Universitat de les Illes Balears

escola politècnica superior

Trabajo Fin de Grado

GRADO EN INGENIERÍA TELEMÁTICA

Implementación de un módulo de análisis de la seguridad de conexiones web

LLORENÇ SAMPOL BENNASAR

Tutor
Dr. María Francisca Hinarejos Campos

Escola Politècnica Superior
Universitat de les Illes Balears
Palma, 3 de julio de 2018

ÍNDICE GENERAL

Acrónimos

OCSP Online Certificate Status Protocol

HTTPS Hypertext Transfer Protocol Secure

SSL Secure Sockets Layer

TLS Transport Layer Security

CA Certificate Authority

EV Extended Validation

API Application Programming Interface

UI User Interface

JS JavaScript

URI Uniform Resource Identifier

CSS Cascading Style Sheets

URL Uniform Resource Locator

OID Object IDentifier

NIST National Institute of Standards and Technology

Resumen

El extenso uso del navegador para el acceso a servicios como la banca, compras, cursos, etc, se ha hecho cada vez más frecuente, así como también la aparición de robos de información relacionados con estos. Entre los ataques informáticos que han ido apareciendo, existen algunos relacionados con el establecimiento de una conexión Web. Por ejemplo, se puede encontrar el *downgrade attack* sobre el protocolo Transport Layer Security (TLS), que puede forzar el establecimiento de la conexión con un protocolo inseguro.

Debido a estos ataques los usuarios cada vez son más conscientes de la importancia de la seguridad en estos entornos, lo que podría llegar a generar una falta de confianza en estos servicios si no se hace algo al respecto. Es necesario aportar a los usuarios una herramienta fácilmente accesible para que estos puedan navegar con más tranquilidad y fiabilidad.

Por ello, el objetivo de este trabajo será desarrollar un módulo que permita analizar la seguridad de una conexión Web. Realizando un estudio previo sobre qué parámetros son importantes a la hora de establecer una comunicación y cómo pueden valorarse todos ellos en conjunto.

CAPÍTULO 1

INTRODUCCIÓN

La seguridad en las conexiones Web consiste en conseguir crear un canal de comunicación entre cliente y servidor que posea las propiedades adecuadas: confidencialidad, integridad y autenticidad.

En algunos servicios online como la banca es necesario verificar de alguna forma que dicho sitio Web es el que dice ser. Además, la información que se intercambia con el servidor debe ser exactamente la misma que la que se envía, sin ninguna modificación de por medio. Y por supuesto, las credenciales que este utiliza para acceder a sus cuentas, así como también los trámites que este lleva a cabo, deben ser totalmente confidenciales. Para poder satisfacer todas estas necesidades es necesario llevar a cabo una serie de verificaciones y negociar ciertos parámetros con el navegador, que marcarán la seguridad de la conexión.

1.1. Protocolos criptográficos

Los protocolos criptográficos marcan las pautas necesarias para que cliente y servidor puedan construir un canal de comunicaciones seguro. Por ello, es necesario analizar el funcionamiento de estos para poder llevar a cabo una buena valoración de la conexión Web.

Para el establecimiento de comunicaciones seguras a nivel Web, Secure Sockets Layer (SSL) y TLS son los protocolos ampliamente utilizados, aunque SSL ya es considerado obsoleto y no debería utilizarse (RFC 7568). Para poder realizar un análisis de los diferentes parámetros que intervienen en el establecimiento de una conexión Web segura se analizará el funcionamiento de las diferentes versiones del protocolo TLS, las cuales constan de varios pasos que llevan a cabo el cliente y el servidor para poder acordar un algoritmo de cifrado simétrico (junto con la clave compartida), y un algoritmo de hash.

Estos se componen de cuatro fases fundamentales, con una funcionalidad cada una de ellas:

- Fase I: establecer la versión de TLS que se utilizará, así como también el *cipher suite*. Este está compuesto por tres partes principales que son el algoritmo de intercambio de claves, el algoritmo de clave simétrica para el cifrado de información y el algoritmo de hash para determinar la integridad del mensaje.
- Fase II: el servidor envía su certificado al cliente para que este pueda autenticarlo, junto a algunos parámetros para poder establecer una clave simétrica de cifrado.
- Fase III: el cliente envía los parámetros necesarios al servidor para que este pueda determinar la misma clave simétrica.
- Fase IV: el cliente y el servidor se ponen de acuerdo para que de aquí en adelante todo el tráfico intercambiado entre ellos esté cifrado.

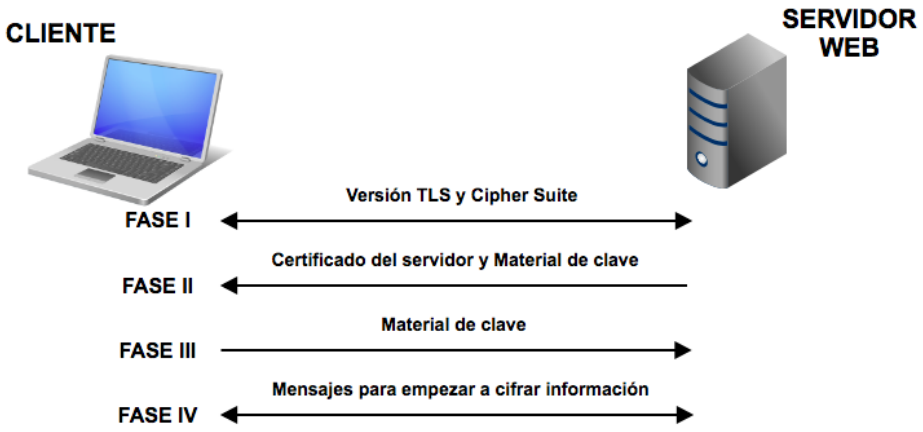

Figura 1.1: Fases del protocolo TLS

Además, cabe mencionar que estas cuatro fases se ejecutan cada vez que se inicia una nueva sesión entre un cliente y un servidor. Durante dicha sesión el cliente y el servidor pueden desconectarse, pero si vuelven a establecer un canal antes de que caduque dicha sesión se seguirán utilizando los mismos parámetros negociados al inicio de la sesión (*cipher suite* y claves simétricas).

Para autenticar a las partes se utilizan certificados X.509, los cuales pueden pertenecer tanto a los clientes como a los servidores, pero en este proyecto no se analizarán las variantes que necesiten del uso de un certificado por parte del cliente. Como se puede observar en la figura 1.2, un certificado X.509 está compuesto por varios campos que aportan información diferente acerca de este, pero de entre todos ellos debe destacarse la clave pública. Esta es la pareja de una clave privada que tan solo posee el propietario del certificado, y que este utiliza para poder autenticarse con el cliente.

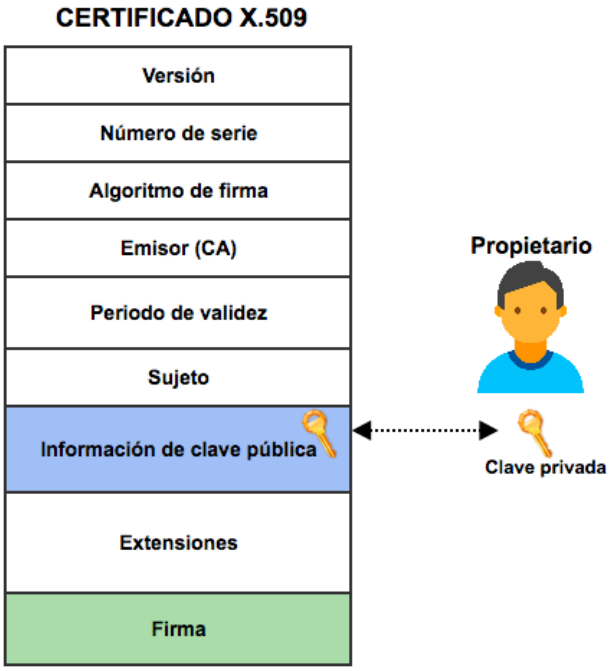

Figura 1.2: Certificado X.509

Además, es necesario que exista un mecanismo para poder verificar la autenticidad de un certificado X.509, el cual es la firma de una Certificate Authority (CA). Para poder entender el funcionamiento de esta firma se deben conocer las autoridades de certificación, que son las encargadas de expedir nuevos certificados. Estas también poseen un certificado, el cual puede ser clasificado como autofirmado o no autofirmado. Los certificados autofirmados son los que han sido firmados por la misma CA del certificado utilizando su clave privada. Por otro lado, los que no están autofirmados han sido firmados por otra CA.

En la figura 1.3 se puede observar un ejemplo de una cadena de certificación. Esta se compone por el certificado del servidor Web, una CA intermedia y una CA raíz. El servidor Web posee la firma de la CA intermedia, la cual a su vez está firmada por la CA raíz. Por otra parte, la CA raíz posee un certificado autofirmado, es decir, se verifica a si misma y a toda la cadena de certificación. Además, al ser su certificado autofirmado, la confianza que pueda aportar la cadena recae sobre la CA raíz. No obstante, la decisión final pertenece al navegador o al usuario, ya que este decide si aceptar la cadena de certificación o no.

Dicho mecanismo se utiliza para poder formar una cadena de certificados firmados unos a otros y que pueda contener una raíz, la cual posee un certificado autofirmado (figura 1.3). Por otro lado, al final de la cadena se encontrará el certificado del servidor Web, el cual vendrá verificado por todos los certificados de las CAs que conforman la cadena.

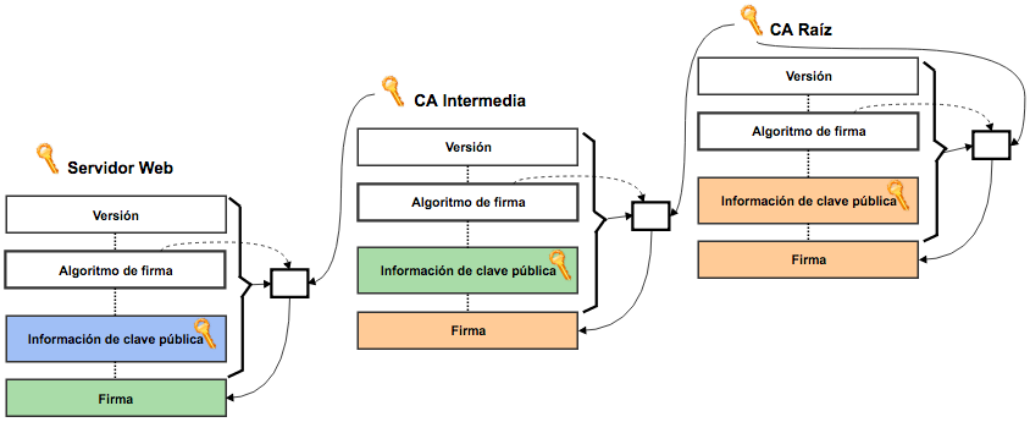

Figura 1.3: Funcionamiento de la cadena de certificación

1.2. Objetivos del proyecto

Una vez se ha explicado el funcionamiento de los protocolos criptográficos, se expondrán cuáles son los objetivos en este proyecto:

- Analizar el funcionamiento del protocolo TLS y destacar qué parámetros determinan la seguridad de una conexión Web.
- Elegir una tecnología para poder desarrollar un módulo de análisis, que permita acceder y analizar la información de seguridad de una conexión Web.
- Desarrollar un algoritmo de prueba que permita evaluar la seguridad de una conexión Web, teniendo en cuenta los parámetros que determinan dicha seguridad.
- Implementar un módulo de análisis de la seguridad de la conexión Web con la tecnología seleccionada, que incorpore el algoritmo de prueba desarrollado.

1.3. Estructura de la memoria

Para finalizar este apartado se detallará el recorrido que seguirá la memoria. Esta se encuentra divida en dos grandes partes:

- En la primera parte se realizará un estudio previo sobre la tecnología que se utilizará para implementar el nuevo módulo de análisis. Para empezar se analizará el funcionamiento de los protocolos criptográficos que marcan las pautas para establecer una conexión Web, y se determinará qué parámetros son importantes para la seguridad de esta conexión. A continuación se determinará qué tecnología se utilizará para implementar el módulo de análisis de la conexión Web. Para ello, se analizará el funcionamiento de un módulo ya existente, el cual permita acceder a la información de seguridad de la conexión.

- En la segunda parte se investigará cómo obtener los parámetros necesarios para implementar el módulo de análisis. Una vez se hayan determinado definitivamente cuáles son accesibles a través de la tecnología seleccionada, se desarrollará un algoritmo de prueba para valorar la seguridad de una conexión Web. Finalmente, se llevará a cabo la implementación del nuevo módulo de análisis, en el cual se detallarán los cambios introducidos respecto a su predecesor. Y se realizarán una serie de pruebas para determinar el correcto funcionamiento de este nuevo módulo.

CAPÍTULO 2

ESTUDIO PREVIO A LA IMPLEMENTACIÓN DEL MÓDULO

En este apartado se identificarán los parámetros que deben analizarse para valorar la seguridad de una conexión Web. Además, se elegirá una herramienta para llevar a cabo el análisis de esta conexión. Dicha herramienta debe permitir el acceso a la información de seguridad de la conexión, la aplicación de un algoritmo de puntuación sobre esta, y además, debe mostrar los resultados de una forma sencilla para el usuario. Una herramienta que podría poseer todas estas características es una extensión de Firefox. Por ello, la primera parte de este apartado se centrará en explicar el funcionamiento de este tipo de complementos de Firefox.

Una vez se haya explicado en qué consiste una extension y cómo puede desarrollarse, se buscará alguna ya existente que implemente funciones que puedan ser reutilizadas en este trabajo. En caso de encontrarse una extensión, se analizará el funcionamiento de esta para saber cómo extraer los datos necesarios para el proyecto y cómo añadir nuevas funciones en la extensión.

2.1. Parámetros a analizar

Teniendo en cuenta las propiedades que hacen segura una comunicación Web y los protocolos existentes para llevarla a cabo (apartado 1.1), se determinará cómo se consiguen dichas propiedades y en qué medida, gracias a diferentes parámetros.

Para empezar, los certificados X.509 son una pieza fundamental para poder autenticar al servidor a través de la firma digital que se introduce en el intercambio de claves del protocolo TLS. Este intercambio de claves se puede llevar a cabo de diferentes maneras a través de varios algoritmos, pero si no incluye uno que proporciona una firma digital, el servidor no puede probar su autenticidad y por lo tanto el cliente no podrá estar seguro de si el servidor es el auténtico.

Además, para que el certificado sea fiable no debe tan solo cumplirse la condición de que la cadena esté completa hasta una CA Raíz de confianza. Sino que también existen

otros parámetros del certificado que deberían comprobarse para dar más fiabilidad a la comunicación. Algunos de estos datos se pueden valorar de igual manera, sea cual sea el propietario del certificado (servidor o CA), puesto que aparecen en cualquier tipo de certificado. Otros parámetros pueden no aparecer en algunos certificados o pueden tomar valores distintos en un tipo de certificado u otro, puesto que les atribuyen propiedades concretas.

Para empezar, algunos parámetros que comparten todos los certificados y pueden puntuarse de igual manera son:

- Algoritmo de firma: algoritmo utilizado por la CA firmante para firmar el certificado.
- Periodo de validez: compuesto por dos campos del certificado que indican a partir de qué momento y hasta cuándo es válido.
- Clave pública: esta es la pareja de la clave asimétrica privada que posee el propietario del certificado, puede ser de diferentes tipos y longitudes.
- Firma del certificado: es el resultado que obtiene la CA al aplicar el algoritmo de firma sobre el certificado.

Los parámetros que no aparecen en todos los certificados de la cadena o que aportan cualidades distintas a cada uno de estos, se encuentran dentro de la sección de extensiones del certificado. Estos parámetros se han clasificado en dos tipos, críticos (*) y no críticos, característica que se explicará más adelante. Los parámetros que se pueden encontrar en el campo de extensiones son:

- *Certificate Subject Alt Name**: es una extensión propia de los certificados de servidor Web, esta contiene todos los nombres de dominio (en el caso de servidores Web) para las cuales es válido dicho certificado.
- *Certificate Key Usage**: es un parámetro que indica para qué puede utilizarse dicho certificado. En el caso de pertenecer a una CA se le permite firmar certificados, pero el certificado de un servidor Web tan solo puede utilizarse para autenticarse y/o cifrar la información durante el intercambio de claves.
- *CRL Distribution Points*: contiene las URIs donde se puede encontrar el fichero *.crl* que contiene los certificados revocados por la CA que ha firmado el certificado. Este parámetro puede encontrarse en cualquier certificado de la cadena excepto en la raíz.
- *Authority Information Access*: incluye las URIs a las cuales se debe realizar la petición Online Certificate Status Protocol (OCSP) para poder obtener la información de revocación del certificado. Al igual que la extensión anterior puede encontrarse en cualquier parte de la cadena de certificados excepto en la raíz.
- *Certificate Basic Constraints**: indica si el propietario del certificado es una CA, y en caso de que lo sea indica el número de CAs intermedias a las cuales este puede expedirles un certificado.

- *Certificate Policies*: en el caso de los certificados de servidores estos pueden someterse a un proceso de validación de identidad para poder obtener un certificado con Extended Validation (EV). Este tipo de certificados incluye en este apartado el OID de la CA que lo ha expedido.

Estas extensiones proporcionan información importante para comprobar el estado de revocación de los certificados, así como otros datos relevantes sobre estos, por ejemplo si este es *Extended Validated*. Pero lo más importante es comprobar que los campos considerados críticos (*) contengan los valores correctos. Por ejemplo, no tiene sentido que un certificado de una CA intermedia tenga una extensión que indique que no es una autoridad certificadora. Dicho dato debería puntuarse muy negativamente, puesto que la base de la autenticación de una comunicación segura se basa en que la cadena de certificación del servidor Web sea fiable, pero si se encuentran fallos en parámetros críticos deberá concluirse que no lo es.

Por otra parte, también están los parámetros que se negocian para una sesión entre el cliente y el servidor, el *cipher suite*. Dentro de este se encuentra el algoritmo de hash que permite verificar la integridad de la información, el sistema de cifrado simétrico que proporcionará privacidad a la comunicación. Y además, durante el intercambio de claves se puede utilizar la clave pública del servidor Web para firmar la información intercambiada, autenticándose este en el proceso.

Dentro del *cipher suite* cabe destacar una característica que puede aportar el algoritmo de intercambio de clave, la cual recibe el nombre de *perfect forward secrecy*. Esta se proporciona a la comunicación cuando el material de claves que se acuerda en cada una de las sesiones es totalmente nuevo y aleatorio. Los algoritmos que proporcionan esta propiedad son los que renuevan dicho material de clave, los cuales son las versiones efímeras de *Diffie-Hellman* (DHE y ECDHE).

Por lo tanto, no se pueden valorar independientemente cada uno de los parámetros que cubren las propiedades mencionadas al principio del apartado (confidencialidad, integridad y autenticidad). Puesto que el *cipher suite* se puede establecer gracias a un largo proceso que también debe reflejarse en la valoración global de la conexión Web.

2.2. Extensiones de Firefox

El navegador de Firefox permite el desarrollo de aplicaciones para que puedan interactuar con este, las cuales reciben el nombre de complementos. Uno de esos complementos, son las extensiones, las cuales permiten añadir funcionalidades adicionales al navegador a través de lenguajes Web (JavaScript (JS), HTML y CSS). Además, a partir de Firefox 57 se cambió la manera en que estas se comunicaban con el navegador, y con ello cambió también la forma en la que podían desarrollarse. Estas nuevas extensiones se desarrollan a partir de una Application Programming Interface (API) denominada WebExtensions, la cual se analizará a continuación y se observará porqué no puede utilizarse esta tecnología para desarrollar este proyecto.

2.2.1. WebExtensions

A partir de Firefox 57 se introdujo el concepto de WebExtensions que cambió en gran medida la forma de desarrollar una extensión, esto se hizo con algunos objetivos

en mente como, mejorar la portabilidad de extensiones de otros navegadores. Pero una de la grandes mejoras que tuvo para los programadores novatos fue la gran facilidad que les aportaba para poder desarrollar una extensión, o para realizar cambios sobre una ya existente sin que esta presentara fallos.

Aunque dicha mejora vino acompañada también de ciertas limitaciones, puesto que antes de Firefox 57 se podía hacer uso de una extensa variedad de interfaces cuyas funcionalidades aún no se han incluído dentro de la nueva API, debido sobretodo a que este cambio se ha producido muy recientemete (14 de noviembre de 2017).

Para poder consultar la nueva API de Firefox se debe acceder a la página web de desarrolladores [1] en la cual podemos encontrar el listado de funcionalidades que esta puede proporcionar a una extensión. Una vez revisada esta lista, no se ha encontrado ninguna de ellas que permita acceder a la información de seguridad necesaria (apartado 2.1). Para reforzar la decisión de no utilizar WebExtensions en este trabajo se ha encontrado información que indica que dicha API se encuentra todavía en desarrollo [2].

2.2.2. Estructura antes de Firefox 57

Puesto que como se ha explicado no es posible acceder a la información de seguridad del navegador a través de WebExtensions se desarrollará una extensión con la estructura que estas tenían antes de que apareciese esta API. Esta estructura podía ser como quisiese el programador, pero debía contener en algún lugar los siguientes ficheros y directorios:

- *install.rdf*: es el fichero que se utiliza para determinar la información sobre la extensión de Firefox cuando se va a instalar. Contiene diferentes metadatos que proporcionan información sobre este, de entre los cuales hay algunos que es obligatorio introducirlos:

    - *id*: es el identificador único de la extensión y puede ser de dos tipos distintos: GUID o una cadena de caracteres parecida a un email. La primera opción es mucho más difícil de generar por lo que se recomienda utilizar el formato de email que presenta el siguiente formato: *nombreDeLaExtension@ejemplo*. La única condición necesaria que debe cumplir dicho id es que sea único, es decir, mientras se utilice un nombre lo suficientemente identificativo, no será necesaria la generación de un GUID.
    - *name*: es el nombre de la extensión que se mostrará por la User Interface (UI).
    - *targetApplication*: en este campo se debe indicar el id único de la aplicación así como también la versión mínima y máxima que esta soporta del navegador. Cabe destacar, que la versión máxima que se indique en este campo permitirá la instalación de la extensión en versiones superiores de Firefox, puesto que tan solo indica hasta que versión se han realizado pruebas con la extensión.
    - *type*: es un número entero que representa el tipo de complemento de Firefox. En este caso es el número “2”, que se utiliza para las extensiones.

  - *version*: es una cadena de caracteres que indica la versión actual de la extensión.

- *chrome.manifest*: el *chrome* se refiere al conjunto de elementos que se encuentran fuera de la ventana de contenido, como por ejemplo las barras de menú del navegador. El *chrome* está compuesto por tres partes necesarias para la mayoría de las extensiones: *content*, *skin* y *locale*. Como podemos ver en la figura 2.1 este fichero se construye de manera que cada línea posea una serie de columnas con con una funcionalidad diferente cada una de ellas:

  - La primera columna se utiliza para indicar a Firefox qué parámetro se está declarando.
  - La segunda es el nombre del paquete que compilará el navegador.
  - La tercera columna no aparece cuando en la primera se hace referencia al *content*, pero en los otros dos casos sí (*skin* y *locale*). El motivo de esto es que puede haber múltiples entradas de *skin* o *locale*, y debe haber un campo identificativo para cada una de ellas.
  - La última columna indica la ruta hacia el fichero que contiene los archivos necesarios para cada caso, los cuales se explicarán a continuación.

- Directorio *content*: contiene la UI y los *scripts* (JS), que son las piezas fundamentales de la extensión. Se utilizan ficheros *.xul* (lenguaje XML para interfaces de usuario de Mozilla) para poder definir la UI.

- Directorio *skin*: incluye los archivos que darán apariencia a la UI mediante el uso de ficheros CSS e imágenes.

- Directorio *locale*: almacena los ficheros que contienen todo el texto que se mostrará por pantalla. Los archivos *propierties* y *.dtd* proporcionan mayor flexibilidad a la hora de alternar entre distintos idiomas.

chrome.manifest

```
1  skin        ssleuth     classic/1.0             content/skin/
2  content     ssleuth     content/
3
4  locale      ssleuth     de-DE       locale/de-DE/
5  locale      ssleuth     en-US       locale/en-US/
6  locale      ssleuth     fr-FR       locale/fr-FR/
7  locale      ssleuth     pl-PL       locale/pl-PL/
```

Figura 2.1: Fichero *chrome.manifest*

2.2.3. Cargar complementos en Firefox

Para poder desarrollar una extensión es necesario saber como poder compilarla en el navegador de Firefox, con el objetivo de ir probando los cambios que se vayan realizando en esta. Para ello se debe introducir en la URL del navegador la dirección

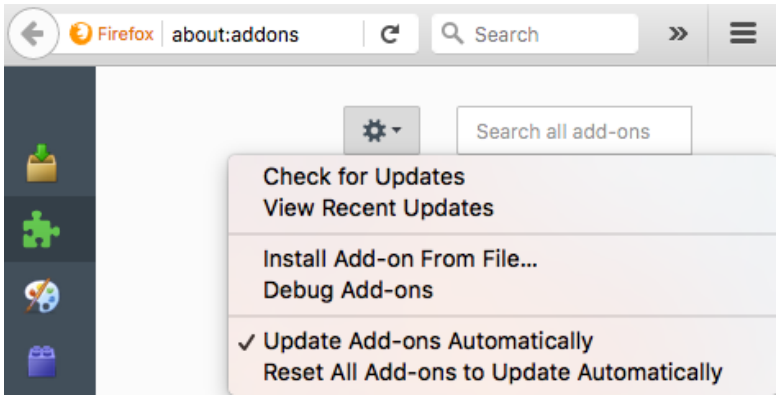

Figura 2.2: Página de gestión de las extensiones de Firefox

donde se gestionan las extensiones, *about:addons*, o también puede accederse a esta página a través de la barra de herramientas.

Como se puede observar en la figura 2.2 se proporcionan una serie de opciones entre las que se pueden encontrar *Install Add-on From File* y *Debug Add-ons*. En ambas opciones se deberá seleccionar el archivo *install.rdf*, pero hay una diferencia principal entre ellas. La primera permite instalar una extensión que permanecerá en el navegador incluso cuando este se reinicie, pero la otra tan solo permitirá depurarla, lo que significa que una vez se cierre el navegador desaparecerá. La depuración es la opción apropiada para poder ir probando una extensión en desarrollo porque esta se ejecuta más rápidamente , y además, para poder instalar una extensión en el navegador se necesita de una verificación de esta. Si se intenta instalar una extensión sin antes haberla verificado, el navegador muestra por pantalla un error y no permite realizar dicha acción (figura 2.3). Este cambio se introdujo a partir de Firefox 44, cuando se eliminó la opción de modificar el parámetro de la configuración interna del navegador que permitía instalar una extensión no verificada (*xpinstall.signatures.required*).

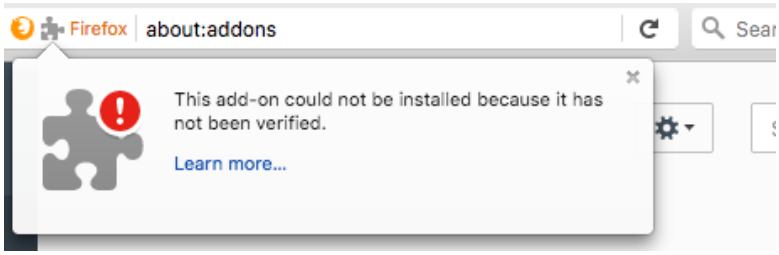

Figura 2.3: Error al instalar extensión no verificada

Una vez se selecciona la opción *Debug Add-ons* el navegador accederá a la página *about:debugging#addons* (figura 2.4) donde se pueden controlar todas las extensiones que se estén depurando en el navegador. La opción que debe seleccionarse para poder depurar una nueva extensión es *Load Temporary Add-on*, que solicitará que se seleccione un archivo (*install.rdf*). Una vez este se haya seleccionado podrá observarse como ha aparecido una nueva extensión en la página, permitiendo realizar dos opciones, *Debug* y *Reload*. La primera sirve para acceder a la herramienta de depuración de

Firefox y la segunda permite volver a cargar el archivo *install.rdf* automáticamente, por si se quieren ir probando los cambios realizados en el código de la extensión.

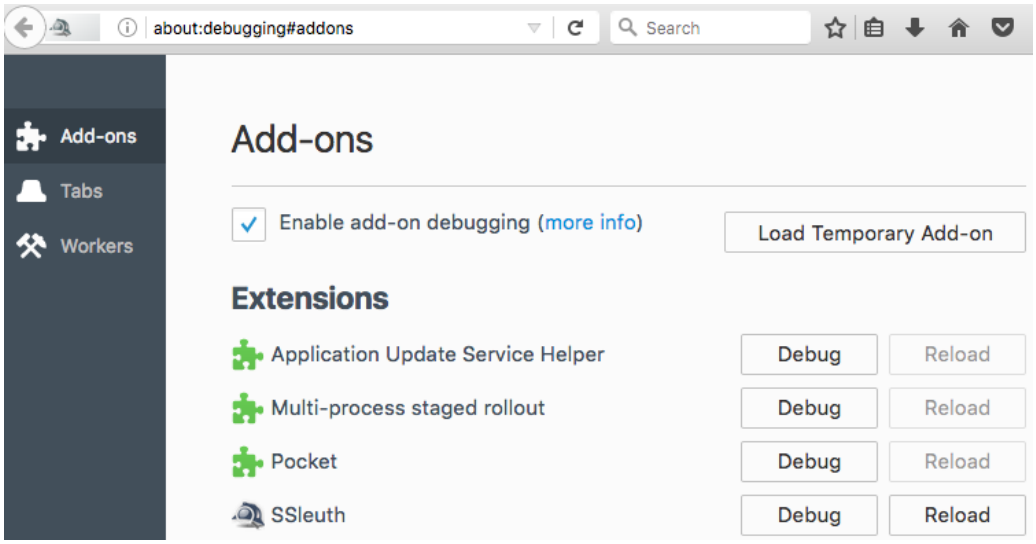

Figura 2.4: Cargar temporalmente un *add on*

2.3. Análisis de la extensión SSLeuth

Como ya se ha explicado al inicio de este capítulo, se intentará reutilizar funcionalidades de extensiones ya existentes para poder desarrollar un módulo de análisis de la conexión. De entre todas las que se pueden encontrar se han analizado cuatro principalmente: *Certificate Patrol*, *CipherFox*, *Calomel SSL Validation* y *SSLeuth*.

Certificate Patrol consiste en una herramienta que permite al usuario gestinar los certificados del navegador. Esta muestra qué certificados tiene configurados como válidos el navegador. Así como también puede avisar al usuario cuando se accede a un servidor que posee un certificado desconocido por el navegador.

CipherFox permite acceder a la información de seguridad de la conexión. Con la ayuda de una pestaña muestra al usuario el *cipher suite* de la conexión Web, y además permite acceder a toda la información de la cadena de certificados.

Como se ha podido observar, en la extensión *Certificate Patrol* no se accede a la información de seguridad de la conexión. Y aunque *CipherFox* parece proporcionar la misma información de seguridad que *SSLeuth* y Calomel SSL Validation, no realiza ninguna valoración de la conexión. Por el contrario, *Calomel SSL Validation* y *SSLeuth* son capaces de acceder a la información de seguridad del navegador y valorar la seguridad de la conexión. Además, ambas extensiones parecen proporcionar y evaluar los mismos parámetros de seguridad, aplicando sistemas de puntuación distintos. Por ello, se ha decidido analizar el funcionamiento de una de estas dos extensiones, *SSLeuth*. Esta posee además otras funcionalidades que se detallarán a continuación.

2.3.1. Funcionalidades

Como se puede observar en la figura 2.5 la función más relevante de *SSLeuth* es proporcionar al usuario una valoración global de la seguridad de la conexión. Esta consiste en un valor que puede ir desde 0 hasta 10, y que además viene acompañada de la información de seguridad de la conexión. El panel que contiene dicha información tiene otras dos pestañas llamadas *Domains* y *Cipher suite*, pero las funcionalidades que estas proporcionan no se analizarán en este proyecto.

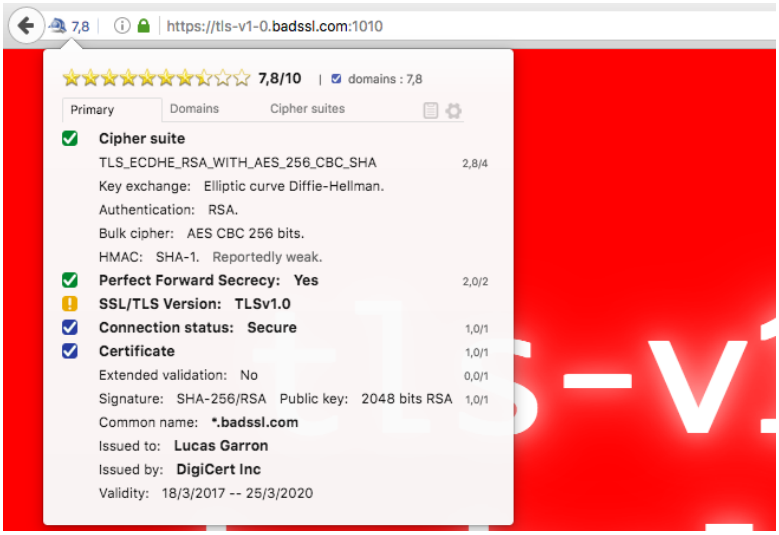

Figura 2.5: Panel HTTPS de ssleuth

Se puede observar como esta extensión es capaz de distinguir cuando se utiliza el protocolo HTTP y Hypertext Transfer Protocol Secure (HTTPS) (figura 2.6). Además de proporcionar al usuario la posibilidad de acceder fácilmente al dominio HTTPS de la página Web, en caso de que este existiese.

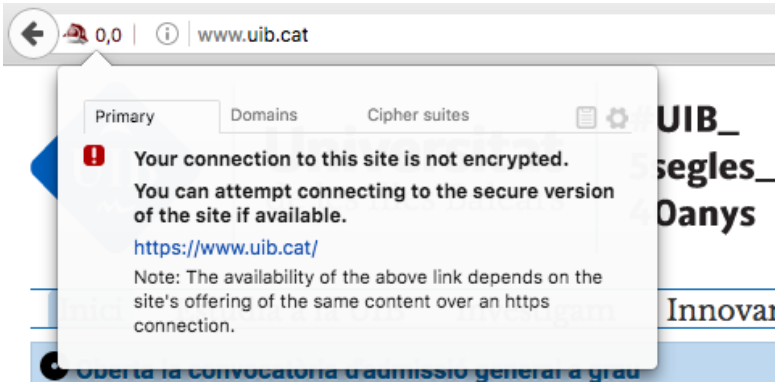

Figura 2.6: Panel HTTP de ssleuth

También posee una página de preferencias donde el usuario puede ver el sistema de puntuación de la extensión e incluso cambiar el valor que se le da a cada una de las partes que lo conforman (figura 2.7).

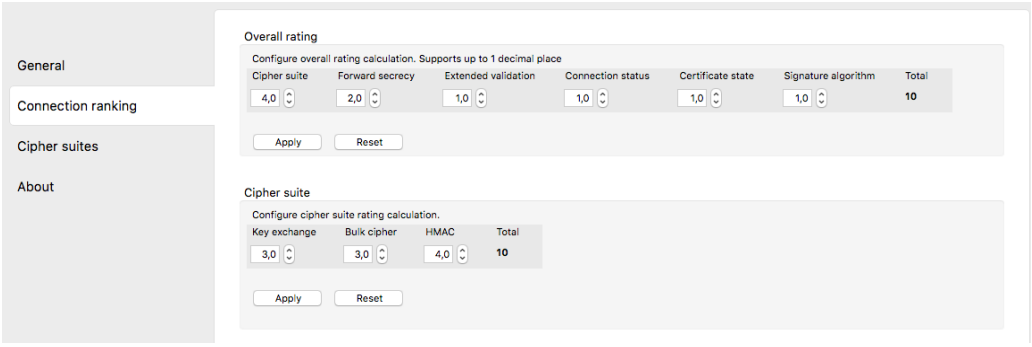

Figura 2.7: Página de preferencias de ssleuth

Estas son las funcionalidades que hacen a *SSLeuth* un buen punto de partida para poder desarrollar una nueva herramienta de análisis, ya que es capaz de acceder a muchos de los parámetros que se han destacado en el apartado 2.1 dentro de una conexión Web segura. Además implementa un sistema de puntuación que permite al usuario cambiar el valor de las ponderaciones que lo componen. Esta funcionalidad permite observar cómo fluctúa la valoración de la conexión cuando se hacen cambios en las ponderaciones.

En la tabla 2.1 se pueden observar cuáles son los parámetros que se desean valorar y que desde un principio *SSLeuth* extrae (color verde). En color naranja se indican otros parámetros necesarios pero que por el momento parece no acceder a ellos la extensión.

Tabla 2.1: Parámetros proporcionados por *SSLeuth*

Propios del Cipher Suite	**Propios de la cadena de certificados**
Versión SSL/TLS	Clave pública del servidor
Algoritmo de intercambio de claves	Algoritmo de firma del servidor
Algoritmo de cifrado simétrico	Validez del certificado del servidor
Algoritmo de Hash	Certificados de la cadena de certificación
	Extensiones de los certificados
	Información de revocación

2.3.2. Estructura de *SSLeuth*

Una vez se ha identificado la información de seguridad que se puede extraer con *SSLeuth*, se localizarán las funciones dentro del código fuente. Además, como se ha indicado en la tabla 2.1 hay algunos parámetros que no proporciona esta extensión, por lo que se deberá comprobar si es o no posible acceder a ellos.

Como se puede observar en la figura 2.8 dentro del directorio de *SSLeuth* se pueden localizar los archivos fundamentales que componen una extensión de Firefox, tal y como se ha descrito en el apartado 2.2.2. Por ello se empezará analizando el contenido de estos, a excepción del archivo *chrome.manifest* que no incluye ninguna variación de lo ya explicado. Además, se puede observar como hay una carpeta llamada *modules* que contiene los scripts que se encargarán de acceder a la información de seguridad

de la conexión. El concepto de módulos se explicará al final de este subapartado, y el funcionamiento de estos scripts se detallará más profundamente en el último apartado de este capítulo.

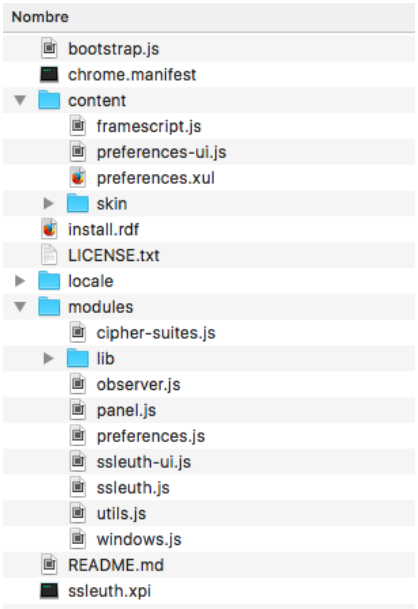

Figura 2.8: Directorio de *SSLeuth*

Fichero install.rdf

En este archivo se encuentran los parámetros básicos que ya se han descrito en el apartado 2.2.2. Pero también contiene otros parámetros, entre los cuales se pueden destacar los siguientes:

- *iconURL*: se utiliza para indicar la URL hacia la imagen que utilizará Firefox como icono de dicha extensión, las dimensiones deben ser 35x35*px*.

- *optionsURL*: contiene la URL del archivo hacia el cual redirigirá Firefox al usuario cuando este acceda a las preferencias de la extensión.

- *optionsType*: valor del 1 al 3 que indica el tipo de UI que se utilizará para mostrar por pantalla las opciones. En este caso es un 3, por lo tanto se abrirá una nueva pestaña del navegador para mostrar la página de preferencias.

- *bootstrap*: consiste en un valor booleano que es *false* por defecto e indica si la extensión es *boot-strappable*. Este tipo especial de extensiones permiten realizar cuatro tipos de acciones de forma mucho más fluida: instalar, desinstalar, arrancar y apagar la extensión [3]. Para ello es necesario crear el archvo *bootstrap.js* donde se defina qué hacer en cada una de las 4 situaciones mencionadas.

Carpeta content

Esta carpeta contiene varios archivos, pero los que se analizarán son *preferences.xul* y *preferences-ui.js*. Ya que el objetivo principal de este análisis no es conecer a la perfección el funcionamiento de la extensión, sino saber reutilizarlo de manera que nos permita incluir las funcionalidades deseadas.

El archivo *preferences.xul* se utiliza como página de opciones de la extensión, específicamente en este caso para cambiar las preferencias de esta. Como se puede observar en la figura 2.9 esta permite navegar entre diferentes opciones a través de la barra lateral izquierda. En el código 2.1 se puede observar como utiliza un simple sistema de "cajas" (*listbox*) para poder conseguirlo.

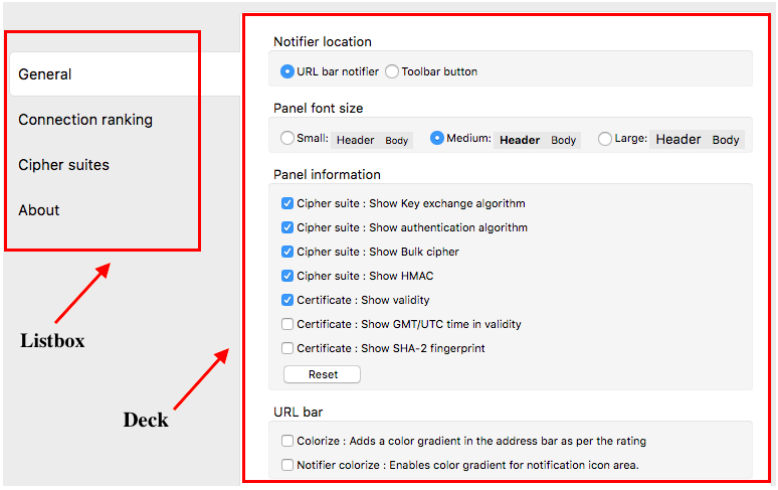

Figura 2.9: Estructura de la UI las preferencias

Código 2.1: Listbox de preferences.xul

```
1  <listbox id="ssleuth-pref-categories"
2      selectedIndex = "0"
3      onselect="document.getElementById('viewport-panel').
           selectedIndex
4                        = this.selectedIndex">
5    <listitem class="ssleuth-pref-category"
6      image=""
7      label="&ssleuth.pref.tab.ssleuthui;"
8      selected="true"/>
9    <listitem class="ssleuth-pref-category"
10     image=""
11     label="&ssleuth.pref.tab.ranking;"/>
12   <listitem class="ssleuth-pref-category"
13     image=""
14     label="&ssleuth.pref.tab.ciphersuites;"/>
15   <listitem class="ssleuth-pref-category"
16     image=""
17     label="&ssleuth.pref.tab.about;"/>
18 </listbox>
```

Como se puede observar en la figura 2.9 la página de preferencias está dividida en dos partes principales, un *listbox* y un *deck*. El *deck* proporciona al usuario diversas funcionalidades, como variar el tipo de letra o cambiar el sistema de puntuación. Pero como este *deck* está compuesto por varias ventanas, se necesita de una herramienta para navegar entre todas ellas, y esta es el *listbox*.

Esta lista se controla a través del archivo *preferences-ui.js*, y como se puede observar en el código 2.1, el *listbox* contiene una propiedad *onselect* que se activa cada vez que se selecciona uno de los *listitems*. Esto provoca que se cambie el contenido del *deck*.

Dentro del *deck* se introducen *scrollboxes* que contienen los parámetros de cada una de las pestañas. Al observar el contenido de estas se puede ver claramente cómo se utilizan *vboxes* y *hboxes* para distribuir los conjuntos de elementos en la UI. Para mostrar un ejemplo se analizará la pestaña *Connection ranking*, más concretamente una de las filas que esta posee.

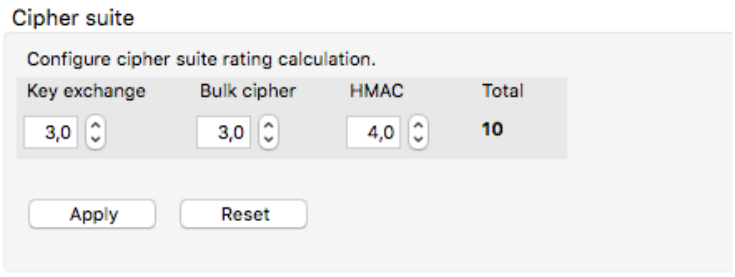

Figura 2.10: UI de configuración del cipher suite

En el archivo *preferences.xul* (código 2.2) se puede observar cómo se utiliza un *grid* (líneas 5 y 37) para introducir los *scrollboxes*. En este *grid* se introduce el número de columnas necesarias, que en este caso son cuatro (líneas 7 a 12). Y las filas que se necesiten que tan solo son dos, una para introducir el nombre que identifica cada columna (líneas 14 a 19) con la ayuda de un *label*. Y la otra para introducir las ponderaciones de cada parámetro del sistema de puntuación, además de utilizarse un *label* para el campo de *Total* (líneas 20 a 35). Además, se utilizan dos botones para poder aplicar los cambios en las ponderaciones o resetearlos a los originales (líneas 42 a 45).

Código 2.2: Groupbox de preferences.xul

```
1  <groupbox>
2    <caption label="&ssleuth.pref.rating.ciphersuite;"/>
3    <description value="&ssleuth.pref.rating.ciphersuite.desc;"/>
4    <vbox align="baseline">
5    <grid equalsize="always" align="baseline"
6        style="background-color:-moz-dialog; height: 2.0em;">
7      <columns equalsize='always'>
8        <column flex='1'/>
9        <column flex='1'/>
10       <column flex='1'/>
11       <column flex='1'/>
12     </columns>
13     <rows>
14       <row>
```

```
15          <label value="&ssleuth.pref.rating.keyexchange;" flex="1
                "/>
16          <label value="&ssleuth.pref.rating.bulkcipher;" flex="1"
                />
17          <label value="&ssleuth.pref.rating.hmac;" flex="1"/>
18          <label value="&ssleuth.pref.rating.total;" flex="1"/>
19        </row>
20        <row align='baseline'>
21          <box flex="1">
22            <textbox width="50" type="number"
23            id="ssleuth-pref-cs-kx-weight" decimalplaces="1"/>
24          </box>
25          <box flex="1">
26            <textbox width="50" type="number"
27            id="ssleuth-pref-cs-cipher-weight" decimalplaces="1"/>
28          </box>
29          <box flex="1">
30            <textbox width="50" type="number"
31            id="ssleuth-pref-cs-hmac-weight" decimalplaces="1"/>
32          </box>
33        <label flex="1" id="ssleuth-pref-cs-rating-total"
34                              style="font-weight:bold;"/>
35        </row>
36      </rows>
37    </grid>
38    <description value=""/>
39    </vbox>
40
41    <hbox>
42      <button label="&ssleuth.pref.ui.button.apply;"
43        id="ssleuth-pref-cs-ratings-apply"/>
44      <button label="&ssleuth.pref.ui.button.reset;"
45        id="ssleuth-pref-cs-ratings-reset"/>
46    </hbox>
47      <description value=""/>
48  </groupbox>
```

La monitorización de los cambios en las ponderaciones se hace a través del archivo *preferences-ui.js*, que utiliza los *ids* de los elementos que contienen estos valores. Además, se deben almacenar dichos valores para poder utilizarlos en el algoritmo de puntuación que tiene implementado la extensión. Para poder explicar el funcionamiento de este proceso se mostrarán las diferentes partes del código *preferences-ui.js* involucradas.

Primero se declaran varios tipos de variables y constantes que se utilizarán dentro del archivo JS (código 2.3). El contenido de cada una de ellas es el siguiente:

- Variables *cxRatingIds* y *csRatingIds* (líneas 1 a 14): son dos arrays de ids de los *textboxes* donde se muestra por pantalla la ponderación de cada parámetro, una para cada conjunto de ponderaciones (conexión y *cipher suite*).

- Constante *prefs* (líneas 18): es una constante donde se almacena la referencia a la interfaz *nsIPrefBranch* de Mozilla, esta será la que se utilizará para poder almacenar las preferencias. Cabe destacar que esta interfaz es un servicio XPCOM y no un objeto, la diferencia fundamental es que no se pueden crear instacias de

los servicios puesto que solo puede haber uno. Para poder acceder a las funciones de un servicio se deben seguir tres pasos [4]:

- Obtener el componente donde se implementa la interfaz *nsIPrefBranch* a través de su correspondiente Uniform Resource Identifier (URI) (*'@mozilla.org/preferences-service;1'*).
- Obtener la referencia a la interfaz *nsIPrefBranch* utilizando la función *getService()*.
- Llamar a la función de la interfaz que se necesite (*prefs.getCharPref()*).

- Constantes *PREF_* (líneas 22 a 29): son las constantes que almacenan el nombre que identificará a las difrentes preferencias almacenadas.
- Variables *cxRating y csRating* (líneas 31 y 32): variables donde se almacenará el valor actual de las ponderaciones. Para hacerlo se utiliza la función *getCharPref(NombreDeLaPreferencia)* de la interfaz *nsIPrefBranch*, que nos permite obtener el estado o valor de la preferencia que indicamos como parámetro. Además, se utiliza la función *JSON.parse* puesto que las ponderaciones están almacenadas en el script *cipher-suites.js* en formato *JSON*, por ello se utiliza esta función para obtener el objeto JS con las ponderaciones dentro de los atributos que este posee.

Código 2.3: Variables del archivo preferences-ui.js

```
1  var    cxRatingIds = [
2      'ssleuth-pref-cipher-suite-weight',
3      'ssleuth-pref-pfs-weight',
4      'ssleuth-pref-ev-weight',
5      'ssleuth-pref-ffstatus-weight',
6      'ssleuth-pref-certstate-weight',
7      'ssleuth-pref-signature-weight'
8  ];
9
10 var    csRatingIds = [
11     'ssleuth-pref-cs-kx-weight',
12     'ssleuth-pref-cs-cipher-weight',
13     'ssleuth-pref-cs-hmac-weight'
14 ];
15
16 const Cc = Components.classes,
17       Ci = Components.interfaces,
18       prefs = Cc['@mozilla.org/preferences-service;1'].
             getService(Ci.nsIPrefBranch),
19       BRANCH = 'extensions.ssleuth.',
20       BRANCHTLS = 'security.ssl3.';
21
22 const PREF_NOTIF_LOC = BRANCH + 'notifier.location',
23       PREF_PANEL_FONT = BRANCH + 'panel.fontsize',
24       PREF_CX_RATING = BRANCH + 'rating.params',
25       PREF_CS_RATING = BRANCH + 'rating.ciphersuite.params',
26       PREF_SUITES_TGL = BRANCH + 'suites.toggle',
27       PREF_PANEL_INFO = BRANCH + 'panel.info',
28       PREF_URL_COLORIZE = BRANCH + 'ui.urlbar.colorize',
29       PREF_NOTIFIER_COLORIZE = BRANCH + 'ui.notifier.colorize';
```

```
30
31  var    cxRating = JSON.parse(prefs.getCharPref(PREF_CX_RATING)),
32         csRating = JSON.parse(prefs.getCharPref(PREF_CS_RATING)),
33         csTglList = JSON.parse(prefs.getCharPref(PREF_SUITES_TGL))
               ,
34         panelInfo = JSON.parse(prefs.getCharPref(PREF_PANEL_INFO))
               ;
```

Una vez se han identificado las variables que se utilizarán a lo largo del archivo *preference-ui.js* se explicará el contenido de la función *prefUI()* (código 2.4), la cual ocupa el resto del fichero. No se explicarán todas y cada una de las funciones que esta contiene, sino las que sean importantes para poder manipular los valores de las ponderaciones. Además, la explicación de dichas funciones se hará sobre los parámetros que se utilizan para valorar el *cipher suite*, puesto que sería lo mismo para los correspondientes a la valoración global de la conexión.

- *initRatings()* (líneas 1 a 11): esta función se llama al instanciar esta clase, y se encarga de introducir dentro de la variable *csRating* los valores actuales de las ponderaciones del *cipher suite*. A continuación, introduce dentro de los *textboxes* los valores correspondientes, y finalmente llama a la función *csRatingChanged* para actualizar el total. El funcionamiento de esta última función se explicará más adelante.

- *addListeners()* (líneas 14 a 28): esta función se encarga de crear los diferentes *listeners* que se utilizarán para detectar que algún valor de ponderación ha cambiado y que los botones *apply* o *reset* han sido pulsados. En el caso de que los valores de ponderación cambien se llamará a la función *csRatingChanged* para actualizar el total, y los botones llamarán a sus correspondientes funciones (*csRatingApply* y *csRatingReset*) que se detallarán más adelante.

- *csRatingChanged()* (líneas 30 a 36): esta función se utiliza para mostrar en la página de preferencias el total de las diferentes ponderaciones. Consiste en una simple suma de los valores almacenados en los *textboxes* de cada parámetro del *cipher suite*, mostrando el resultado en el *label* que se utiliza para el total.

- *csRatingApply()* (líneas 39 a 51): se encarga de introducir dentro de los atributos del objeto *csRating* los diferentes valores de ponderación así como también el total. Después, llama a la función *setCharPref(NombreDeLaPreferencia,NuevoValor)* de la interfaz *nsIPrefBranch* para actualizar el valor almacenado en las preferencias.

- *csRatingReset()* (líneas 53 a 56): es responsable de retornar las ponderaciones a los valores por defecto almacenados en el archivo *cipher-suites.js*. Primero se utiliza la función *prefs.clearUserPref(NombreDeLaPreferencia)* de la interfaz *nsIPrefBranch* para borrar la preferencia guardada. Y finalmente se vuelve a renovar el contenido de la variable *csRating* y el del mostrado por pantalla.

Código 2.4: Funciones de preferences-ui.js

```
var initRatings = function () {
    csRating = JSON.parse(prefs.getCharPref(PREF_CS_RATING));
    for (var [id, val] in Iterator({
        'ssleuth-pref-cs-kx-weight': csRating.keyExchange,
        'ssleuth-pref-cs-cipher-weight': csRating.bulkCipher,
        'ssleuth-pref-cs-hmac-weight': csRating.hmac
    })) {
        document.getElementById(id).value = val;
    }
    csRatingChanged();
};


var addListeners = function () {

    for (i = 0; i < csRatingIds.length; i++) {
        document.getElementById(csRatingIds[i])
        .addEventListener('change', csRatingChanged, false);
    }
    for (var [id, func] in Iterator({
        'ssleuth-pref-cs-ratings-apply': csRatingApply,
        'ssleuth-pref-cs-ratings-reset': csRatingReset,
    })) {
        document.getElementById(id)
            .addEventListener('command', func, false);
    }

};

var csRatingChanged = function () {
    var total = 0;
    for (i = 0; i < csRatingIds.length; i++) {
        total += Number(document.getElementById(csRatingIds[i]).
            value);
    }
    document.getElementById('ssleuth-pref-cs-rating-total').
        value = total;
};


var csRatingApply = function () {
    csRating.keyExchange =
        Number(document.getElementById('
            ssleuth-pref-cs-kx-weight').value);
    csRating.bulkCipher =
        Number(document.getElementById('
            ssleuth-pref-cs-cipher-weight').value);
    csRating.hmac =
        Number(document.getElementById('
            ssleuth-pref-cs-hmac-weight').value);
    csRating.total = csRating.keyExchange +
                     csRating.bulkCipher +
                     csRating.hmac;
    prefs.setCharPref(PREF_CS_RATING,
             JSON.stringify(csRating));
};
```

```
52
53 var csRatingReset = function () {
54     prefs.clearUserPref(PREF_CS_RATING);
55     initRatings();
56 };
```

Además, para que las preferencias funcionen correctamente también se utiliza el fichero *preferences.js* localizado en la carpeta *modules*. Lo único que debe tenerse en cuenta, en este caso, es la variable *defaultPreferences*. Siguiendo con el caso de las ponderaciones del *cipher suite*, se puede observar como al parámetro *'rating.ciphersuite.params'* se le asigna el objeto *ciphersuites.weighting*. Este se encuetra dentro del archivo *cipher-suites.js* (código 2.5) y contiene los valores por defecto de las ponderaciones.

Código 2.5: Ponderaciones del ciphersuite del archivo cipher-suites.js

```
1  const ciphersuites = {
2      .   .   .
3      .   .   .
4      .   .   .
5      weighting: {
6          keyExchange: 3,
7          bulkCipher: 3,
8          hmac: 4,
9          total: 10
10     }
11 };
```

Por último, la carpeta *content* contiene dentro de la carpeta *skin* los ficheros Cascading Style Sheets (CSS) y las imágenes que utiliza la extensión.

Carpeta modules

Una vez se ha explicado el funcionamiento de las preferencias, tan solo resta la carpeta *modules* dentro del directorio de *SSLeuth*. Como se ha mencionado al principio del apartado 2.3.2, esta contiene los archivos JS encargados de proveer las funcionalidades necesarias para que el desplegable muestre correctamente la información de seguridad junto con la puntuación correspondiente.

Puesto que este es el núcleo de funcionamiento de la extensión se realizará un análisis de estos archivos JS en el apartado 2.4. Pero primero se debe explicar cómo se utilizan los módulos. Estos funcionan conjuntamente unos con otros formando el código global JS de la extensión. Cada uno de ellos debe estar compuesto por dos partes fundamentales:

- Array *EXPORTED_SYMBOLS*: donde se almacenan los objetos, métodos o propiedades que utilizarán los demás archivos JS que importen dicho módulo.
- Código: este trata dichos "símbolo" para que aporten la información adecuada.

Por otra parte, para que un script pueda implementar dichos "símbolos" deberá utilizar la función *Components.utils.import("Ruta/Hacia/modulo.js")*. Además, si se necesitase incluir un nuevo módulo en la extensión debería declararse en la función *unloadModules()* del archivo *install.rdf*.

2.4. Estructura del directorio *modules* de *SSLeuth*

Una vez explicado el funcionamiento de los módulos se analizarán los archivos del directorio *modules*. Este análisis se realizará gradualmente para que se pueda entender correctamente la dependencia de unos archivos con otros. Además, en este apartado también se destacarán las partes necesarias para ser utilizadas e integradas en un nuevo algoritmo de valoración de la conexión Web segura. Dejando a un lado las funciones y archivos que no se utilizarán para cumplir el objetivo del trabajo.

2.4.1. Ficheros *cipher-suites.js* y *utils.js*

El primer módulo que se analizará es *cipher-suites.js*, los símbolos que exporta son objetos JSON que contienen información relevante para poder aplicar el algoritmo de valoración de la conexión, además de algunos parámetros adicionales:

- *ciphersuites*: este objeto contiene la información necesaria para poder valorar el *cipher suite* de la conexión. Para ello se indica qué puntuación se le daría a cada parte de este dependiendo de cuál sea el algoritmo. Además, incluye los *strings* que se mostrarán en el desplegable, e incluso las ponderaciones por defecto de cada una de las partes del *cipher suite*.

- *connectionRating*: en este objeto se almacenan las ponderaciones por defecto de la valoración global de la conexión.

- *tlsVersions*: este incluye las diferentes versiones SSL/TLS con un parámetro que indica la valoración que tiene cada una de ellas.

Otro script que no importa ningún otro módulo es *utils.js*, que exporta un símbolo llamado *utils* el cual contiene una función que se utilizará en diferentes archivos JS, *getText(name)* (código 2.6). Esta función es la encargada de devolver el string correspondiente a una variable dependiendo del lenguaje, es decir, esta accede al archivo *locale/panel.properties* que contiene los *strings* que deben mostrarse en el desplegable. Y con la ayuda del parámetro *name* especificado puede localizar dentro de este archivo la cadena de caracteres en el lenguaje adecuado.

Código 2.6: Método getText(name) de utils.js

```
1 var getText = function (name) {
2     try {
3         var bundle = Services.strings
4             .createBundle('chrome://ssleuth/locale/
                  panel.properties');
5         return bundle.GetStringFromName(name);
6     } catch (e) {
7         return name;
8     }
9 };
```

A partir de este momento los archivos importan símbolos de los módulos anteriormente explicados, por ello se incluirán unas figuras para poder observar dichas relaciones de forma más sencilla y esquemática. En la figura 2.11 se puede observar

como el script *preferences.js* utiliza parámetros de *cipher-suites.js*. Como ya se ha mencionado en el apartado 2.3.2, *preferences.js* se utiliza para introducir las preferencias por defecto dentro de un solo objeto JSON y así poder acceder a ellas más fácilmente. Por ello, se importan los símbolos del archivo *cipher-suites.js* que contienen las ponderaciones por defecto del *cipher suite* y de la conexión global.

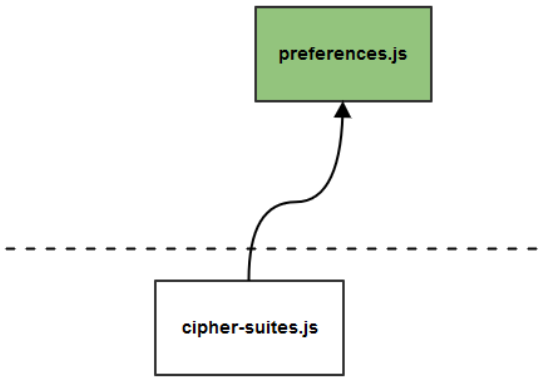

Figura 2.11: Dependencias del directorio *modules* hasta *prefereneces.js*

2.4.2. Fichero *panel.js*

El siguiente módulo es *panel.js*, que se encarga de diseñar el desplegable que se mostrará por pantalla con todos los parámetros de seguridad y sus valoraciones. También importa el módulo *utils.js* (figura 2.12) para poder acceder a los *strings* que debe mostrar. Dentro de este *script* hay varias funciones pero lo importante es saber que el desplegable está compuesto por un panel "padre" que engloba a otros tres (*Primary*, *Domains* y *Cipher suites*), tal y como se puede observar en la figura 2.5. Para navegar entre los diferentes paneles "hijos" se utilizará un sistema de *deck* como en la página *preferences.xul*.

Lo primero que se debe construir son los paneles "hijos", para ello hay tres funciones diferentes, pero la que contiene los datos de seguridad de la página Web es la función *panelMain()*. Este panel se construye con elementos *vbox* y *hbox* que permiten distribuir los datos correctamente. Y además, se utilizan elementos *description* para mostrar el contenido de los distintas partes del panel. Esto se consigue gracias al método *getText(name)* del módulo *utils.js* que se ha explicado en el apartado anterior.

A continuación, como se puede observar en el código 2.7 se declara la variable *panelvbox* que será el panel "padre" de los demás (líneas 1 a 5). Además, se introduce el sistema de puntuación por estrellas (líneas 7 a 11) y numérico (líneas 13 a 16).

Código 2.7: Sistema de puntuación principal de panel.js

```
1 let hb = panelvbox.appendChild(elem('hbox', {
2     id: 'ssleuth-img-cipher-rank-star',
3     align: 'baseline',
4     height: '20'
5 }));
6 //SE INTRODUCEN TODAS LAS ESTRELLAS POR DEFECTO
7 for (var i = 1; i <= 10; i++) {
```

```
8       hb.appendChild(elem('image', {
9           id: 'ssleuth-img-cipher-rank-star-' + i,
10          class: 'ssleuth-star'
11      }));
12  //PUNTUACION EN FORMATO NUMERICO X/10
13  hb.appendChild(elem('description', {
14      id: 'ssleuth-text-cipher-rank-numeric',
15      class: 'ssleuth-text-title-class'
16  }));
```

Lo que sigue es introducir en el panel las pestañas de los diferentes paneles “hijos” (código 2.8), para ello se utilizan elementos del tipo *description* (líneas 1 a 9). Además, se les añade un *listener* a cada pestaña para que se pueda detectar cuál ha sido la última seleccionada (líneas 12 a 17). También se declara un atributo llamado *selected* que marca el estilo de la pestaña. Es decir, la última pestaña seleccionada asignará el valor *true* a este atributo y *false* a las demás, y junto con la ayuda de las hojas de estilo se modificará el aspecto de estas para dar una sensación de navegación entre las pestañas.

Código 2.8: Nueva pestaña en panel "padre"de panel.js

```
1   let chb = hb.appendChild(elem('hbox', {
2       id: 'ssleuth-paneltab-main',
3       _selected: 'true',
4       class: 'ssleuth-paneltab-tab'
5   })); { //NOMBRE QUE SE MOSTRARA EN CADA TAB
6       chb.appendChild(elem('description', {
7           value: utils.getText('tab.primary')
8       }));
9   }
10
11  //NECESARIO PARA DAR LA IMPRESION DE NAVEGACION ENTRE TABS
12  chb.addEventListener('click', function () {
13      doc.getElementById('ssleuth-panel-deck').selectedIndex = 0;
14      doc.getElementById('ssleuth-paneltab-domains').setAttribute(
            '_selected', 'false');
15      doc.getElementById('ssleuth-paneltab-cipher').setAttribute('
            _selected', 'false');
16      doc.getElementById('ssleuth-paneltab-main').setAttribute('
            _selected', 'true');
17  }, false);
```

Finalmente se deben introducir las funciones que crean los paneles “hijos” dentro de un *deck* (*deck.appendChild(panelHijo()*). Esta es la estructura que debe montarse para poder crear el panel completo con las penstañas que se deseeen.

2.4.3. Fichero *ssleuth-ui.js*

Para manipular el contenido de los paneles “hijos” mencionados en el apartado anterior se utilizarán los dos scripts restantes, *ssleuth-ui.js* y *ssleuth.js*. El script *ssleuth-ui.js* es el encargado de introducir dentro del panel los datos de seguridad que le pasa *ssleuth.js*, el cual extrae dichos datos del navegador y los procesa.

Para explicar dicho proceso se empezará viendo el funcionamiento de *ssleuth-ui.js*, el cual importa una gran cantidad de módulos (figura 2.13) con un objetivo distinto en cada uno de ellos:

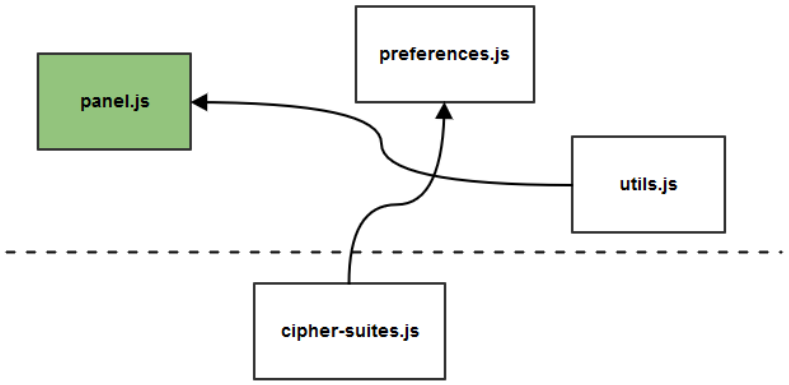

Figura 2.12: Dependencias del directorio *modules* hasta *panel.js*

- *cipher-suites.js*: utilizará el símbolo de *ciphersuites* para poder valorar la fortaleza de este en general y *tlsVersions* para poder mostrar la versión SSL/TLS en el desplegable. Ambos casos se mostrarán más adelante en este apartado.
- *preferences.js*: se puede utilizar para acceder a las preferencias de las distintas ponderaciones, además de ser esencial para otras funcionalidades que proporciona a *ssleuth-ui.js* en *SSLeuth*. Un ejemplo de ellas es el uso de este script para mostrar la información de la pestaña *Domains* que se ha mencionado en el apartado 2.3.1.
- *panel.js*: esta importación le permite controlar el panel globalmente durante el transcurso de la navegación del usuario. Es decir, todos los casos en los que el panel se debe ocultar o mostrar por pantalla serán tratados de forma adecuada.

Una vez se ha explicado el objetivo principal de porqué se han importado dichos módulos, se analizará el contenido del código. De entre todas las funciones que este contiene, la que controla el contenido de la pestaña del panel que nos interesa (*Primary*) es la función *protocolChange()*, que se encuentra dentro de otra función llamada *ui()*, la cual es el único símbolo que exporta este módulo.

La función *protocolChange(proto, data, win, winId)* consiste en un *switch* que dependiendo del protocolo que utilice el navegador accederá a unas funcionalidades o a otras, este protocolo debe indicarse a través del parámetro *proto*. Si el protocolo es desconocido no se mostrará ninguna información en el panel (en blanco), si este es HTTP se mostrará por pantalla un panel indicando que la comunicación no ha sido cifrada (figura 2.6) y en caso de ser HTTPS se llamará a la función *fillPanel(data, win, winId)* (código 2.9). Esta función es la que se encarga de plasmar la información que se ha pasado al módulo *ssleuth-ui.js* a través del parámetro *data*. Los atributos *win* y *winId* se utilizan para poder acceder a los elementos del panel.

Código 2.9: Función fillPanel() de ssleuth-ui.js

```
1 function fillPanel(data, win, winId) {
2     setButtonRank(data.rating, 'https', win);
3     panelConnectionRank(data.rating, win);
4     showCipherDetails(data.cipherSuite, win);
```

```
5       showPFS(data.cipherSuite.pfs, win);
6       showFFState(data.state, win);
7       showCertDetails(data.cert, data.domMismatch, data.ev, win);
8       showTLSVersion(win, winId);
9       showCrossDomainRating(win, winId);
10  }
```

La funcionalidad de cada una de estas funciones que se encuentran dentro de *fillPanel()* es la siguiente:

- *setButtonRank(connectionRank, proto, win)*: se utiliza para mostrar el icono adecuado en el navegador dependiendo de la puntuación global de la conexión que puede ir desde 0 hasta 10. Se establecen diversos rangos de valores que marcarán la imagen que se utilizará, la cual variará de color. Además, se puede observar como también se le indica por parámetro el protocolo HTTPS, aunque este parámetro no se utiliza dentro de la función.

- *panelConnecitonRank(rank, win)*: consiste en una función con dos bucles que muestran en el panel el número de estrellas necesario dependiendo de la valoración (*rank*) del 0 al 10 que se le haya indicado por parámetro. También rellena la parte numérica situada a la derecha de las estrellas con el formato *rank*/*10*.

- *showCipherDetails(data.ciphersuite, win)*: esta función se encarga de añadir al panel los parámetros pertinentes al *cipher suite* de la conexión. Para ello se introducen dentro del objeto *data.ciphersuite* los parámetros adecuados para que estos puedan ser mostrados por el panel.

- *showPFS(data.cipherSuite, win)*: se encarga de rellenar el campo de *Perfect Forward Secrecy* del panel, el cual depende del algoritmo de intercambio de clave del *cipher suite*.

- *showFFState(state, win)*: el navegador indica con un atributo si la conexión es o no segura y se le indica a esta función para que muestre dicha información por el panel. El acceso a este parámetro del navegador, así como también su significado, se explicarán detalladamente más adelante en este apartado.

- *showCertDetails(data.cert, data.domMismatch, data.ev, win)*: cada uno de los parámetros pasados a dicha función contiene información referente al certificado del servidor Web. Estos deberán ser los adecuados para que se puedan mostrar los datos que contiene dicha sección del panel, como se observa en la figura 2.5. La obtención de estos parámetros se explicará más adelante en el análisis del *script ssleuth.js*, pero cabe destacar el uso de una interfaz de Mozilla en esta función, *nsIX509CertValidity*.

  La interfaz *nsIX509CertValidity* permite el acceso a la fecha y hora exacta a partir de la cual el certificado es válido y cuándo expira (código 2.10). Para poder acceder a dicha información se debe obtener el atributo *nsIX509CertValidity* de la interfaz *nsIX509Cert* (explicada a continuación en *ssleuth.js*) y utilizar el método *QueryInterface()* [5] para acceder a las funciones *notBefore* y *notAfter* de dicha interfaz, las cuales proporcionan el periodo de validez del certificado.

Código 2.10: Acceso a la interfaz nsIX509CertValidity en ssleuth-ui.js

```
1  var validity = svCert.validity.QueryInterface(
       Ci.nsIX509CertValidity);
2  var certValidity = doc.getElementById('
       ssleuth-text-cert-validity');
3  var notBefore = new Date(validity.notBefore / 1000),
4      notAfter = new Date(validity.notAfter / 1000);
5
6  if (panelInfo.validityTime) { //Periodo de validez con el
       dia y la hora exactas
7      certValidity.textContent = notBefore.toLocaleDateString
           () + notBefore.toLocaleTimeString() +
8          ' -- ' + notAfter.toLocaleDateString() +
               notAfter.toLocaleTimeString();
9  } else {//Periodo de validez con tan solo los dias
10     certValidity.textContent = notBefore.toLocaleDateString
           () + ' -- ' + notAfter.toLocaleDateString();
11 }
```

- *showTLSVersion(win,winId)*: se encarga de rellenar el parámetro del panel que indica la versión SSL/TLS de la conexión. Aunque no se pase por parámetro dicha versión se accede a esta información a través del módulo *ssleuth.js*, y posteriormente el *script observer.js* se encarga de pasarla a *ssleuth-ui.js*. Puesto que este proceso de comunicación entre módulos no se utilizará más adelante, no se explicará el funcionamiento del módulo *observer.js*.

- *showCrossDomainRating(win, winId)*: esta se encarga de rellenar la información referente al segundo panel "hijo" *Domains*. Este panel recibe la información de seguridad de diferentes nombres de dominios a lo que se realizan conexiones asíncronas. Puesto que en un principio no se utilizará esta funcionalidad no se explicará esta función.

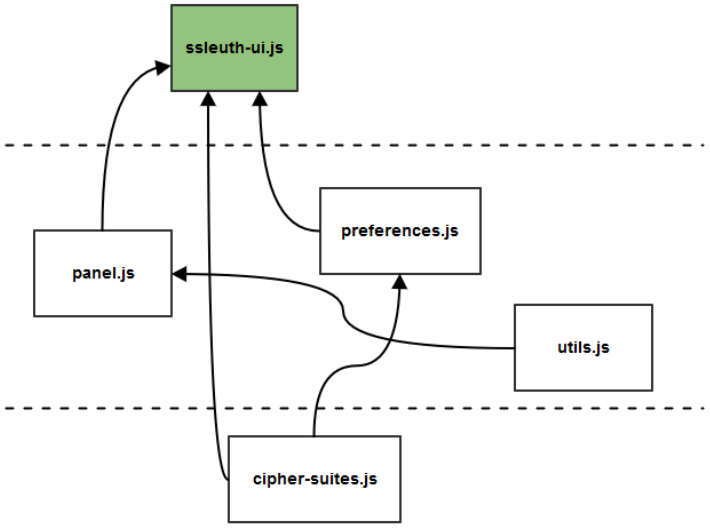

Figura 2.13: Dependencias del directorio *modules* hasta *ssleuth-ui.js*

2.4.4. Fichero *ssleuth.js*

Como se ha podido observar, el módulo *ssleuth-ui.js* necesita la información de seguridad necesaria para poder mostrarla por el panel. Para ello se utiliza el *script ssleuth.js* que accede a dicha información del navegador y la pasa al método *protocolChange()* para que se pueda ver por pantalla. Como ya se ha explicado antes, esta es la función de *ssleuth-ui.js* que comienza el proceso para mostrar los datos de seguridad por el panel.

Tal y como se ha hecho con el resto de *scripts*, se destacarán las funciones que sean relevantes para poder acceder a la información de seguridad de la conexión. Pero antes se mencionará con qué objetivo se han importado los distintos módulos dentro de este script (figura 2.14):

- *cipher-suites.js*: se utiliza para poder acceder a las diferentes variables que este módulo contiene, puesto que se utilizan para poder diferenciar las distintas partes del *cipher suite* que se utiliza en la conexión. Además, contiene la información necesaria para valorar cada uno de los algoritmos que se utilizan.

- *utils.js*: se puede utilizar para acceder a funciones de *log* que proporciona dicho módulo. Pero, al no hacerse uso de este módulo a lo largo del código, no se ha analizado su funcionamiento.

- *preferences.js*: se hace uso de este módulo para poder acceder a las preferencias, es decir, en el momento de aplicar el algoritmo de puntuación se accede a los parámetros de ponderación de cada una de las partes a través de las funciones de este módulo.

- *ssleuth-ui.js*: utiliza este módulo para controlar el panel en cada uno de los eventos por los que puede pasar el navegador, como son cambiar de página, renovarla, cambiar de pestaña, etc. No se profundizará en el funcionamiento de la monitorización de estos cambios, pero este módulo se utiliza también para poder pasarle los datos necesarios a través de la función *onProtocolChange()* para que este script pueda rellenar el panel con la información de seguridad apropiada.

Una vez se ha explicado la funcionalidad de cada uno de los módulos importados dentro de este script se analizarán cuáles son las funciones que nos aportan la información que necesitamos. Entre todas ellas se puede destacar una a partir de la cual se desarrollan las demás, *protocol()*. Dentro de esta función se tratan tres casos distintos, cuando el protocolo utilizado en la conexión Web es desconocido, HTTP o HTTPS.

En el primer caso se llama a la función *protocolChange()* del módulo *ssleuth-ui.js* para que simplemente no muestre información por el panel. En el caso de HTTP se pasará por parámetro la Uniform Resource Locator (URL) en formato HTTPS al módulo *ssleuth-ui.js* y este se encargará de mostrar por pantalla un mensaje indicando al usuario que intente acceder a esta URL (figura 2.6). Finalmente, en caso de ser una conexión a través del protocolo HTTPS el primer fragmento de código que se ejecutará será el siguiente:

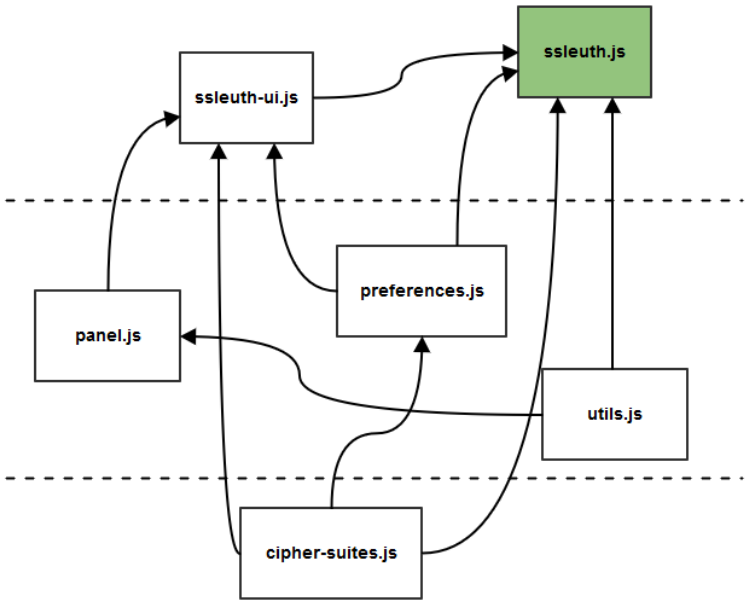

Figura 2.14: Dependencias del directorio *modules* hasta *ssleuth.js*

Código 2.11: Acceso a la interfaz *nsISSLStatus* en *ssleuth.js*

```
1  var secUI = win.gBrowser.securityUI;
2  //Por si la conexion no es segura o hay algon error imprevisto
3  if (!secUI) return;
4
5  var sslStatus = secUI.SSLStatus;
6  if (!sslStatus) {
7      secUI.QueryInterface(Ci.nsISSLStatusProvider);
8      if (secUI.SSLStatus) {
9          //sslStatus = win.gBrowser.securityUI
10         //              .QueryInterface(Ci.nsISSLStatusProvider)
11         //              .SSLStatus
12         sslStatus = secUI.SSLStatus;
13     } else {
14         //Se esconde el panel para evitar problemas con el
               funcionamiento de firefox
15         if (urlChanged) {
16             ui.protocolChange('unknown', '', win, winId);
17         }
18         return;
19     }
20 }
```

En estas primeras líneas se obtendrá una variable que pueda acceder a la interfaz *nsISSLStatus* que contiene la información de seguridad de la conexión. Primero se debe acceder al atributo *securityUI* del elemento *gBrowser* del navegador (línea 1), este hace referencia a la ventana de Firefox que se tiene abierta. Esta propiedad a la que accedemos tiene que retornar un valor del tipo *nsISecureBrowserUI* [6]. Para verificar que no ha habido un error al consultar dicho valor se comprueba que la variable no este vacía (línea 3). Por otra parte, la variable *sslStatus* debería estar vacía todavía (línea

6). Puesto que se debe utilizar el método *QueryInterface()* para que la variable *secUI* pueda acceder al atributo *SSLStatus* [7].

El atributo *SSLStatus* es del tipo *nsISSLStatus* y más adelante se explicará como se puede acceder al *cipher suite* de la conexión y al certificado del servidor Web con la ayuda de este. Aunque, antes de acceder a dicha información la función contiene el siguiente código:

Código 2.12: Comprobar estado de la conexión y *EV* en *ssleuth.js*

```
1  // Estados de Firefox que indican el tipo de conexion
2  if ((state & Ci.nsIWebProgressListener.STATE_IS_SECURE)) {
3      securityState = 'Secure';
4  } else if ((state & Ci.nsIWebProgressListener.STATE_IS_INSECURE)
       ) {
5      securityState = 'Insecure';
6  } else if ((state & Ci.nsIWebProgressListener.STATE_IS_BROKEN))
       {
7      securityState = 'Broken';
8  }
9  // Comprobamos si el certivicado es Extended Validated
10 if (state & Ci.nsIWebProgressListener.STATE_IDENTITY_EV_TOPLEVEL
       ) {
11     extendedValidation = true;
12 }
```

En el código 2.12 se comprueba el valor que contiene el parámetro *state*. Este es un valor que se obtiene de la variable *aState*, a través del método *onSecurityChange()* de la interfaz *nsIWebProgressListener* [8]. Gracias a él se pueden comprobar algunos aspectos de la conexión que se comentarán más adelante. Este valor es un *int* que contiene la información en los dígitos hexadecimales que lo componen, a los cuales se les llamará indicadores. Es decir, que los valores hexadecimales que conformar el número *state* son los que contienen la información (figura 2.15). Para poder comprobar coincidencias en los dígitos hexadecimales entre dos números se utiliza el operador &.

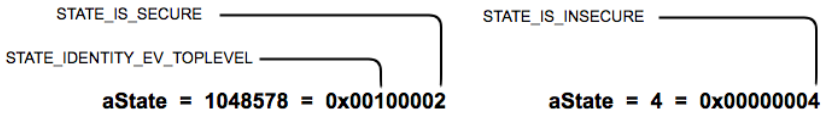

Figura 2.15: Funcionamiento del atributo *aState*

En este fragmento de código, primero se comprueba el estado del canal de comunicaciones. Este puede tomar tres valores distintos que son: *STATE_IS_INSECURE*, *STATE_IS_BROKEN* y *STATE_IS_SECURE*. Por lo tanto, se comprueba que el valor coincida con el tercer caso para asegurarse de que el canal es seguro (líneas 2 a 8). Para realizar dicha comparación se accede a la interfaz *nsIWebProgressListener* y se consultan las constantes de esta que contienen los valores de cada estado.

En la siguiente condición (líneas 10 a 12) se comprueba si el certificado del servidor es *Extended Validated*, para ello también se compara la variable *state* con la constante *STATE_IDENTITY_EV_TOPLEVEL* de la interfaz *nsIWebProgressListener*. Ya que este es uno de los parámetros del certificado del servidor que se ha destacado en el apartado 2.1, para poder valorar la seguridad de la conexión se comprobará que este se extraiga

correctamente. Para ello se seguirá el recorrido de este valor a lo largo del código hasta que se muestra por pantalla en el panel.

Código 2.13: Recorrido de la variable extendedValidation entre ssleuth.js y ssleuth-ui.js

```
1  //Funcion protocol, ssleuth.js
2  ui.protocolChange('https', {
3    rating: rating,
4    cipherSuite: cipherSuite,
5    state: securityState,
6    cert: cert,
7    domMismatch: sslStatus.isDomainMismatch,
8    ev: extendedValidation
9  },
10 win, winId);
11 //Funcion fillPanel(), ssleuth-ui.js
12 showCertDetails(data.cert, data.domMismatch, data.ev, win);
13 //Funcion showCertDetails, ssleuth-ui.js
14 if (ev) {
15   elemEV.textContent = utils.getText('general.yes');
16   elemEV.setAttribute('ev', 'Yes');
17 } else {
18   elemEV.textContent = utils.getText('general.no');
19   elemEV.setAttribute('ev', 'No');
20 }
```

Como se puede observar en el código 2.13, la variable *extendedValidation* se pasa directamente a la función *protocolChange* (líneas 2 a 9). Y dentro de *ssleuth-ui.js* se comprueba esta información directamente para mostrarla por pantalla (líneas 12 a 20), es decir, no se debe realizar ninguna otra comprobación intermedia para saber si el certificado es *Extended Validated*. Para corrobar el correcto funcionamiento de este parámetro se ha accedido a una página Web en la que se ha comprobado que el certificado posee dicha extensión (figura 2.16) y se ha observado como el panel responde correctamente (figura 2.17).

Acceso al *cipher suite* de la conexión

Si se sigue con el contenido de la función *protocol()* se puede observar cómo se llama al método *setTLSVersion(win, winId)*. Este se encargará de comprobar la versión SSL/TLS de la conexión, almacenará el resultado con la ayuda del módulo *observer.js* y se accederá a esta a través del método *showTLSVersion()* del *script ssleuth-ui.js*. Para poder comprobar cuál es la versión SSL/TLS utilizada se realiza el proceso que se observa en el código 2.14.

Código 2.14: Comprobación de la versión SSL/TLS en ssleuth.js

```
1  var index = '';
2  var versionStrings = ['sslv3', 'tlsv1_0', 'tlsv1_1', 'tlsv1_2',
       'tlsv1_3'];
3
4  var secUI = win.gBrowser.securityUI;
5  if (secUI) {
6      var sslStatus = secUI.SSLStatus;
7      if (sslStatus)
8          index = versionStrings[sslStatus.protocolVersion & 0xFF
               ];
```

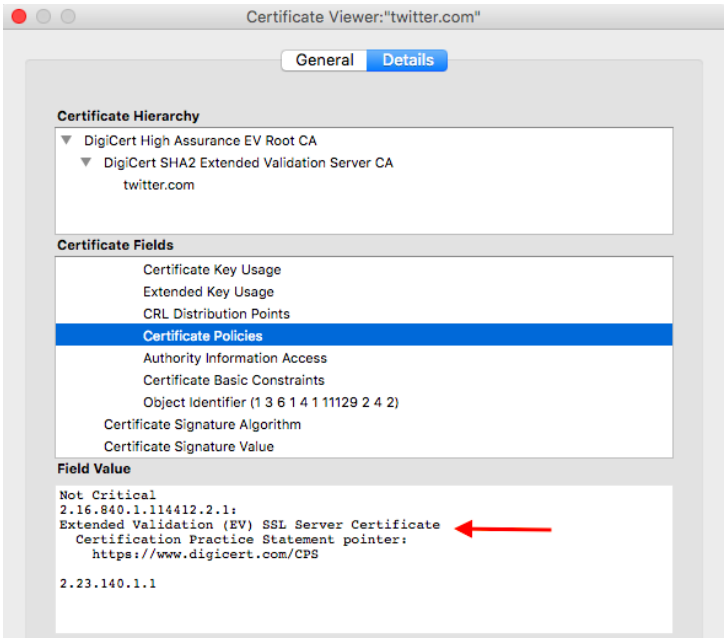

Figura 2.16: Extensión EV del certificado de *.twitter.com

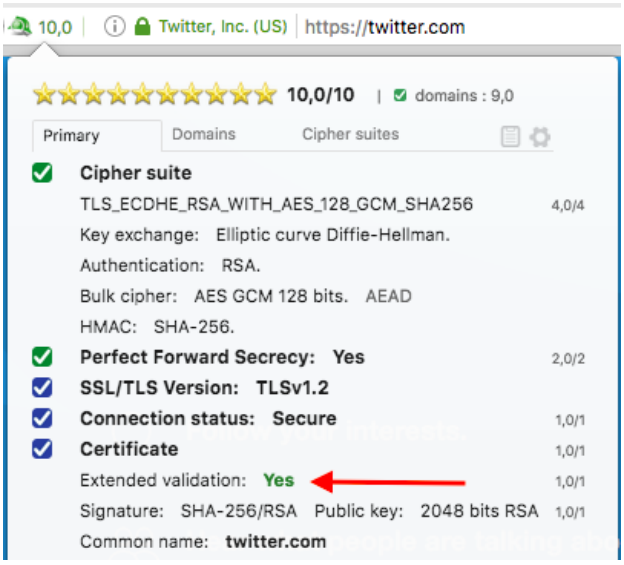

Figura 2.17: Panel mostrando EV del certificado de *.twitter.com

```
9 }
```

Se accede al atributo *protocolversion* de la interfaz *nsISSLStatus* [9] que nos proporciona un número entero del 0 al 3, desde SSL3.0 hasta TLS1.2 (línea 8). Se puede observar como el array de *strings* con el que se extrae el nombre de la versión contiene también TLSv1.3 (línea 2), pero esta aún no está implementada en la interfaz *nsISSLStatus*. Para corroborar el buen funcionamiento de este parámetro se puede observar como

en las figuras 2.5 y 2.17 se han podido detectar correctamente las versiones SSL/TLS.

Para obtener el *cipher suite* de la conexión se accede al atributo *cipherName* de la interfaz *nsISSLStatus* que contiene dicha información. Pero para poder valorar cada una de las partes se debe poder distinguirlas, para ello se utiliza la función *getCsParam(param)*, que aprovecha la estructura de los objetos almacenados dentro del archivo *cipher-suites.js* para asignarles propiedades a los distintos algoritmos y acceder a ellas.

Código 2.15: Funcionamiento funcion getCsParam en sslueth.js

```
1  //Funcion getCsParam en sslueth.js
2  function getCsParam(param) {
3      for (var i = 0; i < param.length; i++) {
4          if ((cipherName.indexOf(param[i].name) != -1)) {
5              return param[i];
6          }
7      }
8              return null;
9  }
10 //Array keyExchange en cipher-suites.js
11 keyExchange: [{
12             name: '_ECDHE_',
13             rank: 10,
14             pfs: 1,
15             ui: 'Elliptic curve Diffie-Hellman',
16             notes: ''
17         },
```

Como se puede observar en el código 2.15, se muestra el primer elemento del array *keyExchange* del archivo *cipher-suites.js* (líneas 11 a 17). Este muestra las propiedades de uno de los algoritmos de intercambio de claves. Entre ellas se puede encontrar el *rank* que es la valoración que se le da con un número del 0 al 10, el parámetro *pfs* que hace referencia a la propiedad *Perfect Forward Secrecy* que aportan algunos algoritmos de intercambio de clave indicando con un 1 si la posee, o un 0 en caso contrario. Pero hay dos atributos que poseen todas los arrays que contiene el objeto ciphersuites de este archivo, *name* y *ui*. El primero se utiliza en la función *getCsParam()* para localizar el algoritmo correcto dentro del *cipherName* (líneas 2 a 9). Para ello se utiliza una función de comparación llamada *indexOf* que permite comprobar si un *string* está contenido dentro de otro. Es decir, se comprueba si el atributo *name* está contenido en el *cipherName*, en caso que esto sea cierto se habrá localizado el algoritmo correspondiente. Por otra parte, el atributo *ui* será el nombre que se mostrará en el panel para dicho algoritmo.

Además, para asegurarse de poder cubrir todos los algoritmos de cifrado simétrico se utiliza otra vez la variable *state* para acceder a la valoración que le da el navegador de Firefox a la conexión, la cual puede tomar tres valores distintos: *STATE_SECURE_HIGH*, *STATE_SECURE_MED* o *STATE_SECURE_LOW*. Como se puede observar en el código 2.16 se le asignará una puntuación distinta al cifrado simétrico dependiendo de qué grado de seguridad nos indique el navegador.

Código 2.16: Banderas de seguridad en ssleuth.js

```
1  if (cipherSuite.bulkCipher.name === '') {
2    if (state & Ci.nsIWebProgressListener.STATE_SECURE_HIGH) {
3      cipherSuite.bulkCipher.rank = cs.strength.MAX;
```

```
4     } else if (state & Ci.nsIWebProgressListener.STATE_SECURE_MED)
          {
5       cipherSuite.bulkCipher.rank = cs.strength.HIGH - 1;
6       } else if (state &
            Ci.nsIWebProgressListener.STATE_SECURE_LOW) {
7           cipherSuite.bulkCipher.rank = cs.strength.MED - 1;
8       }
9   }
```

Acceso al certificado del servidor Web

Una vez se ha analizado el mecanismo para extraer los distintos algoritmos que componen el *cipher suite* resta explicar cómo acceder al certificado del servidor. Este se puede obtener a través del atributo *serverCert* de la interfaz *nsISSLStatus*, el cual es del tipo *nsIX509Cert* y permite acceder a la información del certificado del servidor [10]. Con esta variable primero se comprobará si la longitud de la clave pública es suficiente, dependiendo del algoritmo que se utiliza. El algoritmo de clave pública está almacenado en la variable *cipherSuite* y juntamente con el certificado del servidor se utiliza la función *getKeySize()* para poder averiguar la longitud de la clave asimétrica (código 2.17).

Código 2.17: Función getKeySize() en ssleuth.js

```
1  function getKeySize(cert, alg) {
2      var keySize = '';
3      try {
4          //Estructura ASN1 del certificado
5          var certASN1 = Cc['@mozilla.org/security/nsASN1Tree;1']
6              .createInstance(Ci.nsIASN1Tree);
7          certASN1.loadASN1Structure(cert.ASN1Structure);
8
9          switch (alg) {
10         case 'RSA':
11             keySize = certASN1.getDisplayData(12)
12                 .split('\n')[0]
13                 .match(/\d+/g)[0];
14             break;
15         case 'ECC':
16             keySize = certASN1.getDisplayData(14)
17                 .split('\n')[0]
18                 .match(/\d+/g)[0];
19             break;
20         }
21     } catch (e) {
22         log.error('Error getKeySize() : ' + e.message);
23     }
24     return keySize;
25 }
```

La función *getKeySize()* es necesaria puesto que la interfaz *nsISSLStatus* no permite acceder a la longitud de la clave pública directamente, pero esta contiene un atributo que proporciona la estructura ASN1 del certificado, *ASN1Structure*. Para acceder a ella se debe crear una referencia a la interfaz *nsIASN1Tree* con su correspondiente componente y la función *createInstance()* (líneas 5 y 6). Una vez tenemos dicha referencia podemos

utilizar la función *loadASN1Srtucture(nsIASN1Object)* [11] para cargarle la estructura ASN1 que se desea, que en este caso es la del certificado X509 del servidor Web (línea 7). A continuación, se accede a la línea del certificado que contiene la longitud de la clave pública con la función *getDisplayData()* de *nsIASN1Tree* (líneas 9 a 20), la cual depende del algoritmo de autenticación. En caso de que este sea RSA, la longitud de la clave pública se encuentra en la posición 12 de la estructura ASN1, y cuando se trata de una clave pública de curva elíptica en la 14. Para comprobarlo se ha accedido a la información de Firefox de dos certificados, cada uno con una clave de autenticación de cada tipo (figura 2.18).

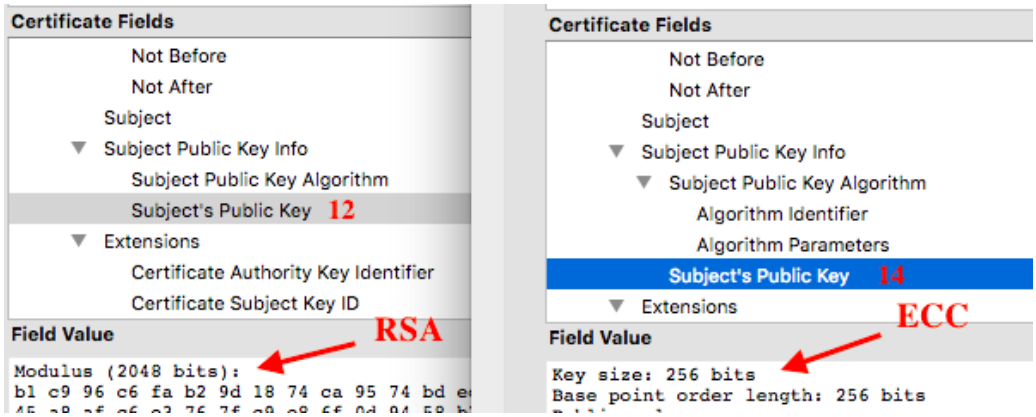

Figura 2.18: Longitud de la clave pública dentro del certificado

Para poder obtener el algoritmo de firma del certificado tampoco se dispone de ningún atributo de la intefaz *nsIX509Cert* que lo proporcione directamente. Por lo tanto se utiliza la función *getSignature(cert)* para acceder a la estructura ASN1 del certificado de nuevo, función que retorna una variable con tres atributos: el algoritmo de cifrado, el agoritmo de hash y la puntuación que se le ha dado al conjunto de ambos.

Código 2.18: Función getSignatureAlg(cert) en ssleuth.js

```
1  function getSignatureAlg(cert) {
2      try {
3          var certASN1 = Cc['@mozilla.org/security/nsASN1Tree;1']
4              .createInstance(Ci.nsIASN1Tree);
5          certASN1.loadASN1Structure(cert.ASN1Structure);
6          var sigText = certASN1.getDisplayData(4).replace(/PKCS
               #1/g, ''),
7              signature = {
8                  hmac: '',
9                  enc: '',
10                 rating: 0
11             };
12         //A veces tan solo aparece el OID identificativo del
               algoritmo de firma
13         if (sigText.indexOf('1 2 840 10045') != -1) {
14             if (sigText.indexOf('1 2 840 10045 4 1') != -1) {
15                 sigText = 'ECDSA with SHA-1';
16             } else if (sigText.indexOf('1 2 840 10045 4 3 1') !=
                    -1) {
17                 sigText = 'ECDSA with SHA-224';
```

```
18              } else if (sigText.indexOf('1 2 840 10045 4 3 2') !=
                    -1) {
19                  sigText = 'ECDSA with SHA-256';
20              } else if (sigText.indexOf('1 2 840 10045 4 3 3') !=
                    -1) {
21                  sigText = 'ECDSA with SHA-384';
22              } else if (sigText.indexOf('1 2 840 10045 4 3 4') !=
                    -1) {
23                  sigText = 'ECDSA with SHA-512';
24              }
25          }
26          //Objeto ciphersuites de cipher-suites.js para acceder a
                los algoritmos de hash y de encriptacion asimetrica
27          const cs = ciphersuites;
28          //Se identifica el algoritmo de hash
29          for (var i = 0; i < cs.HMAC.length; i++) {
30              if ((sigText.indexOf(cs.HMAC[i].ui) != -1) || ((
                    cs.HMAC[i].sigui) &&
31                      (sigText.indexOf(cs.HMAC[i].sigui) != -1)))
                            {
32                  signature.hmac += cs.HMAC[i].ui;
33                  signature.rating += cs.HMAC[i].rank;
34                  break;
35              }
36          }
37          //Se identifica el algoritmo de cifrado asimetrico
38          for (var i = 0; i < cs.authentication.length; i++) {
39              if ((sigText.indexOf(cs.authentication[i].ui) != -1)
                    ) {
40                  signature.enc += cs.authentication[i].ui;
41                  signature.rating += cs.authentication[i].rank;
42                  signature.rating /= 2;
43                  break;
44              }
45          }
46          return signature;
47      } catch (e) {
48          log.error('Error getSignatureAlg() : ' + e.message);
49      }
50  }
```

En esta función (código 2.18) se puede observar como primero se extrae el algoritmo de firma de la estructura ASN1 del certificado del servidor (líneas 3 a 6), y se inicializa la variable con los tres atributos que retornará dicha función (líneas 7 a 11). A continuación, se trata el caso en el que el campo del algoritmo de firma contiene el Object IDentifier (OID) de este en lugar de la descripción (líneas 13 a 25), esto puede ocurrir con algunos algortimos de ECDSA. Y para finalizar se extraen tanto el algoritmo de hash (líneas 29 a 36) como el algoritmo de cifrado (líneas 38 a 45) que se utilizan para firmar el certificado, introduciendo la puntuación de cada uno de estos en el atributo correspondiente y diviendo el resultado de la suma de ambas puntuaciones entre dos.

Una vez se han obtenido todos los parámetros necesarios para aplicar el algoritmo de puntuación de la extensión se accede a las ponderaciones a través del módulo *preferences.js* y se obtiene la valoracion final. Esta se envía junto a otros parámetros necesarios a través de la función *protocolChange()* (código 2.13, líneas 2 a 9) al módulo

ssleuth-ui.js para que se pueda mostrar toda la información por el panel. Uno de estos parámetros que también se envían es *domMismatch*. Dentro de él se almacena el atributo *isDomainMismatch* de la interfaz *nsISSLStatus*, el cual indica si el nombre de dominio de la página Web es el correcto, de acuerdo a lo que indica su certificado X.509.

CAPÍTULO 3

DESARROLLO DE UNA EXTENSIÓN DE ANÁLISIS DE LA SEGURIDAD WEB

En este apartado se llevará a cabo el desarrollo de la nueva extensión, y para ello se seguirán una serie de pasos. Primero es necesario ver qué parámetros de los expuestos en la tabla 2.1 son accesibles a través de una extensión de Firefox, puesto que hay algunos a los que *SSLeuth* no accede. A continuación se integrarán estos datos en un algoritmo de valoración de la seguridad de la conexión, el cual se integrará en la nueva extensión. Finalmente, tras implementar las funcionalidades necesarias en la nueva extensión, se realizarán una serie de pruebas. El objetivo de estas es comprobar el correcto funcionamiento de la extensión, así como también analizar las diferencias con *SSLeuth*.

3.1. Parámetros de seguridad accesibles

En base al análisis realizado en el capítulo 2 sobre el funcionamiento de *SSLeuth*, se ha averiguado como acceder a cierta información de seguridad. Para acceder a estos parámetros se deben utilizar unas determinadas interfaces de Firefox que son las siguientes:

- *nsiWebProgressListener:* esta interfaz contiene un método llamado *onSecurityChange()* que proporciona tres parámetros diferentes, entre los cuales se encuentra uno llamado *aState*. El funcionamiento de esta variable se ha explicado en el apartado 2.4.4, y consiste en un valor númerico que permite comprobar distintos tipos de información a través de indicadores. De entre todos estos indicadores sería apropiado analizar el contenido de tres de ellos:

  - *Security state flags*: indican si el canal de comunicaciones se encuentra o no cifrado.

    - *Security strength flags*: proporcionan tres grados de seguridad de la conexión en caso de que esta se encuentre cifrada.
    - *State identity flags*: indica si el certificado del servidor es del tipo *Extended Validated*.

- *nsISSLStatus*: los atributos de esta interfaz permiten acceder a distintos parámetros de la conexión Web:

    - *serverCert*: esta variable nos devuelve un objeto de la interfaz *nsIX509Cert* referente al certificado del servidor.
    - *cipherName*: contiene el *cipher suite* de la conexión.
    - *keyLength* y *secretKeyLength*: longitudes de las claves pública y privada del servidor respectivamente, la unidad utilizada son bits.
    - *isDomainMismatch*: es un valor booleano que indica si el dominio de la página Web coincide con uno de los que indica el certificado. Este parámetro evitará que se deba consultar la extensión *Certificate Subject Alt Name* para comprobar que el nombre de dominio de la página Web es el correcto.

- *nsIX509Cert*: esta interfaz nos permite acceder a la información de un certificado X.509:

    - *validity*: este atributo nos retorna un objeto de la interfaz *nsIX509CertValidity* que permite acceder a la fecha a partir de las cual el certificado es válido (*notBefore*) y cuándo este cadacurá (*notAfter*).
    - *ASN1Structure*: es un atributo del tipo *nsIASN1Object* el cual podemos cargar en otra variable del tipo *nsIASN1Tree* para poder acceder al contenido de la estructura ASN1 del certificado.
    - *commonName*: contiene el *common name* del propietario del certificado para el cual este ha sido expedido.
    - *organization*, *organizationUnit*, *issuerOrganization*, *issuerOrganizationUnit*: son otros parámetros que contienen información del certificado sobre quién lo ha firmado y para quién lo ha hecho.

Ya se ha comprobado que todos estos parámetros son accesibles a través de una extensión de Firefox gracias a *SSLeuth*. Pero, si se observa el contenido de la tabla 2.1 aún es necesario acceder a otros parámetros, como son: los demás certificados de la cadena, la información de revocación y las extensiones de los certificados que contienen información crítica sobre estos. Todos ellos necesarios para poder valorar correctamente la seguridad de una conexión Web.

Cadena de certificados

Para poder acceder a la cadena de certificación se puede utilizar el atributo *issuer* de la interfaz *nsIX509Cert*. Para explicar el método que se utilizará para consultar toda la cadena se supondrá un ejemplo en el que se tiene una cadena de certificación compuesta por dos certificados, A y B. Donde el certificado B es el que pertenece a

la CA y el certificado A es el del servidor Web. Por lo tanto, al tener acceso al objeto *nsIX509Cert* del certificado A es posible utilizar el atributo *issuer* de esta interfaz para acceder al certificado B. El atributo *issuer* devolverá el objeto *nsIX509Cert* del certificado B, es decir, el perteneciente a la *CA*.

Utilizando el atributo *issuer* en bucle se podrá acceder a toda la cadena de certificación de cualquier servidor Web. Aunque es necesario saber en qué momento se deberá parar de consultar el atributo *issuer*, y esto se hará cuando se llegue a la raíz de la cadena. Para conseguirlo se debe recordar que el certificado de la raíz estará autofirmado (si la cadena está completa), por lo tanto se podrá comprobar este dato con el atributo *isSelfSigned*. Este devuelve un booleano que indica si el certificado está autofirmado o no.

Información de revocación

Por otra parte, la información de revocación del certificado no es accesible a través de ninguna de las interfaces antes mencionadas, puesto que la interfaz *nsIX509Cert* eliminó las funciones que permitían forzar una petición OCSP [12]. Existe otra interfaz llamada *nsIX509CertDB* que posee un método llamado *asyncVerifyCertAtTime()* que permite comprobar de forma asíncrona los usos de la clave de un certificado con su correspondiente servidor OCSP [13]. Sin embargo, no se ha conseguido realizar dicha consulta con éxito y por lo tanto no se ha podido analizar la información de revocación de los certificados.

Otra opción que también se ha planteado para obtener la información de revocación ha sido construir una aplicación externa. Esta realizaría una consulta *OCSP* solicitando la información de revocación del certificado, además, analizaría la información recibida y la transmitiría al navegador utilizando lenguaje nativo. Pero el objetivo de esta extensión es el no tener que compartir información entre dos módulos distintos (extensión y aplicación externa) para poder analizar la seguridad de la conexión.

Aunque no sea posible acceder a la información de revocación, se ha comprobado si el navegador de Firefox verifica el estado del certificado del servidor Web a través de un petición OCSP. Además, si el navegador realiza dicha consulta se observará el comportamiento de este cuando el certificado está revocado. Este análisis de funcionamiento del navegador no pretende justificar la no comprobación de la información de revocación. Pero, se puede forzar al navegador a que no acceda nunca a un servidor Web con un certificado revocado realizando un cambio en su configuración.

Para analizar el funcionamiento del navegador, se ha consultado la configuración por defecto de este (*about:config*), y se han buscado los parámetros que hacen referencia a OCSP, de entre las cuales se han resaltado dos:

- *security.OCSP.enabled*: número entre 0 y 2 que determina el comportamiento de OCSP en el navegador. En la configuración por defecto es 1, y esto indica que el navegador utilizará OCSP para validar los certificados en los que se indique la URL del servidor OCSP [14].

- *security.OCSP.require*: valor booleano que marca el comportamiento del navegador al ejecutar el protocolo OCSP. Si el valor de este parámetro es *false*, el navegador dará por válido el certificado si no se recibe una respuesta por parte

del servidor OCSP. En cambio, si es *true* se esperará a la respuesta del servidor OCSP siempre [15].

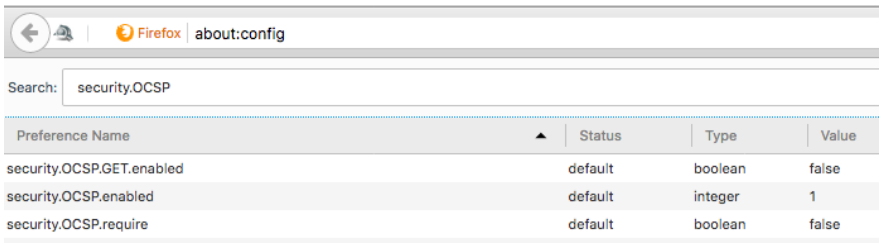

Figura 3.1: Editor de configuración de Firefox

En la figura 3.1 se puede observar como el valor por defecto del parámetro *security.OCSP.require* es *false*. Esto podría provocar que el navegador diera por válido un certificado, si el servidor *OCSP* no responde. Por ello, para evitar este comportamiento del navegador se deberá cambiar el valor de este parámetro a *true*.

Además de analizar la configuración del navegador referente a OCSP, se ha comprobado que esta se aplique correctamente. Para ello, se ha establecido una conexión *HTTPS* y se ha analizado el tráfico intercambiado con el servidor con ayuda del programa *Wireshark* (figura 3.2).

Se ha accedido a la información del certificado en cuestión y se ha comprobado que el número de serie del certificado al cual pertenece la respuesta OCSP sea el de la página Web a la que se ha realizado la conexión. Además, se ha extraído la URL donde se encuentra el *OCSP Responder* del certificado del servidor y se ha realizado una petición DNS inversa de esta para ver si coincide con la IP capturada en Wireshark.

Después de realizar estas comprobaciones se ha observado como el navegador conserva dicha información de revocación para nuevas conexiones con el mismo certificado hasta que estas caduquen, o hasta que se cierre completamente el navegador de Firefox. Además, se ha accedido a una página Web con un certificado ya revocado para ver cómo reacciona el navegador (figura 3.3). Como se puede observar no permite el acceso a esta página, indicando por pantalla el error *SEC_ERROR_REVOKED_CERTIFICATE*.

Por lo tanto, si se realiza el cambio mencionado en el parámetro *security.OCSP.require*, el navegador no accederá a una página Web con un certificado revocado. Si el certificado está revocado se bloqueará el acceso a la página Web (figura 3.3). Por otra parte, si no se recibe una respuesta del servidor OCSP, el navegador se quedará esperando y no accederá tampoco a la página Web.

Extensiones de los certificados

Por último están las extensiones restantes que se deben comprobar en los certificados. Una de ellas se puede obtener mediante el atributo *keyUsages*, la cual es *CertificateKeyUsage*. Esta variable nos proporciona un string con el contenido de dicho campo de las extensiones y así no se deberá localizar dicha información manualmente. Por otra parte, para comprobar la extensión *Certificate Policies* se deberá recorrer la estructura ASN1 del certificado X.509 con la ayuda de la interfaz *nsIASN1Tree*.

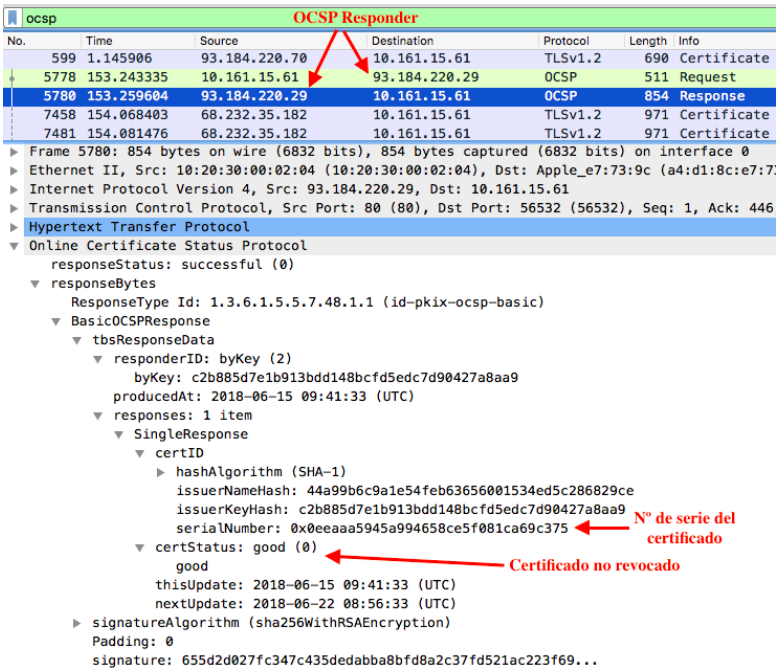

Figura 3.2: Captura de Wireshark petición OCSP

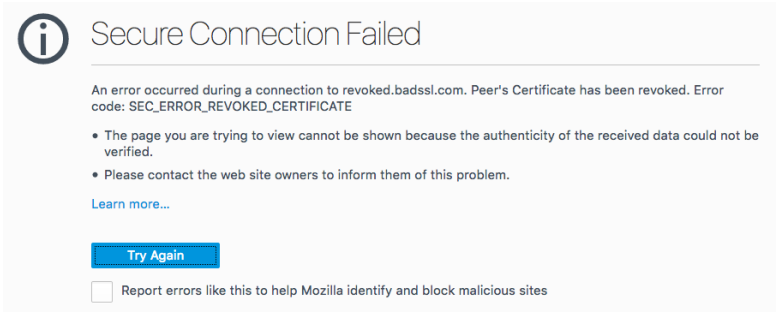

Figura 3.3: Página Web con certificado revocado

3.2. Algoritmo de puntuación

Con todos los parámetros obtenidos, se va a diseñar un algoritmo de puntuación de prueba para verificar el correcto funcionamiento de la extensión. Este algoritmo también podrá servir de base para el diseño final del mismo.

El algoritmo que nos permitirá valorar la seguridad de la conexión Web se ha dividido en dos partes. Primero se comprueban los parámetros críticos de la conexión (apartado 2.1), y si alguno de ellos es incorrecto en alguno de los certificados de la cadena, se valorará la conexión con un 0, puesto que la cadena no es fiable y en consecuencia la comunicación tampoco.

Por otra parte, en la segunda mitad se valorará la conexión con una nota del 0 al 10. Para ello se aplicarán una serie de fórmulas que se mostrarán a continuación. Estas se componen por variables P_ y W_, las primeras hacen referencia a punutaciones que

pueden tomar valores desde 0 hasta 10. Por otra parte, las segundas son las ponderaciones que se aplican en las distintas fórmulas. Estas pueden tomar valores desde 0 hasta 1, y la suma de todas las ponderaciones contenidas en una misma fórmula siempre es igual a 1.

$$PT = PC * WC + PCC * WCC \tag{3.1}$$

La fórmula 3.1 es la que se aplicará para conseguir una valoración total (PT), y está dividida en dos partes distintas. La primera valora los parámetros que tan solo afectan a la conexión directamente (PC) y en la segunda se puntúa la cadena de certificados (PCC).

$$PC = PCS * WCS + PEV * WEV \tag{3.2}$$

En la fórmula 3.2 se puede observar como los parámetros que afectan directamente a la conexión son el *cipher suite* (PCS) y el valor que indica si el certificado es *Extended Validated* (PEV). Este segundo es el único de los campos que se puede encontrar entre las extensiones del certificado que no se consideran críticos. Este parámetro aporta mayor fiabilidad al certificado del servidor, pero no afecta al resto de la cadena de certificación. Por ello se ha decidido introducir en esta parte de la puntuación.

$$PCS = PTLS * WTLS + PKE * WKE + PBC * WBC + PH * WH \tag{3.3}$$

Para valorar el *cipher suite* se puntúa cada una de sus partes independientemente: versión del protocolo SSL/TLS ($PTLS$), algoritmos de intercambio de clave (PKE), de cifrado simétrico (PBC) y de hash (PH). La valoración que recibirá cada una de estas partes dependerá de la fortaleza de dicho protocolo SSL/TLS o algoritmo, para ello se utilizará la tabla 3.1 para mostrar la valoración de todos los que se tendrán en cuenta en la implementación. Además, cabe destacar que la propiedad *perfect forward secrecy* que se ha explicado en el apartado 2.1 se ha tenido en cuenta dentro de la puntuación que se le ha dado a cada uno de los algoritmos de intercambio de clave.

$$PCC = \sum_{i=1}^{CertLength} WCT_i * (PV_i * WV + PPK_i * WPK + PSA_i * WSA) \tag{3.4}$$

Por otra parte, los certificados que componen la cadena de certificación se valorarán por separado, asignándoles una ponderación a cada uno de ellos (WCT_i). Para valorar cada uno de estos individualmente se tendrán en cuenta tres parámetros que serán: la validez de este (PV) con dos valores posibles 0 o 10, la clave pública (PPK) que se comprobará de qué tipo es y se puntuará de acuerdo a su longitud (tabla 3.2), y finalmente el algoritmo de firma (PSA) que obtendrá una valoración del 0 al 10 (tabla 3.3).

Cabe mencionar que la asignación de las puntuaciones se ha hecho en base a los datos extraídos de los RFCs 8247 y 7465. Además, estas puntuaciones se han definido para poder implementar el algoritmo de prueba en la extensión y son modificables. Es decir, se pueden incluir los algoritmos y longitudes de claves que sean necesarios, así como también cambiar la valoración de estos para poder desarrollar un algoritmo más refinado en un futuro. Por otra parte, las equivalencias entre longitudes de clave RSA

y ECC (tabla 3.2) se han obtenido del National Institute of Standards and Technology (NIST) [16].

Tabla 3.1: Puntuaciones de los algoritmos del *cipher suite*

Versión SSL/TLS		**Intercambio de clave**		**Cifrado en bloque**		**HMAC**	
SSL 0.3	0	ECDHE_RSA	10	CHACHA20_POLY1305	10	SHA_512	10
TLS 1.0	2	ECDH	9	AES_256_GCM	10	SHA_384	10
TLS 1.1	6	DHE_RSA	9	AES_128_GCM	8	SHA_256	9
TLS 1.2	10	DH	7	AES_256_CBC	7	SHA_224	8
TLS 1.3	10	RSA	5	CAMELLIA_256_CBC	7	SHA1	5
				CAMELLIA_128_CBC	7	MD5	0
				3DES_EDE_CBC	3		
				RC2_CBC_40	2		
				RC4	0		
				DES	0		

Tabla 3.2: Puntuaciones de las longitudes de claves RSA y ECC

RSA		**ECC**	
1024 bits	5	≥160 bits	5
2048 bits	9	≥224 bits	9
≥ 3072 bits	10	≥256 bits	10

Tabla 3.3: Puntuaciones de los algoritmos de firma

Algoritmo de firma			
ECDSA_SHA512	10	RSA_SHA512	10
ECDSA_SHA384	10	RSA_SHA384	9
ECDSA_SHA256	10	RSA_SHA256	8
ECDSA_SHA224	9	RSA_SHA224	7
ECDSA_SHA1	3	RSA_SHA1	2
ECDSA_MD5	0	RSA_MD5	0

3.3. Implementación

Para poder explicar la implementación que se ha realizado se empezará por describir el panel que se ha diseñado, con el contenido que se mostrará en este. Más adelante se enseñará cómo controlar el contenido de dicho panel y qué cambios se han realizado en los *scripts ssleuth-ui.js* y *ssleuth.js*. Finalmente, se explicará cómo se ha implementado el algoritmo de puntuación, viendo como los factores de ponderación son totalmente configurables por el usuario.

3.3.1. Funciones encargadas de la construcción del panel

Se ha comenzado modificando el módulo *panel.js* para poder mostrar por pantalla un panel con tan solo dos pestañas. En una de ellas se mostrará el *cipher suite* así como también datos del certificado del servidor Web que no comparten los demás certificados de la cadena. Uno de ellos es para indicar si el certificado es *Extended Validated* y el otro indica si el dominio de la página Web es correcto.

Por otra parte, la segunda pestaña mostrará la información de la cadena de certificados entre la que se podrá encontrar: el algoritmo de firma, la longitud y tipo de clave pública y la información referente al firmante y al sujeto de cada uno de los certificados de la cadena. Además, si se encuentra algún error crítico en algún certificado de la cadena, se notificará a través de la pestaña dedicada a la cadena de certificados. La única excepción a esta afirmación ocurre cuando el nombre de dominio no es el correcto, información que se proporcionará a través de la pestaña de conexión.

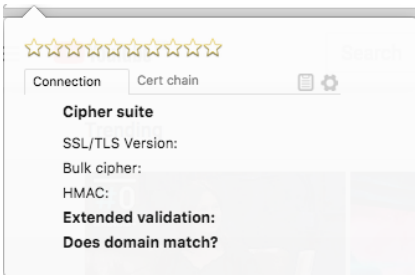

Figura 3.4: Pestaña conexión

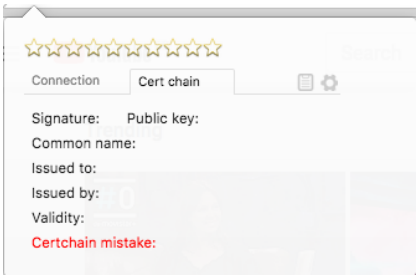

Figura 3.5: Pestaña cadena de certificados

Para poder navegar entre los distintos certificados de la cadena se utilizará un desplegable, para ello se reutilizará el código que *SSLeuth* utiliza para implementar la pestaña *Domains*. Las líneas de código involucradas en este proceso están reflejadas en el código 3.1. Lo primero que se debe hacer es introducir dentro del panel de la cadena de certificados un *grid*, el cual contendrá los elementos *rows* para poder mostrar como un desplegable, los nombres que representarán a los distintos certificados de la cadena (líneas 2 a 6).

Una vez se haya realizado dicha modificación en el módulo *panel.js* se deberá controlar el contenido de este panel, para ello se utilizará la función *loadCertChain(win)* del *script ssleuth-ui.js* (línea 23 a 87). Dentro de esta se deberá acceder a la cadena de certificados, pero al no poder pasarse el certificado del servidor por parámetro, se definirá un atributo global dentro del módulo que contendrá esta información (*var certificado*, línea 10).

A continuación se almacenarán los certificados de la cadena dentro de un *array* utilizando el atributo *issuer* para ir accediendo a toda la cadena, y el valor booleano *isSelfSigned* para comprobar que ya se ha acabado la cadena (líneas 32 a 36). Antes de empezar a rellenar las *rows* se deberá borrar cualquier información que esta contenga con anterioridad (líneas 39 a 41). Además, se utilizará la variable *displayName* del atributo *ASN1Structure* para identificar cada uno de los certificados (líneas 55 a 69).

Finalmente, se le debe asignar un *listener* a cada uno de los certificados del desplegable, para actualizar la información de la pestaña correctamente (líneas 75 a 81). Es decir, estos *listeners* serán los encargados de llamar a la función *showSelectedCertDe-*

tails(cert, win), pasándole por parámetro el certificado seleccionado en el desplegable. Esta función, *showSelectedCertDetails(cert, win)*, se encargará de mostrar los datos de este certificado en la pestaña dedicada a la cadena de certificados, su funcionamiento se explicará más adelante en este apartado.

Código 3.1: Funciones involucradas en el desplegable de la cadena de certificados

```
1  //PANEL.JS
2  let grid = hb.appendChild(elem('grid', {})); {
3      grid.appendChild(elem('rows', {
4      id: 'ssleuth-cert-chain-rows'
5      }));
6  }
7
8  //SSLEUTH-UI.JS
9  //Variable global del modulo que contendra el certificado del
       servidor
10 var certificado = '';
11
12 //Funcion que llama a loadCertChain(win) para caragar su
       contenido en el panel
13 function panelVisible(win) {
14     loadCertChain(win);
15 }
16
17 //Actualizar el contenido del atributo certificado
18 function fillPanel(data, win, winId) {
19     this.certificado = data.cert;
20 }
21
22 //Cargar la informacion en el desplegable y rellenar el panel
23 function loadCertChain(win){
24     try{
25
26         var doc = win.document;
27         var rows = doc.getElementById('ssleuth-cert-chain-rows')
               ;
28         var cert = this.certificado; //USADO PARA CONSTRUIR EL
               ARRAY CERTCHAIN Y AVERIGUAR LA LONGITUD DE LA CADENA
29         var certChain = [];
30
31         //Llenar el array de la cadena de certificados
32         while(!cert.isSelfSigned){
33             certChain.push(cert);
34             cert = cert.issuer;
35         }
36         certChain.push(cert);
37
38         //Reiniciar el elemento rows
39         while (rows.hasChildNodes()) {
40             rows.removeChild(rows.firstChild);
41         }
42
43         var row = rows.appendChild(create(doc, 'row', {
44             align: 'baseline'
45         }));
46
47         var m_list = row.appendChild(create(doc, 'menulist', {
```

```
48              class: 'ssleuth-panel-certChain-menulist'
49          }));
50
51          var m_popup = m_list.appendChild(doc.createElement('
                menupopup'));
52
53          for (var j = 0; j < certChain.length; j++){
54
55              if((certChain[j].ASN1Structure.displayName.indexOf(
                    utils.getText('builtin.text')) != -1)
56              ||(certChain[j].ASN1Structure.displayName.indexOf(
                    utils.getText('software.text')) != -1)){
57                  //NOMBRE QUE SE MOSTRARA EN EL DESPLEGABLE PARA
                        REPRESENTAR EL CERTIFICADO J
58                  //SE HA UTILIZADO UN SPLIT ":" PARA ELIMINAR
                        SUBSTRINGS
59                  var mi = m_popup.appendChild(create(doc,'
                        menuitem',{
60                      label: certChain[j].
                            ASN1Structure.displayName.split(":")[1],
61                      value: j
62                  }));
63              }else{
64                  //NOMBRE QUE SE MOSTRARA EN EL DESPLEGABLE PARA
                        REPRESENTAR EL CERTIFICADO J
65                  var mi = m_popup.appendChild(create(doc,'
                        menuitem',{
66                      label: certChain[j].
                            ASN1Structure.displayName,
67                      value: j
68                  }));
69              }
70              //SE SELECCIONA EL CERTIFICADO ELEGIDO
71              if(certificadoSeleccionado == mi.value){
72                  m_list.selectedItem = mi;
73              }
74              //LISTENER QUE SE LE IMPLEMENTA A CADA CERTIFICADO
                    DEL DESPLEGABLE
75              m_popup.addEventListener('command', function (event)
                     {
76                  //SE CAMBIA EL VALOR DEL ATRIBUTO "
                        certificadoSeleccionado" PARA QUE ESTE SE
                        SELECCIONE EN EL DESPLEGABLE
77                  certificadoSeleccionado = event.target.value;
78                  //RELLENAR PANEL DE CERTIFICATE CHAIN DEPENDIEDO
                         DEL CERTIFICADO SELECCIONADO
79                  showSelectedCertDetails(certChain[
                        certificadoSeleccionado], win);
80
81              }, false);
82
83          }
84
85      }catch(e){
86          log.debug('Error loadCertChain' + e.message);
87      }
88  }
```

3.3.2. Funciones encargadas de introducir la información en el panel

Una vez se ha implementado el panel donde se introducirá toda la información de seguridad, se deben construir las funciones que se encargarán de rellenarlo. Una de ellas seguirá siendo *fillPanel()* del módulo *ssleuth-ui.js* de la cual se han eliminado las funciones que ya no hacen falta, se han modificado algunas que se utilizan de forma diferente y se han creado nuevas (código 3.2).

Código 3.2: Función fillPanel() en ssleuth-ui.js (modificada)

```
1  function fillPanel(data, win, winId) {
2    //Actualiza el atributo certificado
3      this.certificado = data.serverCert;
4    //Conservadas para mostrar valoracion global
5      setButtonRank(data.rating, 'https', win);
6      panelConnectionRank(data.rating, win);
7    //Modificada para mostrar informacion deseada
8      showCipherDetails(data.cipherSuite, data.tls, win);
9      //Parametros unicos del certificado del servidor
10     showEVandDomain(data.ev,data.domMismatch,win);
11     //Mostrar informacion de la cadena de certificados
12     this.certificadoSeleccionado = 0;
13     showSelectedCertDetails(data.serverCert,win);
14     showCertMistake(data.certMistake, win);
15 }
```

Primero se actualiza el atributo *certificado*, como ya se ha explicado anteriormente (línea 3), y se reutilizan las funciones que permiten mostrar la valoración global de la conexión (líneas 5 y 6). A continuación, se utiliza la función *showCipherDetails*, pero introduciendo algunos cambios en esta (código 3.3). Se eliminan los campos de información que no se desean mostrar y se cambia el objetivo de utilización del campo identificado por el nombre *ssleuth-text-cipher-suite-kxchange-notes*. Este elemento contiene información referente al algoritmo de intercambio de clave, por ello se hará uso de este para indicar si proporciona a la comunicación *perfect forward secrecy* (líneas 13 y 14). Además, se ha cambiado el sistema que utilizaba *ssleuth* para mostrar el protocolo SSL/TLS utilizado, puesto que ahora se debe pasar a través del módulo *ssleuth.js* como atributo al llamar a la función *protocolChange()* (línea 8).

Código 3.3: Función showCipherDetails() en ssleuth-ui.js (modificada)

```
1  function showCipherDetails(cipherSuite, tls, win) {
2      var doc = win.document;
3      const cs = ciphersuites;
4      //NOMBRE COMPLETO DEL CIPHERSUITE
5      doc.getElementById('ssleuth-text-cipher-suite').textContent
           =
6          (cipherSuite.name);
7      //VERSION SSL/TLS
8      doc.getElementById('ssleuth-text-tls-version').textContent =
           tls.ui + '.';
9      //ALGORITMO DE INTERCAMBIO DE CLAVE
10     doc.getElementById('ssleuth-text-cipher-suite-kxchange').
           textContent =
11         (cipherSuite.keyExchange.ui + '.');
12     //UTILIZADO PARA MOSTRAR SI SE CUMPLE LA PROPIEDAD DE
           PERFECT FORWARD SECRECY
```

```
13      doc.getElementById('ssleuth-text-cipher-suite-kxchange-notes
            ').textContent =
14          utils.getText(cipherSuite.keyExchange.notes);
15      //ALGORITMO DE CIFRADO SIMETRICO JUNTO CON EL MODO
16      doc.getElementById('ssleuth-text-cipher-suite-bulkcipher').
            textContent =
17          (cipherSuite.bulkCipher.ui + ' ' +
                cipherSuite.cipherKeyLen +
18              ' ' + utils.getText('general.bits') + '.');
19      //ALGORITMO DE HASH
20      doc.getElementById('ssleuth-text-cipher-suite-hmac').
            textContent =
21          (cipherSuite.HMAC.ui + '. ');
22  }
```

Una vez rellenada la información de la pestaña de conexión, se debe hacer lo mismo con la de la cadena de certificados mediante la función *showSelectedCertDetails()* (código 3.2, línea 13). Esto debe hacerse puesto que al acceder a la página Web no se ha activado ningún *listener* del desplegable de la cadena, y en consecuencia no se mostraría ninguna información por el panel. Para solucionarlo se ha establecido que siempre que se cambie de pestaña se muestre la información del certificado del servidor Web (código 3.2, línea 12). La función *showSelectedCertDetails()* se ha basado en la función *showCertDetails()* de *SSLeuth*, pero no se puede reutilizar la parte que permite mostrar el algoritmo de firma y la clave pública. El motivo de esto es que el algoritmo de firma se procesaba en el módulo *ssleuth.js*, y ahora debe hacerse en *ssleuth-ui.js*, puesto que el contenido de esta pestaña del panel ahora cambia dinámicamente debido al desplegable.

Código 3.4: Cambios en la función showCertDetails() en ssleuth-ui.js

```
1  //RELLENAR ALGORITMO DE FIRMA DEL CERTIFICADO
2  var signatureAlg = getSignatureAlgParams(cert);
3  doc.getElementById('ssleuth-text-cert-sigalg')
4      .textContent = signatureAlg;
5
6  //RELLENAR CLAVE PUBLICA DEL CERTIFICADO
7  var pubKeyAlg = getPubKeyAlg(cert);
8  var pubKeySize = getKeySize(cert, pubKeyAlg);
9  doc.getElementById('ssleuth-text-cert-pub-key')
10     .textContent = (pubKeySize + ' ' + utils.getText('
           general.bits') + ' ' + pubKeyAlg);
```

El fragmento de código 3.4 pertenece a la función *showSelectedCertDetails*, y este se encarga de procesar la información del certificado. Se han reutilizado las funciones *getSignatureAlgParams()* y *getKeySize* para obtener el algoritmo de firma (línea 2 a 4) y la longitud de la clave pública respectivamente (líneas 7 a 10).

La función *getSignatureAlgParams()* (código 3.5) es una modificación de la llamada *getSignatureAlg()* utilizada en *SSLeuth*. Esta función accede al campo que contiene el algoritmo de firma del certificado (líneas 2 a 6) y lo extrae (líneas 22 a 28). Para conseguir extraerlo se consulta el módulo *cipher-suite.js*, pero de una forma diferente a como lo hace *SSLeuth*. En *SSLeuth* se reutilizaban los algoritmos del *cipher suite* para conseguir el algoritmo de firma. Pero se ha considerado que esta práctica no es la correcta, y se ha creado una nueva constante dentro de *cipher-suites.js* llamada *certificate*. Este cambio

se ha hecho también para poder valorar correctamente el algoritmo de firma en el módulo *ssleuth.js*. Ya que en *SSLeuth* se utilizaba la misma puntuación del algoritmo de hash para valorar el algoritmo de firma, como la función HMAC del *cipher suite*. Al definir una nueva constante en *cipher-suites.js* se han podido separar ambos conceptos, y asignarles una puntuación independiente a cada uno de ellos.

Código 3.5: Función getSignatureAlgParams() en ssleuth-ui.js

```
1  function getSignatureAlgParams(cert) {
2          var certASN1 = Cc['@mozilla.org/security/nsASN1Tree;1']
3              .createInstance(Ci.nsIASN1Tree);
4          certASN1.loadASN1Structure(cert.ASN1Structure);
5          //OBTENEMOS ALGORITMO DE FIRMA DEL CERTIFICADO
6          var sigText = certASN1.getDisplayData(4).replace(/PKCS
               #1/g, '');
7          //ALGUNOS ALGORITMOS SOLO CONTIENEN EL OID
8          if (sigText.indexOf('1 2 840 10045') != -1) {
9              if (sigText.indexOf('1 2 840 10045 4 1') != -1) {
10                 sigText = 'ECDSA with SHA-1';
11             } else if (sigText.indexOf('1 2 840 10045 4 3 1') !=
                   -1) {
12                 sigText = 'ECDSA with SHA-224';
13             } else if (sigText.indexOf('1 2 840 10045 4 3 2') !=
                   -1) {
14                 sigText = 'ECDSA with SHA-256';
15             } else if (sigText.indexOf('1 2 840 10045 4 3 3') !=
                   -1) {
16                 sigText = 'ECDSA with SHA-384';
17             } else if (sigText.indexOf('1 2 840 10045 4 3 4') !=
                   -1) {
18                 sigText = 'ECDSA with SHA-512';
19             }
20         }
21         //EXTRAEMOS EL ALGORITMO DE FIRMA
22         const ct = certificate;
23         for (var i = 0; i < ct.signatureAlg.length; i++){
24             if  ((sigText.indexOf(ct.signatureAlg[i].sign) != -1
                   ) &&
25                 ((sigText.indexOf(ct.signatureAlg[i].hash1) !=
                       -1) || (sigText.indexOf(ct.signatureAlg[i].
                       hash2) != -1))){
26                  return ct.signatureAlg[i].ui;
27             }
28         }
29 }
```

La función *getKeySize()* (código 3.4, línea 8) se utiliza para para obtener la longitud de la clave pública del certificado, como ya se ha mencionado antes. Pero esta función necesita un parámetro que indique cuál es el algoritmo de clave pública. Esta información era fácilmente accesible cuando se trataba de analizar tan solo el certificado del servidor, puesto que se tenía una referencia a la interfaz *nsISSLStatus* que contiene el *cipher suite* (atributo *cipherName*) donde se indica dicho algoritmo. Ahora esto ya no es posible, y se ha diseñado otra función llamada *getPubKeyAlg(cert)* que permite extraer dicha información a través del certificado (código 3.6). En esta se accede a la estructura ASN1 del certificado, y se comprueban las posiciones donde se puede

encontrar el campo *Algorithm identifier* dependiendo de si la clave pública es RSA o de curva elíptica.

Código 3.6: Funcion getPubKeyAlg() en ssleuth-ui.js

```
1  function getPubKeyAlg(cert){
2      try{
3          var certASN1 = Cc['@mozilla.org/security/nsASN1Tree;1']
4              .createInstance(Ci.nsIASN1Tree);
5          certASN1.loadASN1Structure(cert.ASN1Structure);
6          //PRIMERO COMPROBAMOS QUE SEA RSA
7          if((certASN1.getDisplayData(11)).indexOf('RSA') != -1){
8              return 'RSA';
9          }
10         //SI NO ES RSA ENTONCES COMPROBAMOS QUE SEA CURVA
               ELIPTICA
11         else if(certASN1.getDisplayData(12).indexOf('Elliptic')
               != -1){
12             return 'ECC';
13         }
14         //TRATAR CASO POR DEFECTO
15         else{
16             return 'UNKNOWN';
17         }
18     }catch(e){
19         return e.message;
20     }
21 }
```

Una vez se han hecho los cambios necesarios en los módulos *panel.js* y *ssleuth-ui.js*, faltaría aplicar el algoritmo de puntuación en el script *ssleuth.js*. Este módulo se encargará de pasar los parámetros necesarios junto con la puntuación a través del método *protocolChange()*, el cual se encuentra dentro de la función *protocol.onHttps()*, al igual que ocurre en *SSLeuth* (apartado 2.3.2). Esta función ha sido modificada de tal forma que nos permita implementar el algoritmo de puntuación y así poder obtener la valoración global de la conexión (código 3.7). También es la encargada de transmitir a *ssleuth-ui.js* la información que introducirá en la pestaña de conexión del panel. Se han eliminado algunas variables y se han introducido otras (líneas 3 a 14), como por ejemplo el objeto *cert* que antes tenía varios atributos, ahora solo contiene el certificado del servidor. Ya que como se ha explicado antes, la pestaña dedicada a la cadena de certificados es dinámica, y la información se extrae dentro de *ssleuth-ui.js*. Además, se ha reutilizado la función *getCsParam()* para extraer los datos acerca de los algoritmos que componen el *cipher suite* de la conexión (líneas 16 a 26), y la condición que nos permite saber si el certificado es *Extended Validated* (líneas 38 a 40). También se ha modificado el método *setTLSVersion()* por *getTLSVersion()* para que devuelva el protocolo SSL/TLS utilizado, en vez de pasar dicha información a través del módulo *observer.js* como se hacía antes (líneas 29 a 35).

A continuación se comprueba si el nombre de dominio no es válido y que el protocolo SSL/TLS utilizado no esté puntuado con un 0 (línea 43). Si alguna de las dos condiciones se cumple, se valorará la conexión con un 0, en caso contrario se consultarán las ponderaciones de las diferentes partes del algoritmo de puntuación (líneas 49 a 52). Después se aplica el algoritmo de prueba, desde la línea 55 hasta la 92 se puede observar como se calculan las diferentes partes que lo componen. Aunque cabe

destacar la sección que se encarga de valorar la cadena de certificados, esta puede contener varios errores que se explicarán a continuación:

- *noError*: valor que toma esta variable por defecto, el cual conserva si no se ha detectado ningún parámetro crítico incorrecto dentro de la cadena de certificación. Además, este es el único caso en el que no aparecerá ningún mensaje de error en el panel.
- *Incomplete Chain*: este error se comprueba en la función *getCertLength()* que devuelve la longitud de la cadena, y se asegura de que el último certificado de la cadena esté autofirmado. Si no es así, el atributo *selfSigned* de este será siempre falso y habrá un momento que al intentar acceder al siguiente certificado con el atributo *issuer* devolverá un error. Este error será capturado y se introducirá el valor *Incomplete Chain* en la variable *certMistake*.
- *Repeated Cert*: este error podría considerarse no crítico, ya que puede no deberse al contenido de los certificados sino a la construcción de la cadena. Es decir, en la función *getCertLength()* también se compara en todo momento si el certificado al que se acaba de acceder es el mismo que se obtiene con el atributo *issuer*, utilizando la función *equals()* que implementa la interfaz *nsIX509Cert*. Si esta condición es cierta significa que hay certificados repetidos dentro de la cadena y por lo tanto esta puede que esté mal construida, o puede que esta sea incorrecta.
- *Web Server Certificate Mistake*: si el error se ha localizado en el certificado del servidor, se deberán comprobar las extensiones *Certificate Key Usage* y/o *Certificate Basic Constraints*.
- *CA or intermediate CA Certificate Mistake*: se deberán comprobar las CAs de la cadena, más concretamente en las extensiones *Certificate Key Usage* y/o *Certificate Basic Constraints*.
- *CA Mistake*: este error se ha definido para detectar un fallo que puede aparcer en la extensión *Certificate Basic Constraints*. Dentro de los certificados pertenecientes a CAs se utiliza esta extensión para indicar el *path length*. Este parámetro indica la cantidad de certificados para CAs intermedias que puede emitir. Por lo que se comprobará que el valor que indique sea el correcto, dependiendo del tipo de CA: CAs que tan solo emitan certificados finales (0), CAs intermedias (1 hasta *unlimited*) y CAs raíz (*unlimited*).

Si se detecta alguno de estos errores en la cadena de certificación, se almacenará el motivo de este en una variable llamada *certMistake*. La cual se utiliza al llamar a la función *showCertMistake* en *ssleuth-ui.js*, tal y como se puede observar en la línea 14 del código 3.2.

Código 3.7: Función protocol.onHttps en ssleuth.js (modificada)

```
1  var onHttps = function (state, urlChanged, win, winId) {
2    //VARIABLES QUE SE UTILIZARAN
3      const cs = ciphersuites;
4      var cipherName = sslStatus.cipherName,
5        cert = sslStatus.serverCert,
```

```
6           extendedValidation = 0,
7           rating = 0,
8           tls = null,
9         cipherSuite = {
10            name: cipherName,
11            cipherKeyLen: sslStatus.secretKeyLength,
12            bulkCipher: null,
13            HMAC: null
14        };
15      //METODO PARA EXTRAER ALGORITMOS DEL CIPHER SUITE
16      function getCsParam(param) {
17          for (var i = 0; i < param.length; i++) {
18              if ((cipherName.indexOf(param[i].name) != -1)) {
19                return param[i];
20              }
21          }
22          return null;
23      }
24      cipherSuite.keyExchange = getCsParam(cs.keyExchange);
25      cipherSuite.bulkCipher = getCsParam(cs.bulkCipher);
26      cipherSuite.HMAC = getCsParam(cs.HMAC);
27
28      //BUCLE QUE IDENTIFICA LA VERSION SLL/TLS Y EXTRAE SU
            VALORACION
29      var tlsVersion = getTLSVersion(win, winId);
30      for (var i = 0; i < cs.tlsVersions.length; i++) {
31          if (tlsVersion === cs.tlsVersions[i].name) {
32              tls = cs.tlsVersions[i];
33              break;
34          }
35      }
36
37      //SE COMPRUEBA SI EL CERTIFICADO ES EXTENDED VALIDATED
38      if (state &
            Ci.nsIWebProgressListener.STATE_IDENTITY_EV_TOPLEVEL) {
39          extendedValidation = 10;
40      }
41
42      //COMPROBAR PRIMERO SI EL DOMINIO COINCIDE Y LA VERSION SSL/
            TLS ES MAYOR O IGUAL A TLSV1.0
43      if(sslStatus.isDomainMismatch || tls.rank == 0){
44          //PUNTUACION IGUAL A CERO SI SE CUMPLE ALGUNA DE LAS
                CONDICIONES ANTERIORES
45          rating = 0;
46      }else{ //SI NO SE HA CUMPLIDO NINGUNA CONDICION, APLICAMOS
            EL ALGORITMO DE PUNTUACION
47
48          //SE EXTREN LAS PODNERACIONES DE LAS DIFERENTES PARTES
49          const totalWeighting = ssleuth.prefs['
                rating.total.params'];
50          const cxWeighting = ssleuth.prefs['
                rating.connection.params'];
51          const csWeighting = ssleuth.prefs['
                rating.ciphersuite.params'];
52          const ctWeighting = ssleuth.prefs['
                rating.certificate.params'];
53
```

```
54          //CALCULAR PUNTUACION DEL CIPHER SUITE
55          var csr =   (tls.rank * csWeighting.tlsVersion + //
                VERSION DE SSL/TLS
56                      cipherSuite.keyExchange.rank *
                            csWeighting.keyExchange + //KEY EXCHANGE
57                      cipherSuite.bulkCipher.rank *
                            csWeighting.bulkCipher +  //BULK CIPHER
58                      cipherSuite.HMAC.rank * csWeighting.hmac) /
                            csWeighting.total; //HASH
59
60          //CALCULAR PUNTUACION DE LA CONEXION
61          var cxr =   (csr * cxWeighting.cipherSuite +
62                      extendedValidation * cxWeighting.evCert) /
                            cxWeighting.total;
63
64          //CALCULAR PUNTUACION DE LA CADENA DE CERTIFICADOS
65          var certMistake = 'noError';
66          var certLength = getCertLength(cert.serverCert);
67          if(typeof certLength == 'number'){
68              var certChain = getChain(cert.serverCert, certLength
                    );
69              var ctsr = [];
70              for (var i = 0; i < certChain.length; i++){
71                  var ctr = getCertificateRating(certChain[i],
                        ctWeighting,i,certLength);
72                  if(typeof ctr != 'number'){
73                      certMistake = ctr;
74                      break;
75                  }
76                  ctsr.push(ctr);
77              }
78              if(certMistake.indexOf('noError') == -1){
79                  rating = 0;
80              }else{
81                  var ccr = ctsr[0]/(certChain.length + 1); //MAS
                        PESO AL CERTIFICADO DEL SERVIDOR
82                      ccr += ctsr[i]/(certChain.length + 1);
83                  }
84              //CALCULAR PUNTUACION TOTAL
85                  rating =    ((cxr * totalWeighting.connection +
86                            + ccr * totalWeighting.certChain) /
                                  totalWeighting.total).toFixed(1);
87              }
88          }else{
89            //ERROR CRITICO ENCONTRADO AL DETERMINAR LA LONGITUD
                  DE LA CADENA DE CERTIFICADOS
90            rating = 0;
91              certMistake = certLength;
92          }
93
94      }
95      //FUNCION DEL MODULO SSLEUTH-UI.JS
96      ui.protocolChange('https', {
97          rating: rating,
98                cipherSuite: cipherSuite,
99                  domMismatch: sslStatus.isDomainMismatch,
100                 ev: extendedValidation,
```

```
101                     prova: error,
102                     tls: tls,
103                     serverCert: cert.serverCert,
104                     certMistake: certMistake
105             },
106         win, winId);
107 };
```

Para finalizar todo este proceso, se han introducido todas las variables necesarias en la función *protocolChange()* para que pueda mostrarse la información correctamente por pantalla (líneas 96 a 106).

A parte de la implementación que se ha explicado, se han rediseñado los archivos *preferences.xul*, *preferences-ui.js* y *preferences.js* para que se adapten al algoritmo deseado y que se le permita al usuario cambiar las distintas ponderaciones. Puesto que el funcionamiento de estos archivos se ha explicado en el apartado 2.3.2, no se detallarán los cambios que se han realizado. Aunque en el apartado siguiente se mostrará un ejemplo en el que se verá el diseño final de la página de preferencias, para comprobar que esta funciona correctamente.

También se han introducido dentro del *script cipher-suites.js* todos los algoritmos y protocolos SSL/TLS necesarios (tabla 3.1) con sus respectivas puntuaciones, así como también las variables necesarias para almacenar las ponderaciones establecidas por defecto. Además, se han introducido modificaciones en el archivo CSS que controla el estilo del panel (*ssleuth.css*), así como también en los archivos almacenados en la carpeta *locale* que contienen los strings para los disitntos idiomas. Finalmente, se han eliminado del código todas las funcionalidades implementadas anteriormente que podían generar conflictos con la nueva extensión.

3.4. Pruebas de funcionamiento

Una vez modificada la extensión se han realizado pruebas con diversas páginas Web para demostrar el buen funcionamiento de esta. Primero se ha comprobado que la puntuación que se le asigna a una página Web es la correcta, teniendo en cuenta el algoritmo de prueba. A continuación se ha accedido a páginas Web con errores críticos en la cadena de certificación, para poder ver como la extensión localiza dichos fallos. Y finalmente se ha realizado una comparación con la extensión *SSLeuth*, analizando porqué la puntuación que se le asigna a una misma página Web no es la misma en ambas extensiones.

3.4.1. Aplicación del algoritmo de prueba y ponderaciones configurables

En estas pruebas se accederá a una página Web (página I) sin realizar ningún cambio en las ponderaciones, haciendo uso de las configuradas por defecto (tabla 3.4). A continuación se comprobará que la información de seguridad que muestra la extensión es la correcta, y se verificará que el algoritmo de prueba se aplica correctamente. Después se utilizará la página de preferencias de la extensión para cambiar el valor de algunas ponderaciones (tabla 3.4), y así ver como varía la valoración global de la conexión. Cabe destacar que en la tabla de ponderaciones se le han asignado números

del 1 al 3 a los certificados, perteneciendo estos al servidor Web (1), CA intermedia (2) y CA raíz (3).

Tabla 3.4: Tabla de ponderaciones por defecto y modificadas

Ponderaciones		**Por defecto**	**Modifcadas**
Certificado 1	WCT1	0,5	0,5
Certificado 2	WCT2	0,25	0,25
Certificado 3	WCT3	0,25	0,25
Validez	WV	0,2	0,1
Clave pública	WPK	0,3	0,2
Algoritmo de firma	WSA	0,5	0,7
Versión SSL/TLS	WTLS	0,1	0,1
Alg. interc. de claves	WKE	0,4	0,3
Alg. cifrado en bloque	WBC	0,2	0,3
HMAC	WH	0,3	0,3
Cipher suite	WCS	0,8	0,7
Extended Validation	WEV	0,2	0,3
Conexión	WC	0,6	0,5
Cadena de certificados	WCC	0,4	0,5

Al acceder a la página I se ha comprobado como la información de seguridad que nos proporciona el panel es la correcta (figuras 3.6 y 3.7). Para ello se ha accedido a esta misma información a través del navegador y se ha comprobado que ambas coinciden. Por otra parte, para comprobar que el algoritmo de prueba se ha aplicado correctamente, se ha elaborado la tabla 3.5. En esta tabla se muestra la puntuación que tiene cada una de las partes del algoritmo de acuerdo a lo especificado en las tablas 3.1, 3.3 y 3.2.

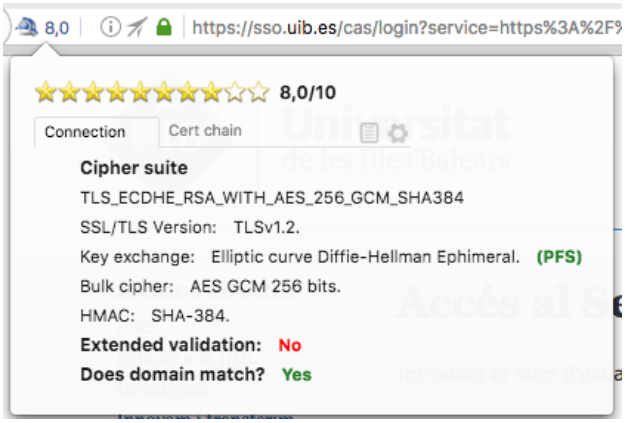

Figura 3.6: Puntuación y pestaña de conexión (página I)

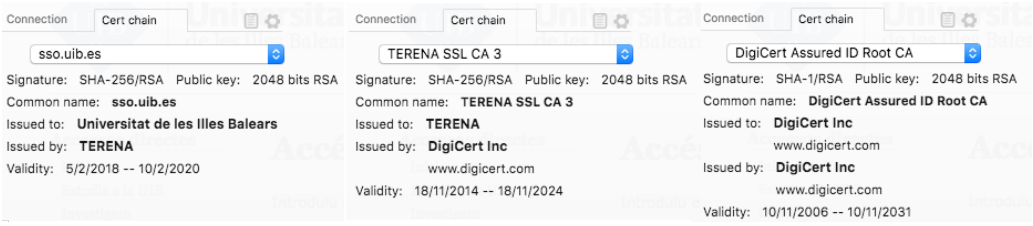

Figura 3.7: Pestaña de la cadena de certificados (página I)

Tabla 3.5: Cálculo puntuación total (página I, ponderaciones por defecto)

Validez	Clave Pública	Firma	
PV1 = 10	PPK1 = 9 (2048 bits RSA)	PSA1 = 8 (SHA256/RSA)	
PV2 = 10	PPK2 = 9 (2048 bits RSA)	PSA2 = 8 (SHA256/RSA)	
PV3 = 10	PPK3 = 9 (2048 bits RSA)	PSA3 = 2 (SHA1/RSA)	
Versión SSL/TLS	Intercambio de claves	Cifrado en bloque	HMAC
PTLS = 10 (TLS 1.2)	PKE = 10 (ECDHE_RSA)	PBC = 10 (AES256 GCM)	PH = 10 (SHA384)
Extended Validation			
PEV = 0			

Una vez se han recopilado todos los datos necesarios para aplicar el algoritmo de prueba, se ha aplicado y obtenido como resultado 7,98. Como se puede observar es la misma puntuación que aparece en la figura 3.6, puesto que se redondea a los decimales y da como resultado 8. Ahora que se ha podido comprobar como el algoritmo de prueba se aplica correctamente con las ponderaciones por defecto, se cambiará el valor de estas (tabla 3.4) con ayuda de la página de preferencias (figura 3.8), y se observará el resultado obtenido en la extensión.

Se puede observar como la puntuación obtenida ahora es un 7,2 (figura 3.9), y al calcularla manualmente se llega al mismo resultado (7,18). Por lo que con esta modificación se ha comprobado el buen funcionamiento del algoritmo en la extensión. Además, se puede analizar el porqué de esta bajada de puntuación observando los datos de la tabla 3.5. En estos se puede observar como el algoritmo de firma del certificado de la CA raíz está valorado con un 2, además el certificado del servidor Web no es *Extended Validated*. Estas han sido las razones principales por las que la puntuación ha bajado, ya que se le ha dado más importancia (ponderación más alta) a estos parámetros tras la modificación.

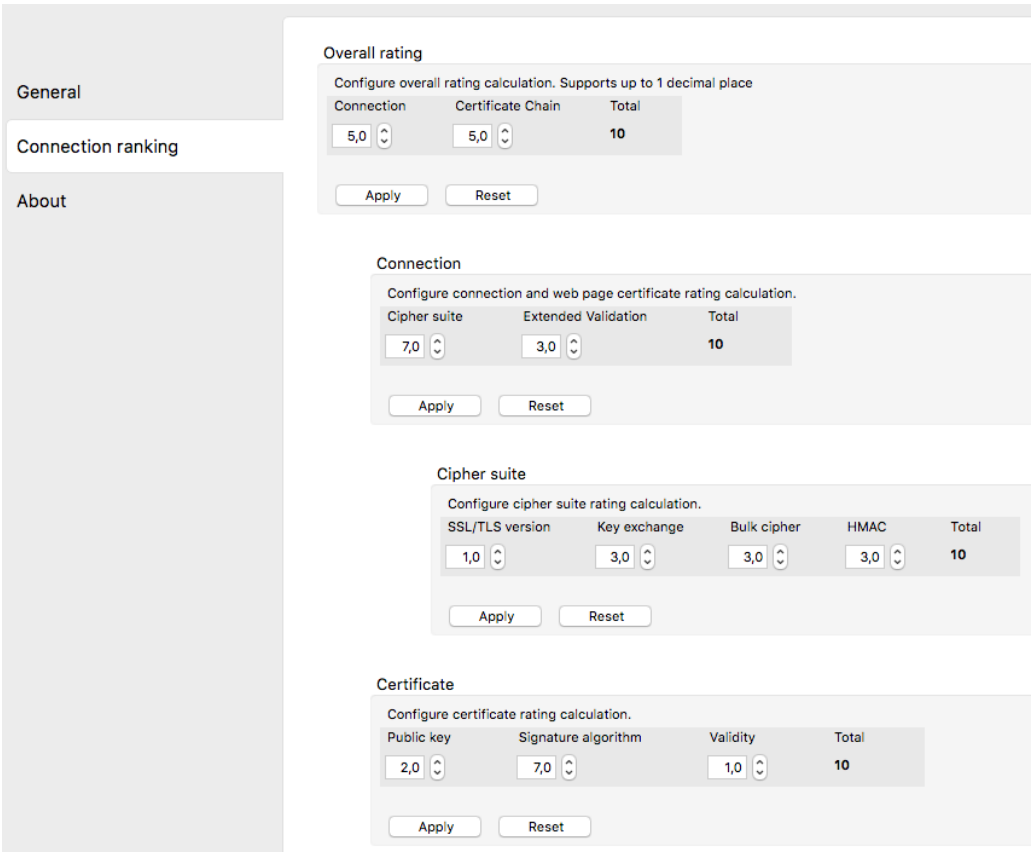

Figura 3.8: Página de preferencias de la nueva extensión

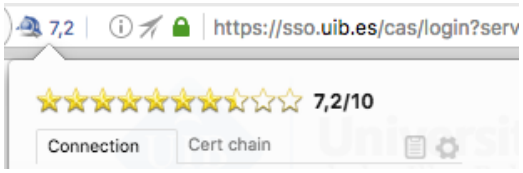

Figura 3.9: Puntuación tras modificación de ponderaciones (página I)

3.4.2. Errores críticos en la cadena de certificación

Otras páginas Web en las que se puede probar la nueva extensión son aquellas en las que la cadena de certificados contiene errores. Para ello se ha accedido a varios dominios que contienen fallos en sus ceritifcados:

- Certificado expirado: se ha accedido a una página Web (página II) con un certificado expirado. Como puede observarse en la figura 3.10 este ha caducado hace tiempo. Por lo tanto, se indica dicho dato en el panel y se aplica el algoritmo de puntuación acorde a dicha información. Al no ser un error crítico no se la valora con un 0 automáticamente.

- Nombre de dominio incorrecto: como se puede observar en la figura 3.11 el certificado de la página Web (página III) indica que el nombre de dominio para el que ha sido expedido es **.badssl.com* o *badssl.com*. Este dato viene indicado en la extensión *Certificate Subject Alt Name*. Para ser válido el dominio *wrong.host.badssl.com* no debería tener un punto entre *wrong* y *host*, por ello la valoración es de 0 y se indica en el panel que se debe al nombre de dominio.

- Certificado autofirmado: en el certificado de la siguiente página Web (página IV) se puede observar como la longitud de la cadena es de uno (figura 3.12). Por lo tanto, al analizar el campo del firmante (*issuer*) se ha comprobado como es el mismo que el del sujeto, es decir, ha sido autofirmado. Pero no es posible por varias razones, una es que no contiene la extensión *Certificate Key Usages* donde podría indicarse que este certificado puede firmar otros certificados. Además, si se observa el contenido de la extensión *Certificate Basic Constraints* esta indica que no es una CA. Por lo tanto, en el panel se indica que se ha encontrado un error crítico en el certificado del servidor.

- Cadena de certificados incompleta: para poder identificar una cadena incompleta se ha accedido a una página Web (página V) que contiene los mismos parámetros que en el caso anterior, pero en el campo del firmante (*issuer*) se ha comprobado que este no es el mismo que el del sujeto (figura 3.13). Por ello, se interpreta como una cadena de certificados incompleta, se valora con un 0 y se indica por el panel que este ha sido el motivo.

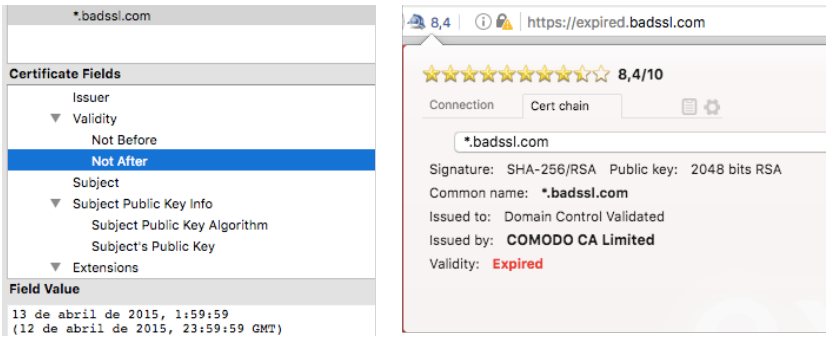

Figura 3.10: Certificado expirado (página II)

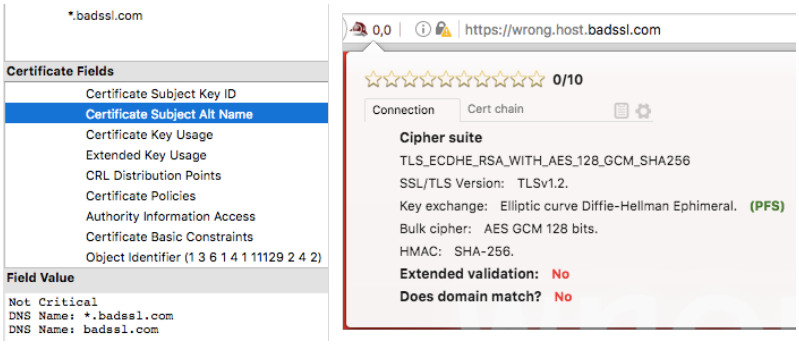

Figura 3.11: Nombre de dominio no válido (página III)

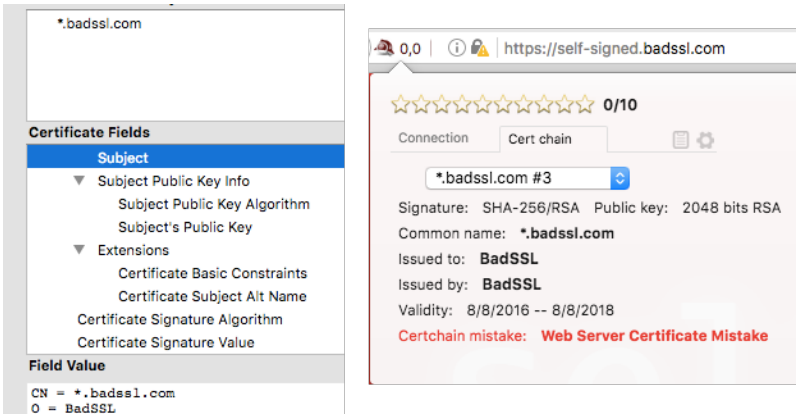

Figura 3.12: Certificado autofirmado sin *keyUsage* (página IV)

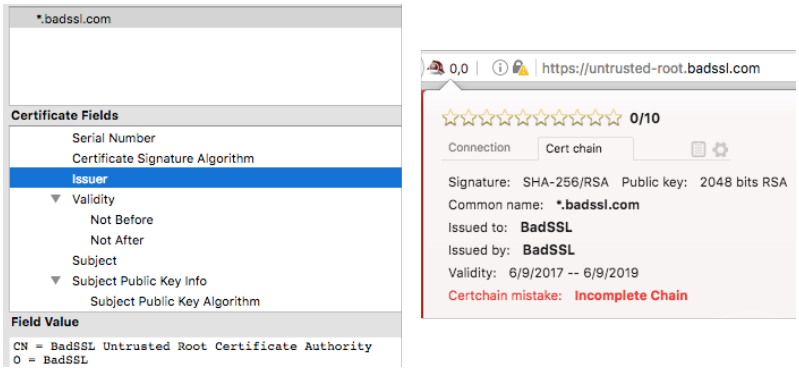

Figura 3.13: Cadena de certificación incompleta (página V)

3.4.3. Comparación con valoraciones de *SSLeuth*

Para finalizar, en este apartado de pruebas se analizarán las diferencias en el sistema de puntuación respecto a *SSLeuth*. Para ello se visitará la misma página Web (página VI) con ambas extensiones, se observarán los resultados y se analizará porqué son diferentes.

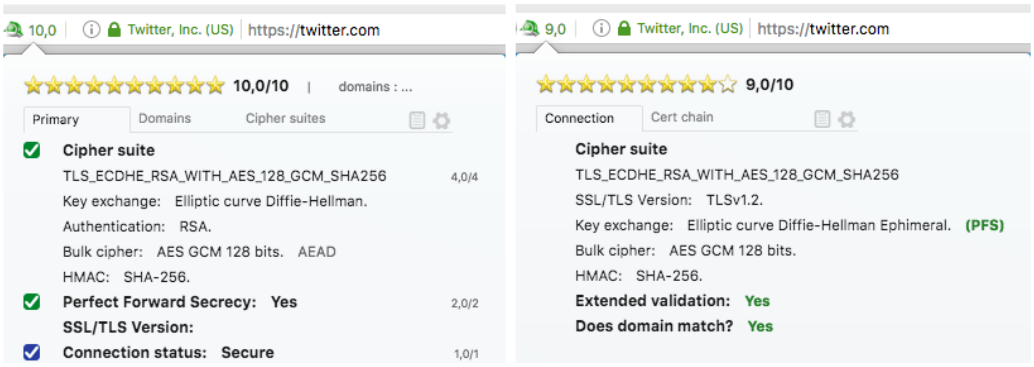

Figura 3.14: Comparación de las puntuaciones globales (página VI)

Como se puede observar en la figura 3.14 la valoración obtenida por la extensión *SSLeuth* es un 10, pero en la nueva extensión se le da un 9. A pesar de ser todavía muy alta la puntuación, esta ha bajado un punto entero. El motivo principal de esta reducción es que *SSLeuth* no comprueba la cadena de certificación. En cambio, la nueva extensión analiza el contenido de todos los certificados que conforman la cadena. Otro factor que marca la diferencia en la puntuación es el *cipher suite*, puesto que en la nueva extensión esta no obtiene una valoración de un 10. Si se observa la tabla 3.1, se puede comprobar que tan solo la versión SSL/TLS y el algoritmo de intercambio de clave reciben un 10.

Por otra parte, la diferencia en el algoritmo de puntuación es un factor muy importante que marca la valoración global. En el caso de *SSLeuth* hay parámetros que afectan directamente al cómputo final, por ejemplo *Extended Validation*. Pero, en la nueva extensión esta variable no afecta directamente a la puntuación final, y por lo tanto no suele influir tanto en esta.

Todos estos factores que marcan la puntuación que se le llegue a dar a una página Web se deben estudiar muy detenidamente realizando una gran cantidad de pruebas. Además, este estudio serviría de apoyo para diseñar un algoritmo de puntuación mucho más preciso. Aunque en ningún momento podría considerarse definitivo, puesto que se debe ir actualizando a medida que evoluciona la tecnología.

CAPÍTULO 4

POSIBLES MEJORAS

Una vez se ha acabado la implementación han surgido algunas ideas que podrían aplicarse en un futuro a la extensión que se ha diseñado, como por ejemplo más comprobaciones que puede realizar el algoritmo de puntuación, nuevas funcionalidades y una optimización del código. A continuación se detallarán cada una de estas mejoras que podrían implementarse en un futuro:

- Se podría comprobar que la firma de los certificados sea la correcta, puesto que en principio esta extensión confía en que dicha comprobación se haga correctamente por el navegador Firefox.
- También debería investigarse más si hay alguna manera de poder comprobar la información de revocación del certificado. Por una parte, se podría confiar en la información revocación de Firefox, y acceder a ella de alguna forma. Aunque se podría optar por realizar dicha consulta directamente desde la extensión, es decir, que esta realizase la petición al servidor OCSP.
- Podría introducirse una nueva funcionalidad a la extensión que permitiera analizar una gran cantidad de dominios de forma automática. Esto facilitaría la realización de estudios estadísticos con grandes volúmenes de información.
- Aunque este proyecto se ha enfocado en analizar las variables que se necesitan para construir un canal de comunicaciones seguro, podría ampliarse el proyecto analizando la página Web en general. En este caso deberían analizarse otros factores, como podrían ser los permisos que el usuario le ha concedido a la página Web.
- Se debería optimizar el código de la extensión, eliminando algunas partes que no se utilizan tras los cambios que se han realizado sobre esta. También podría implementarse un nuevo módulo que contenga las funciones que tienen en común los archivos *ssleuth.js* y *ssleuth-ui.js*.

CAPÍTULO 5

CONCLUSIONES

A lo largo de la realización de este proyecto se han podido extraer varias ideas clave referentes tanto a la seguridad de una conexión Web, como a la misma realización de este. Además, estas últimas han marcado el recorrido que debía tomar el proyecto y por ello es necesario destacarlas.

Se ha visto cuán importante es temporalizar un trabajo para poder desarrollarlo correctamente, puesto que a lo largo de este no se ha conseguido realizar todo lo deseado debido a que algunas tecnologías no están disponibles todavía. Por una parte, no ha sido posible construir un módulo de análisis que sea duradero a lo largo del tiempo, puesto que utiliza una tecnología que se está dejando ya de lado en el ámbito de los navegadores. Pero debido a ser demasiado pronto aún, no se encuentran las herramientas necesarias para poder implementar las funcionalidades de esta extensión con la nueva API (*WebExtensions*). Además, algunas funcionalidades que antes estaban disponibles pueden ya no estarlo o funcionar incorrectamente debido a la ausencia de soporte, como se ha podido observar con algunas funciones de la interfaz *nsIX509Cert*, que antes permitían forzar al navegador a realizar una petición OCSP.

No obstante, la extensión que se ha implementado es capaz de consultar casi toda la información de seguridad del navegador que se había planteado al inicio del trabajo (a excepción de la información de revocación). Por ello se puede utilizar para realizar estudios sobre qué parámetros son más relevantes en una conexión segura. Puede que dentro de unos pocos meses, cuando ya se hayan implementado las funciones necesarias dentro de *WebExtensions*, se puedan utilizar todos estos datos para diseñar una extensión duradera.

Por otro lado, se ha visto qué tan importante es la elaboración de una buena documentación a la hora de realizar un código. Ya que la documentación de la extensión no proporciona prácticamente ninguna indicación para poder analizar la extensión. Aunque ha servido para poder desarrollar nuevas técnicas de análisis que antes no se poseían, y que podrán ayudar en un futuro en casos similares.

Por lo referente a la seguridad de una conexión Web se ha observado qué tan difícil es definir un algoritmo correcto para valorarla, puesto que hay muchas variables que

influyen en esta. Además, a parte de esta gran cantidad de factores también es necesario definir correctamente el término "seguridad de una conexión Web". Es decir, se debe definir qué hace que una conexión Web sea más segura que otra y reflejarlo en un buen algoritmo.

Finalmente, cabe destacar la importancia que puede tener el desarrollo de una extensión de este tipo para concienciar a la gente sobre su seguridad en Internet. Puesto que es una herramienta fácilmente accesible por el usuario, y aunque proporcione datos que pueden ser muy técnicos, muestra al usuario una valoración global de la conexión. Además, esta extensión puede retroalimentarse de la información que esta reciba por parte de los usuarios, y así ir actualizándose constantemente.

BIBLIOGRAFÍA

[1] "Webextensions API," https://developer.mozilla.org/en-US/Add-ons/WebExtensions/API, [Online; accessed 8-June-2018]. 2.2.1

[2] "Webextensions: SSL(TLS) status API request," https://bugzilla.mozilla.org/show_bug.cgi?id=1322748, [Online; accessed 8-June-2018]. 2.2.1

[3] "Bootstrapped extensions," http://mdn.beonex.com/en/Extensions/Bootstrapped_extensions.html, [Online; accessed 8-June-2018]. 2.3.2

[4] "Interfaces XPCOM," https://developer.mozilla.org/es/docs/Mozilla/Tech/XUL/Tutorial_de_XUL/Interfaces_XPCOM, [Online; accessed 8-June-2018]. 2.3.2

[5] "QueryInterface," https://developer.mozilla.org/es/docs/nsISupports/QueryInterface, [Online; accessed 8-June-2018]. 2.4.3

[6] "Tabbrowser," https://developer.mozilla.org/en-US/docs/Mozilla/Tech/XUL/tabbrowser, [Online; accessed 8-June-2018]. 2.4.4

[7] "nsISSLStatusProvider interface reference," http://doxygen.db48x.net/mozilla-full/html/d5/de4/interfacensISSLStatusProvider.html, [Online; accessed 10-June-2018]. 2.4.4

[8] "nsIWebProgressListener," https://developer.mozilla.org/en-US/docs/Mozilla/Tech/XPCOM/Reference/Interface/nsIWebProgressListener, [Online; accessed 10-June-2018]. 2.4.4

[9] "nsISSLStatus.idl," https://dxr.mozilla.org/mozilla-esr45/source/security/manager/ssl/nsISSLStatus.idl, [Online; accessed 10-June-2018]. 2.4.4

[10] "nsIX509Cert interface reference," http://doxygen.db48x.net/mozilla-full/html/d3/d53/interfacensIX509Cert.html, [Online; accessed 10-June-2018]. 2.4.4

[11] "nsIASN1Tree.idl," https://dxr.mozilla.org/mozilla-beta/source/security/manager/pki/nsIASN1Tree.idl, [Online; accessed 11-June-2018]. 2.4.4

[12] "Remove nsIX509Cert.getUsagesArray, requestUsagesArrayAsync, and getUsagesString," https://bugzilla.mozilla.org/show_bug.cgi?id=1284946, [Online; accessed 10-June-2018]. 3.1

[13] "nsIX509CertDB.idl," https://dxr.mozilla.org/mozilla-beta/source/security/manager/ssl/nsIX509CertDB.idl, [Online; accessed 10-June-2018]. 3.1

[14] “About:config entries,” http://kb.mozillazine.org/About:config_entries, [Online; accessed 17-June-2018]. 3.1

[15] “security.ocsp.require - Why isn’t this enabled by default?” https://groups.google.com/forum/#!topic/mozilla.support.firefox/Gydr1AynQJ8, [Online; accessed 26-June-2018]. 3.1

[16] “Keylength - NIST report on criptographic key length and criptoperiod,” https://www.keylength.com/en/4/, [Online; accessed 26-June-2018]. 3.2