-46d9-81a8-1c9950d83639 httpdetection](Unknown) HTTP request POST
192.168.1.50:2560/
2015-08-08 12:30:49,853 [INFO] [192.168.164.139:42570<=>192.168.1.50:2560 5e131045-
b7dd-46d9-81a8-1c9950d83639 logging](Unknown) Connection closed
2015-08-08 12:30:49,908 [INFO] [192.168.164.139:42569<=>192.168.1.50:443 44b4583a-
9e01-4387-a4b9-302b47f735b9 logging](Unknown) Connection closed
```

Cliente (cURL) vulnerable a ataque heartbleed

```
2015-09-04 15:09:17,494 [INFO] Starting...

2015-09-04 15:09:22,237 [INFO] [192.168.164.147:47425<=>127.0.0.1:3128 bd7620ea-2c97-
434b-ad20-99c51591b9fa logging](Unknown) Selected for connection

2015-09-04 15:09:22,238 [INFO] [192.168.164.147:47425<=>127.0.0.1:3128 bd7620ea-2c97-
434b-ad20-99c51591b9fa logging](Unknown) Connection established

2015-09-04 15:09:22,241 [ERROR] [192.168.164.147:47425<=>127.0.0.1:3128 bd7620ea-2c97-
434b-ad20-99c51591b9fa httpdetection](Unknown) HTTP request CONNECT
192.168.1.53:443192.168.1.53:443

2015-09-04 15:09:22,249 [INFO] [192.168.164.147:47425<=>127.0.0.1:3128 bd7620ea-2c97-
434b-ad20-99c51591b9fa logging](Unknown) Handler being removed

2015-09-04 15:09:22,249 [INFO] [192.168.164.147:47425<=>127.0.0.1:3128 bd7620ea-2c97-
434b-ad20-99c51591b9fa clientheartbleed](Unknown) Selected for connection

2015-09-04 15:09:22,255 [ERROR] [192.168.164.147:47425<=>127.0.0.1:3128 bd7620ea-2c97-
434b-ad20-99c51591b9fa insecurecipherdetection](Unknown) Client enabled export TLS/SSL
cipher suites TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA,
TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA, TLS_RSA_EXPORT_WITH_DES40_CBC_SHA,
TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5, TLS_RSA_EXPORT_WITH_RC4_40_MD5

2015-09-04 15:09:22,388 [CRITICAL] [192.168.164.147:47425<=>127.0.0.1:3128 bd7620ea-2c97-
434b-ad20-99c51591b9fa clientheartbleed](Unknown) Heartbleed response received

2015-09-04 15:09:22,403 [INFO] [192.168.164.147:47425<=>127.0.0.1:3128 bd7620ea-2c97-
434b-ad20-99c51591b9fa clientheartbleed](Unknown) Connection closed
```

REFERENCIAS

1. **ITU-T.** Recomendación X.509. *Tecnología de la información - Interconexión de sistemas abiertos - El directorio: Marcos para certificados de claves públicas y atributos.* Octubre 2012.

2. **M. Myers, R. Ankney, A. Malpani, S. Galperin, C. Adams.** X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. *IETF RFC 2560.* 1999.

3. **Oppliger, Rolf.** *SSL and TLS: theory and practice* . s.l. : Artech House, 2009.

4. **T. Dierks, E. Rescorla.** The Transport Layer Security (TLS) Protocol version 1.2. *IETF RFC 5246.* 2008.

5. **OpenSSL Project.** SSL - OpenSSL SSL/TLS library. [En línea] https://www.openssl.org/docs/ssl/ssl.html.

6. **JSON.org.** JSON in Java. [En línea] 2002. http://www.json.org/java/.

7. **The Legion of Bouncy Castle Inc.** The Legion of the Bouncy Castle Java Cryptography APIs. [En línea] 2013. https://www.bouncycastle.org/java.html.

8. **OpenSSL Project.** openssl - OpenSSL command line tool. [En línea] https://www.openssl.org/docs/apps/openssl.html.

9. **García, Roberto de Miguel.** *Criptografía clásica y moderna.* s.l. : Septem Ediciones, 2009.

10. **Johannes A. Buchmann, Evangelos Karatsiolis, Alexander Wiesmaier.** *Introduction to Public Key Infrastructures.* s.l. : Springer, 2013.

11. **Indra Sistemas, SA.** Infraestructura de Clave Pública (PKI) - Incibe. [En línea] 2005. https://www.incibe.es/extfrontinteco/es/pdf/Formacion_PKI.pdf.

12. **Project, Massimiliano Pala and OpenCA.** OpenCA Research Labs - OCSP Responder. [En línea] https://pki.openca.org/projects/ocspd/.

13. **PrimeKey Solutions AB.** EJBCA - Open Source PKI Certificate Authority - OCSP Architecture. [En línea] 2002-2015. http://www.ejbca.org/docs/architecture-ocsp.html.

14. **Free Software Foundation, Inc.** GNU Wget. [En línea] 2010. http://www.gnu.org/software/wget/.

15. **The Apache Software Foundation.** The Apache HTTP Server Project. [En línea] 1997-2015. http://httpd.apache.org/.

16. **Free Software Foundation, Inc.** GNU sed. [En línea] 2014. http://www.gnu.org/software/sed/.

17. **Google.** google/nogotofail. [En línea] 2015. https://github.com/google/nogotofail/.

18. **Open Source Project .** ProxyChains - TCP and DNS through proxy server. HTTP and SOCKS. [En línea] http://proxychains.sourceforge.net/.

19. **Wikipedia.** Infraestructura de clave pública - Wikipedia. [En línea] 2014. https://es.wikipedia.org/wiki/Infraestructura_de_clave_p%C3%BAblica.

20. **Internet Security Research Group.** Let's Encrypt. [En línea] 2015. https://letsencrypt.org/.

21. **Wikipedia.** Transport Layer Security - Wikipedia. [En línea] 2015. https://es.wikipedia.org/wiki/Transport_Layer_Security.

22. **Fernández, Godofredo y Nebrera, Pablo.** Apuntes Seguridad 3º GITT Universidad de Sevilla. 2014.

23. **Ellingwood, Justin.** How To Create a SSL Certificate on Apache for Ubuntu 14.04. [En línea]

https://www.digitalocean.com/community/tutorials/how-to-create-a-ssl-certificate-on-apache-for-ubuntu-14-04.

24. Quickly using OpenSSL in C. [En línea] http://savetheions.com/2010/01/16/quickly-using-openssl-in-c/.

25. **Guillermo Julián.** HTTPS Así funciona. [En línea] http://www.genbeta.com/web/https-asi-funciona.

26. **Sánchez, Cristina.** Criptografía cuántica y adiós a los espías: ¿la esperanza de un mundo sin ciberataques? *eldiario.es.* 2014, http://www.eldiario.es/hojaderouter/seguridad/criptografia-cuantica-seguridad-ciberespionaje_0_315669050.html.

27. **Yébenes Moreno, Sergio.** Sistema de control de acceso en Redes Wireless con el DNI electrónico. 2009.

28. **Marín Carreño, David.** Sistema libre para la gestión de una autoridad de certificación X.509: gnoMint. 2009.

29. **Stemberg, Daniel.** curl and libcurl. [En línea] http://curl.haxx.se/.

30. **Bonet Esteban, Enrique V.** Creación y administración de certificados de seguridad mediante OpenSSL - Apuntes Administración y Gestión de Redes Doble Titulación Informática+Telemática Universidad de Valencia. [En línea] http://informatica.uv.es/it3guia/AGR/apuntes/teoria/presentaciones/Certificados.pdf.