Facultad de INFORMÁTICA | UNIVERSIDAD NACIONAL DE LA PLATA

Especialización en Redes y Seguridad

Trabajo final integrador

Mejoras en la seguridad web del usuario mediante el uso de un proxy local

Tesista: Ing. Fernando A. Boettner
Director: Dr. Fernando G. Tinetti
Co-Director: Mg. Nicolás Macia

Agosto de 2022

ÍNDICE

Capítulo 1: Introducción

El objetivo de este trabajo es analizar el uso de un proxy local como herramienta utilizada para mitigar algunos ataques a los que usuarios de aplicaciones web están expuestos cuando las mismas tienen vulnerabilidades que pueden permitir el robo y/o suplantación de identidad. Mediante el uso de un proxy local podría ser posible la manipulación de las cookies intercambiadas con el usuario de tal forma que los ataques a los que la aplicación podría estar expuesta no afecten la sesión del usuario.

1.1 Objetivos

A los efectos de alcanzar esta meta se han definido los siguientes objetivos específicos:

1. Presentar los conceptos fundamentales del uso de cookies como mecanismo para mantener sesiones sobre el protocolo HTTP [1] y HTTPs [2].

2. Exponer las amenazas más frecuentes a las que los usuarios están expuestos ante fallas en el manejo de sesiones de una aplicación web.

3. Analizar las técnicas existentes para mitigar dichas amenazas.

4. Proponer un método para evitar el robo o suplantación de sesiones, a partir de la utilización de un servicio de proxy local.

1.2 Motivación

Como es sabido, HTTP es un protocolo que no maneja estados. Esto significa que cada conexión que genera un usuario desde un navegador hacia un servidor web es independiente. La técnica utilizada para que los servidores web puedan tener trazabilidad de las acciones que realizan los usuarios es el uso de *cookies* [3]. De hecho, gracias a esta técnica los servidores web logran mantener sesiones de usuario, a pesar de que cada petición proveniente del navegador web sea independiente de la anterior.

Las *cookies* son datos generalmente aleatorios que un servidor web envía a los navegadores como parte de los encabezados HTTP. Esos datos son almacenados localmente por los clientes, quienes vuelven a enviar el mismo dato al servidor en cada petición posterior. De esta forma, el servidor puede relacionar las distintas peticiones que le llegan.

En un proceso típico de identificación en un sitio web, el usuario envía sus credenciales de acceso, el servidor recibe estos datos y los valida. Si dicha validación es exitosa, entonces envía una respuesta al cliente incluyendo una *cookie*. El cliente almacena esa *cookie* y a partir de ese momento todas las peticiones que haga posteriormente incluirán ese dato. De

esa forma, el servidor podrá saber de qué usuario se trata y así otorgar acceso a las secciones y/o funcionalidades que corresponda.

En el caso de comunicaciones seguras, utilizando HTTPs, el tráfico entre el navegador y el servidor web está cifrado utilizando SSL [4] o TLS [5], de esta forma, un atacante que tuviera acceso al canal de comunicaciones no podría descifrar el contenido de dicho tráfico. Sin embargo, el hecho de cifrar el canal de comunicación entre el cliente y el servidor no resuelve el problema de la identificación de usuarios, y por lo tanto el concepto de *cookies* se aplica de igual manera que con HTTP.

Si bien el manejo de sesiones de usuario utilizando cookies funciona correctamente, la privacidad de la navegación puede verse comprometida por el robo de las *cookies*. Existen distintos mecanismos que permiten a los atacantes obtener las *cookies* que se intercambian entre navegadores y servidores web. Por un lado, técnicas basadas en la obtención de *cookies* mediante escucha o *sniffing* de la red, y por el otro, mecanismos para obtener dichas cookies directamente desde los navegadores. En ambos casos, los atacantes pueden lograr la suplantación de identidad, robo de sesiones o incluso denegaciones de servicio.

OWASP [6] (Open Web Application Security Project) es un proyecto de código abierto dedicado a determinar y combatir las causas que hacen que el software sea inseguro. La comunidad OWASP está formada por empresas, organizaciones educativas y particulares de todo el mundo. Juntos constituyen una comunidad de seguridad informática que trabaja para crear artículos, metodologías, documentación, herramientas y tecnologías que se liberan y pueden ser usadas gratuitamente por cualquiera.

Uno de los documentos más importantes y extendidos de OWASP es el denominado *OWASP Top Ten* [7], donde se detallan las diez vulnerabilidades más riesgosas de las aplicaciones web. En particular, las vulnerabilidades del Top Ten (versión año 2021) mencionadas a continuación hacen referencia a la problemática del robo de *cookies*:

A02: Fallas criptográficas *(en 2017, categorizado como **A3**: Exposición de datos sensibles)*
Este riesgo hace referencia al robo de datos ya sea desde el servidor, en tránsito o del cliente. Entre esos datos sensibles están las cookies.

*“Un atacante monitorea el tráfico de la red (por ejemplo, en una red inalámbrica insegura), degrada las conexiones de HTTPS a HTTP, intercepta solicitudes y **roba la cookie** de sesión del usuario. Luego, el atacante utiliza esta cookie y secuestra la sesión del usuario (ya autenticado), accediendo o modificando los datos privados, incluso podría alterar los datos enviados”.*

A03: Inyección *(en 2017, categorizado como **A7**: Cross-Site Scripting)*
Los ataques XSS (Cross-Site Scripting) permiten que un atacante pueda ejecutar comandos arbitrarios en el navegador de la víctima. Por lo general esto ocurre por falta de validaciones de datos en la aplicación.

“Un atacante puede usar XSS para enviar un script malicioso a un usuario desprevenido. El navegador del usuario final no tiene forma de saber que no se debe confiar en el script y lo ejecutará. Debido a que cree que el script proviene de una fuente confiable, el script malicioso puede acceder a cualquier ***cookie****, token de sesión u otra información confidencial retenida por el navegador y utilizada en este sitio”.*

La presente propuesta plantea el análisis de la utilización de un **proxy local** como herramienta para mitigar el riesgo del robo de las *cookies* de los navegadores web a partir de vulnerabilidades de tipo XSS, y puede considerarse relacionada con la temática de [8]. En particular, en [8] se propone un mecanismo alternativo para el manejo de sesiones web en el que no se hace necesario el intercambio de cookies entre los navegadores y los servidores web, aprovechando el concepto de conexiones persistentes incluido en HTTP 1.1.

En este trabajo se propone el uso de un proxy local, que permita la manipulación de cabeceras HTTP, pudiendo así alterar los valores de las cookies. De ese modo, aun si las cookies fueran robadas de los navegadores web de los clientes, no serían de utilidad para los atacantes.

En el capítulo 2 se presentan los fundamentos del protocolo HTTP y su evolución. Se explica el funcionamiento del mecanismo de sesiones de usuario, mediante el uso de cookies, y finalmente se agrega una breve reseña sobre el funcionamiento de HTTPS y su relación con el manejo de sesiones web.

El capítulo 3 detalla distintos tipos de amenazas a la seguridad web, en particular, aquellas relacionadas con el manejo de sesiones, junto con las técnicas de mitigación empleadas para cada tipo de ataque.

En el capítulo 4 se presenta una alternativa para mitigar el riesgo del robo de las *cookies* de los navegadores web, mediante la utilización de un proxy local. Se detalla el proceso de instalación y configuración de un proxy, y se realizan distintas pruebas de concepto para probar la factibilidad de manipular las cabeceras HTTP, a fin de alterar las cookies intercambiadas entre los navegadores y los servidores web.

Por último, en el capítulo 5 se presentan las conclusiones del trabajo y se mencionan algunas posibles líneas de investigación para futuros trabajos relacionados.

Capítulo 2: El protocolo HTTP

El Protocolo de transferencia de hipertexto (HTTP) es uno de los protocolos de aplicación más difundidos y ampliamente utilizados en Internet: es el lenguaje común entre clientes y servidores, lo que permite la web moderna. Se basa, principalmente, en interacciones de petición-respuesta, entre clientes y servidores web. Desde sus inicios, se ha convertido en el protocolo preferido para el intercambio de información en la World Wide Web. Con el paso del tiempo, surgieron adaptaciones y mejoras al protocolo original [9], para cubrir las demandas de las aplicaciones web modernas.

2.1 Versiones del protocolo

HTTP/0.9 – El protocolo de una sola línea

La versión inicial de HTTP, no tenía número de versión; aunque posteriormente se la denominó como 0.9 para distinguirla de las versiones siguientes. HTTP/0.9 es un protocolo extremadamente sencillo: una petición consiste simplemente en una única línea, que comienza por el único método posible **GET**, seguido por la dirección del recurso a pedir:

```
GET /miPaginaWeb.html
```

La respuesta también es muy sencilla, solamente consiste el archivo pedido:

```
<HTML>
Una pagina web muy sencilla
</HTML>
```

Al contrario que sus posteriores evoluciones, el protocolo HTTP/0.9 no usa cabeceras HTTP, con lo cual únicamente es posible transmitir archivos HTML, y ningún otro tipo de archivos. Tampoco había información del estado ni códigos de error: en el caso de un problema, el archivo HTML pedido, era devuelto con una descripción del problema dentro de él, para que una persona pudiera analizarlo.

HTTP/1.0 – Desarrollando expansibilidad

La versión HTTP/0.9 era muy limitada y tanto los navegadores como los servidores, pronto ampliaron el protocolo para que fuera más flexible. Los puntos más destacados son:

- La versión del protocolo se envía con cada petición: HTTP/1.0 se añade a la línea de la petición GET.

- Se envía también un código de estado al comienzo de la respuesta, permitiendo así que el navegador pueda responder al éxito o fracaso de la petición realizada.
- El concepto de cabeceras de HTTP, se presentó tanto para las peticiones como para las respuestas, permitiendo la transmisión de meta datos y conformando un protocolo muy versátil y ampliable.
- Con el uso de las cabeceras de HTTP, se pudieron transmitir otros documentos además de HTML, mediante la cabecera *Content-Type*.

En el siguiente ejemplo se pueden ver dos casos de petición-respuesta, desde un cliente hacia un servidor; la primera es una petición de contenido HTML tradicional, y la segunda una imagen:

```
GET /mypage.html HTTP/1.0
User-Agent: NCSA_Mosaic/2.0 (Windows 3.1)

200 OK
Date: Tue, 15 Nov 1994 08:12:31 GMT
Server: CERN/3.0 libwww/2.17
Content-Type: text/html
<HTML>
Una pagina web con una imagen
  <IMG SRC="/miImagen.gif">
</HTML>
```

```
GET /myImagen.gif HTTP/1.0
User-Agent: NCSA_Mosaic/2.0 (Windows 3.1)

200 OK
Date: Tue, 15 Nov 1994 08:12:32 GMT
Server: CERN/3.0 libwww/2.17
Content-Type: text/gif
(image content)
```

Estas innovaciones, no se desarrollaron de forma planeada, sino más bien con una aproximación de prueba y error, entre los años 1991 y 1995: un servidor y un navegador, añadían una nueva funcionalidad y se evaluaba su aceptación. Debido a esto, en ese periodo eran muy comunes los problemas de interoperabilidad. En noviembre de 1996, para poner fin a estos problemas se publicó un documento informativo que describía las prácticas adecuadas, la RFC 1945. Este documento es la definición del protocolo HTTP/1.0. Resulta curioso, que realmente no es un estándar oficial.

HTTP/1.1 – El protocolo estándar

Un año antes de que se publicara el documento de HTTP/1.0, se comenzó a trabajar en un proceso de estandarización formal de una nueva versión: HTTP/1.1, que finalmente se publicó en enero de 1997, solo unos meses después del HTTP/1.0.

HTTP/1.1 aclaró ambigüedades y añadió numerosas mejoras:

- Una conexión podía ser reutilizada, ahorrando así el tiempo de reabrirla repetidas veces para mostrar los recursos empotrados dentro del documento original pedido.
- Enrutamiento ('Pipelining' en inglés) se añadió a la especificación, permitiendo realizar una segunda petición de datos, antes de que fuera respondida la primera, disminuyendo de este modo la latencia de la comunicación.
- Se permitió que las respuestas a peticiones puedan dividirse en fragmentos.
- Se añadieron controles adicionales a los mecanismos de gestión de la caché.
- La negociación de contenido, incluyendo el lenguaje, el tipo de codificación, o tipos, se añadieron a la especificación, permitiendo que servidor y cliente, acordasen el contenido más adecuado a intercambiarse.
- Gracias a la cabecera *Host*, pudo ser posible alojar varios dominios en la misma dirección IP.

Fue publicado inicialmente como RFC 2068, pero gracias a la capacidad de crear cabeceras en forma sencilla se realizaron actualizaciones y revisiones posteriores: RFC 2616, publicado en Junio de 1999 y posteriormente en los documentos RFC 7230 y RFC 7235 publicados en Junio del 2014. En definitiva, el protocolo HTTP/1.1 ha sido increíblemente estable durante más de 15 años.

Se muestra a continuación un ejemplo de dos peticiones dentro de la misma conexión, que en definitiva es una de las grandes mejoras que incluyó este protocolo, logrando minimizar la latencia en las comunicaciones:

```
GET /en-US/docs/Glossary/Simple_header HTTP/1.1
Host: developer.mozilla.org
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:50.0) Gecko/20100101 Firefox/50.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://developer.mozilla.org/en-US/docs/Glossary/Simple_header

200 OK
Connection: Keep-Alive
Content-Encoding: gzip
Content-Type: text/html; charset=utf-8
Date: Wed, 20 Jul 2016 10:55:30 GMT
Etag: "547fa7e369ef56031dd3bff2ace9fc0832eb251a"
Keep-Alive: timeout=5, max=1000
Last-Modified: Tue, 19 Jul 2016 00:59:33 GMT
Server: Apache
Transfer-Encoding: chunked
Vary: Cookie, Accept-Encoding

(...contenido...)


GET /static/img/header-background.png HTTP/1.1
Host: developer.cdn.mozilla.net
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:50.0) Gecko/20100101 Firefox/50.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://developer.mozilla.org/en-US/docs/Glossary/Simple_header

200 OK
Age: 9578461
Cache-Control: public, max-age=315360000
Connection: keep-alive
Content-Length: 3077
Content-Type: image/png
Date: Thu, 31 Mar 2016 13:34:46 GMT
Last-Modified: Wed, 21 Oct 2015 18:27:50 GMT
Server: Apache

(image content of 3077 bytes)
```

HTTP/2 – El protocolo para un mayor rendimiento

A lo largo de los años, las páginas Web han llegado a ser mucho más complejas, incluso llegando a poder considerarse como aplicaciones por derecho propio. La cantidad de contenido visual, el tamaño de los scripts, y los scripts que añaden interactividad ha aumentado mucho también.

Por esta razón, hace más de una década, se comenzó a trabajar en una nueva versión del protocolo HTTP, con el objetivo principal de mejorar la performance. Se trata de un desarrollo de la empresa Google denominado SPDY, que sirvió de base para el desarrollo de HTTP/2. Este protocolo tiene notables diferencias fundamentales respecto a la versión anterior:

- Es un protocolo binario, en contraposición a estar formado por cadenas de texto, tal y como estaban basados sus protocolos anteriores. Si bien con este cambio ya no se puede leer ni crear directamente el contenido, se pueden utilizar técnicas de optimización.
- Es un protocolo multiplexado: Se pueden realizar peticiones en paralelo sobre la misma conexión, por lo que no está sujeto a mantener el orden de los mensajes, ni otras restricciones que tenían los protocolos anteriores HTTP/1.x.
- Comprime las cabeceras, ya que éstas normalmente son similares en un grupo de peticiones. Esto elimina la duplicación y retardo en los datos a transmitir.
- Esto permite al servidor almacenar datos en la caché del cliente, previamente a que estos sean pedidos, mediante un mecanismo denominado *server push*.

Estandarizado de manera oficial en Mayo de 2015 (RFC 7540), HTTP/2 ha conseguido mucho éxito. En la actualidad, aproximadamente el 45% de los sitios web ya utilizan este protocolo [10]. Los sitios web con mucho tráfico fueron aquellos que lo adoptaron más rápidamente, ahorrando considerablemente las sobrecargas en la transferencia de datos.

Esta rápida adopción era esperada, ya que el uso de HTTP/2 no requiere de una adaptación de los sitios Web y aplicaciones: el uso de HTTP/1.1 o HTTP/2 es transparente para ellos. El uso de un servidor actual, comunicándose con un navegador actualizado, es suficiente para permitir su uso: únicamente en casos particulares fue necesario impulsar su utilización; y según se actualizan servidores y navegadores antiguos, su utilización aumenta, sin que requiera un mayor esfuerzo de los desarrolladores Web.

2.2 Sesiones de usuario

Como se mencionó anteriormente, HTTP es un protocolo que no maneja estados (*stateless*, en inglés). Esto significa que cada petición que genera un usuario desde un navegador hacia un servidor web es independiente de la anterior, es decir, en los mensajes intercambiados no hay información que permita al servidor determinar una correlación entre dichas peticiones.

Ahora bien, en el apartado anterior, se menciona que HTTP/1.1 incluye la posibilidad de enviar múltiples conexiones dentro de la misma conexión TCP. Sin embargo, esas conexiones TCP solo se mantienen abiertas unos pocos segundos [11], por lo que se sigue considerando a HTTP/1.1 y versiones posteriores como protocolos sin estados.

Las sesiones de usuario hacen referencia al conjunto de peticiones interrelacionadas. Pero, ¿cómo podría saber un servidor web que determinada conexión corresponde al mismo usuario que ya había ingresado previamente al sistema? Considerando que el protocolo HTTP en sí no ofrece esta posibilidad, entonces se idearon distintos mecanismos para lograr

esa trazabilidad en las peticiones. El mecanismo más difundido y utilizado desde los comienzos de la World Wide Web hasta la actualidad es el intercambio de cookies [12].

Las cookies fueron desarrolladas por primera vez en 1994 por Lou Montulli [13], un empleado de Netscape Communications, junto con John Giannandrea [14]. La primera aplicación real de cookies en la web fue determinar si los visitantes del sitio web de Netscape habían estado allí antes, y posteriormente se usó el mismo concepto para el desarrollo de portales de comercio electrónico. Hoy día, hay innumerables aplicaciones web que requieren del concepto de sesión para permitir a los usuarios realizar operaciones, por ejemplo, sitios de banca electrónica, redes sociales, y portales en general que requieren la identificación de los usuarios que los acceden.

Las *cookies* son datos generalmente aleatorios que un servidor web envía a los navegadores como parte de los encabezados HTTP. Esos datos son almacenados localmente por los clientes, quienes vuelven a enviar el mismo dato al servidor en cada petición posterior. De esta forma, el servidor puede relacionar las distintas peticiones que le llegan.

Tiempo de vida de una cookie

Además del nombre y valor asociado de una cookie, también se pueden adicionar parámetros o directivas que determinan el comportamiento que tendrán clientes y servidores al intercambiar dicha cookie. Las directivas ***Expires*** y ***Max-Age*** indican el tiempo de validez de una cookie, en el primer caso mediante una fecha y hora, y en el segundo caso con una cantidad específica de milisegundos.

```
Set-Cookie: id=a3fWa; Expires=Wed, 21 Oct 2015 07:28:00 GMT;
```

Si estas directivas no son especificadas por el servidor, entonces la cookie sólo tendrá validez mientras permanezca abierto el navegador web. Este tipo de cookies se denominan **cookies temporales**. En caso de que un servidor necesite que se descarte una cookie, por ejemplo ante el cierre de una sesión de usuario, entonces simplemente deberá volver a enviar la cookie al navegador web, pero indicando una fecha de expiración anterior a la fecha actual.

Actualmente, la mayoría de los sitios web que ofrecen las entidades bancarias o financieras utilizan cookies para identificar las sesiones de usuarios, con un tiempo de vida de unos pocos minutos. Mientras el usuario continúa operando, dichas cookies se van actualizando extendiendo su validez, pero ante la falta de actividad, las cookies expiran y por tal razón el usuario es llevado nuevamente a la página de bienvenida, invitándolo a que vuelva a identificarse.

Por otra parte, sitios como Facebook, LinkedIn, Instagram, entre otros, utilizan cookies con una validez de varios meses, incluso años. Eso permite que un usuario pueda utilizar estos servicios sin necesidad de identificarse una y otra vez al acceder a dichos sitios.

2.3 Agregando Seguridad: HTTPS

Hypertext Transfer Protocol Secure (HTTPS) es una extensión de HTTP que se utiliza para proveer una comunicación segura entre los servidores y los clientes web, mediante el uso de una capa adicional para cifrar el tráfico del canal de comunicación mediante TLS (Transport Layer Security), o su predecesor SSL (Secure Sockets Layer).

La motivación principal detrás del uso de HTTPS es lograr la privacidad e integridad de los datos mientras viajan por la red entre un navegador y un servidor web. De esta forma, la información queda protegida ante eventuales intentos de intrusión con técnicas de “escucha” (sniffing) de la red.

La seguridad de HTTPS básicamente se basa en el uso de criptografía asimétrica [15] para el intercambio de una clave de sesión entre el cliente y el servidor. Una vez intercambiada esa clave, tanto el cliente como el servidor enviarán la información encriptada con dicha clave.

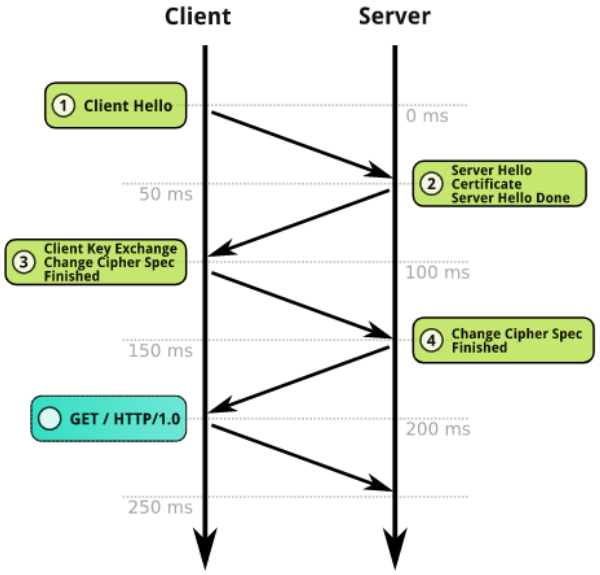
Funcionamiento de HTTPS

Cuando un navegador inicia una conexión, el servidor web envía una lista de versiones de SSL / TLS compatibles que se hayan configurado, y un conjunto de algoritmos de cifrado. Por otra parte, el navegador web también envía su respectiva lista también. Entonces, se elige la mejor versión de los protocolos SSL / TLS y el algoritmo de cifrado según las preferencias del servidor.

El servidor responderá al navegador web enviando su certificado digital [16], que incluye su clave pública para que pueda verificar la identidad del servidor web. El navegador comprueba que el certificado enviado por el servidor sea auténtico, y, entonces, el navegador genera una clave maestra para que tanto navegador como servidor puedan usarla más tarde. Esta clave maestra es enviada al servidor usando la clave pública del servidor para cifrarla, y luego, es descifrada por el servidor usando su clave privada. Con esta clave maestra, tanto el cliente como el servidor podrán utilizar criptografía simétrica [22] para el intercambio de datos, quedando asegurados para el resto de la sesión.

Debido a que HTTPS utiliza una capa adicional para brindar seguridad (TLS), se puede cifrar la totalidad del protocolo HTTP subyacente. Esto incluye la URL solicitada (qué página web en particular se solicitó), parámetros de consulta, encabezados y **cookies** (que a menudo contienen información de identidad sobre el usuario).

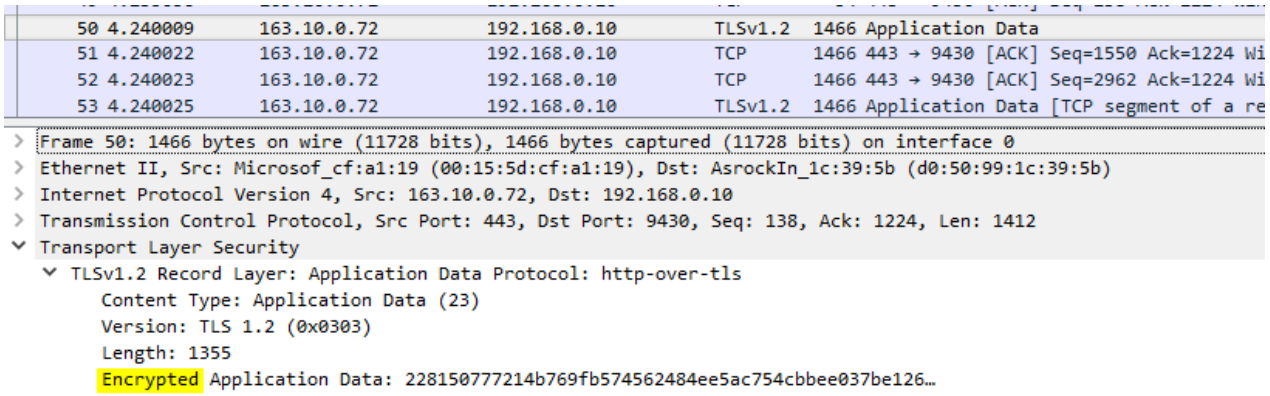
La utilización de HTTPS es especialmente importante en redes no protegidas (como los puntos de acceso público a Wi-Fi), ya que cualquier persona en la misma red local puede rastrear paquetes y descubrir información confidencial no protegida por HTTPS.



Full TLS handshake

La siguiente captura se realizó con el software Wireshark [17], donde se puede apreciar el intercambio de mensajes realizados al acceder al sitio web www.unlp.edu.ar.

Nótese que después del intercambio inicial (TLS Handshake), la información enviada desde el servidor hacia el cliente ya se encuentra encriptada.



Ahora bien, ¿qué significa el hecho de que HTTPS proteja el intercambio de información, incluyendo las cookies? Recordemos una de las vulnerabilidades más críticas según OWASP:

A02: Fallas criptográficas
*"Un atacante monitorea el tráfico de la red (por ejemplo, en una red inalámbrica insegura), degrada las conexiones de HTTPS a HTTP, intercepta solicitudes y **roba la cookie** de sesión del usuario. Luego, el atacante utiliza esta cookie y secuestra la sesión del usuario (ya autenticado), accediendo o modificando los datos privados, incluso podría alterar los datos enviados".*

El uso de HTTPs **no impide** que un atacante pueda hacerse de las cookies, ya sea tomando el control del servidor o robándolas desde los almacenes de datos de los navegadores web. No obstante, usando HTTPs sí evitamos la intercepción de las cookies **en tránsito**, es decir, usando técnicas para capturar tráfico de red (*sniffing*, en inglés).

Los ataques XSS (Cross-Site Scripting), también referenciados por OWASP, son métodos usados para robar cookies desde los navegadores, y por lo tanto, pueden llevarse a cabo independientemente de si se usa o no HTTPs.

En los siguientes capítulos del presente trabajo se abordarán este tipo de ataques en concreto y se planteará un mecanismo para mitigar los riesgos asociados al robo de cookies de los navegadores, ofreciendo así mayor seguridad a los usuarios en el uso de aplicaciones web.

Capítulo 3: Amenazas a la seguridad web en el manejo de sesiones

Las aplicaciones web pueden verse expuestas a distintos ataques que afectan la seguridad y privacidad de los usuarios, como también la integridad de la información. Estos ataques se diseñan con objetivos diversos, por ejemplo:

- Acceder a una aplicación web para obtener y/o modificar información.
- Generar una denegación de servicio, es decir, evitar que los usuarios puedan acceder.
- Acceder al servidor que aloja la aplicación web sólo como un paso intermedio para ganar acceso a la red de la organización.
- Acceder a una aplicación web haciéndose pasar por un usuario legítimo.

A los efectos del presente trabajo, nos interesan los ataques relacionados al manejo de sesiones web, es decir, el robo de sesiones y por ende, la identidad de los usuarios.

El robo de una sesión web se refiere a la posibilidad de que un atacante pueda usar la aplicación web como si fuese el usuario víctima, sin conocer sus credenciales de acceso. Dado que una sesión web se basa en la utilización de cookies, si un atacante logra robar las cookies de sesión intercambiadas entre el navegador de la víctima y el servidor web, entonces podría acceder a la aplicación web usando esas mismas cookies y por lo tanto acceder a la sesión del usuario.

Existen dos términos muy utilizados cuando se habla de ataques de robo de sesión: **Secuestro de sesión** (*Session Hijacking*) y **Suplantación de sesión** (*Session Spoofing*). Si bien están estrechamente relacionados, el secuestro y la suplantación de identidad difieren en el momento del ataque. El secuestro de sesión se realiza contra un usuario que actualmente está conectado y autenticado, por lo que, desde el punto de vista de la víctima, el ataque a menudo hará que la aplicación objetivo se comporte de manera impredecible o se bloquee. Con la suplantación de sesión, en cambio, los atacantes usan tokens de sesión robados o falsificados para iniciar una nueva sesión y hacerse pasar por el usuario original, que podría no estar al tanto del ataque. A continuación, se describen las técnicas principales para llevar a cabo este tipo de ataques.

3.1 Cross-site scripting (XSS)

Se trata de una de las técnicas más usadas para el secuestro de sesiones y consiste básicamente en la inyección de código en sitios o aplicaciones vulnerables, logrando que el navegador web del cliente ejecute dicho código al cargar una determinada página. De esta forma, un atacante podría aprovecharse de un sitio web vulnerable e inyectar código que permita obtener los datos de las cookies de sesión de un sitio web, pudiendo entonces realizar un secuestro de sesión. Se distinguen tres tipos de ataques XSS:

XSS Reflejado

Cuando una aplicación web solicita datos mediante el método GET y los mismos no son debidamente validados, resulta posible inyectar código malicioso. Ese código no se almacena en el servidor, sino que se “refleja” como parte de la respuesta del servidor web. Un atacante puede construir una URL inyectando el código malicioso y hacerla llegar a la víctima, por ejemplo, vía email. Cuando se acceda a dicha URL en el navegador de la víctima, se ejecutará el código malicioso. Por ejemplo, el siguiente código Javascript con AJAX permitiría llevar a cabo un ataque de este tipo:

```
var cookiesDeUsuario = document.cookie;

var xhr = new XMLHttpRequest(); // Objeto Ajax
xhr.open('GET', 'www.servidor-atacante.com/cookies.php');
xhr.send('c=' + cookiesDeUsuario);
```

Cuando este código sea ejecutado por el navegador de la víctima, se estarán enviando las cookies al servidor del atacante.

XSS Almacenado

A diferencia del método anterior, donde el código malicioso inyectado se refleja como parte de la respuesta del servidor, en este caso la aplicación almacena los datos proporcionados por el usuario, por lo tanto, si esos datos no están debidamente validados y un atacante logra enviar código malicioso, el mismo quedará almacenado y eventualmente otros usuarios podrían ejecutar dicho código sin saberlo. Una página web sería vulnerable a este tipo de XSS si por ejemplo un usuario ingresa código Javascript en datos propios, como un teléfono o domicilio, para que dicho código se ejecute posteriormente cuando un empleado o administrador accede al perfil de dicho usuario. Los sitios como blogs o foros son los candidatos ideales para este tipo de ataques, ya que si se logra inyectar código en un posteo, todos los lectores del mismo ejecutarán dicho código con solo acceder al contenido. El código presentado anteriormente podría ser utilizado también en un ataque de XSS Almacenado, logrando que cada visitante del sitio “entregue” involuntariamente las cookies al atacante.

XSS Basados en DOM

En este caso, al igual que los anteriores, el ataque se puede llevar a cabo por la falta de validación de los datos que un usuario envía al servidor web. La diferencia de este tipo de XSS con el XSS reflejado es que si miramos el código no veremos el JavaScript malicioso directamente en el HTML que forma parte de la respuesta del servidor, ya que todo sucede en el DOM [18].

Una técnica usual es la distribución de correos electrónicos que posean un link que apunte a un sitio web válido pero incluyendo parámetros que logren inyectar el código malicioso, incluso ofuscando dicho código para que no resulte evidente la manipulación del link.

La técnica por excelencia para evitar los ataques de tipo XSS, consiste en la validación de los datos recibidos por el servidor desde los navegadores de los usuarios. Muchos frameworks de desarrollo de software, incluyen técnicas de codificación y sanitización que ayudan a los desarrolladores a generar código no vulnerable a este tipo de ataques. No obstante, es fundamental que quienes desarrollan aplicaciones web se mantengan actualizados en este tipo de problemáticas. La fundación OWASP ofrece una guía [24] para la prevención de XSS.

3.2 Session Side Jacking

Se denominan de esta forma a las técnicas que permiten a los atacantes interceptar las cookies de sesión mediante la escucha (sniffing) del tráfico de la red de la víctima. Si un atacante logra hacerse de una cookie de sesión, podrá usarla para acceder al sitio web destino como si fuese el usuario víctima, sin necesidad de conocer las credenciales de acceso.

Lógicamente, para poder llevar a cabo este tipo de ataques, se requiere contar con acceso a la red de la víctima, y dependiendo de la topología de la red puede ser más o menos complejo efectuar la escucha de tráfico:

3.2.1 Sniffing en redes interconectadas con hubs y/o puntos de acceso inalámbricos

En este caso, el ataque es muy sencillo, solo se necesita un analizador de protocolos, como por ejemplo Wireshark, que se ejecute en un equipo conectado a la misma red de la víctima. Esto permitirá que el atacante pueda “ver” el tráfico intercambiado entre el navegador de la

víctima y el servidor web. Por esta razón, es habitual encontrar recomendaciones sobre la no utilización de redes Wi-fi no conocidas, por ejemplo, en aeropuertos, donde podrían efectuarse este tipo de ataques sin demasiadas complicaciones.

3.2.2 Sniffing en redes interconectadas mediante switches

En este caso, dado que las las tramas unicast que recibe el switch solo son enviadas hacia el puerto de destino, no resulta posible realizar una escucha de tráfico en forma directa. Sin embargo, hay algunas posibilidades a tener en cuenta:

Port mirroring / Port span

La mayoría de los fabricantes de switches ofrecen una funcionalidad que permite copiar las tramas de datos dirigidas hacia uno o más puertos, en otro puerto. El objetivo es tener una herramienta de diagnóstico ante algún problema que requiera análisis detallado a nivel de red. Por lo tanto, esta misma funcionalidad podría ser utilizada de manera maliciosa para realizar sniffing. Resulta evidente que, en este caso, el atacante debe tener acceso físico y lógico al switch, para poder conectar un equipo a un puerto previamente configurado como span o mirror.

MAC Flooding

Otra posibilidad es realizar un primer ataque llamado MAC Flooding, para luego poder realizar escuchas en la red. Se trata de una técnica que básicamente consiste en enviar una gran cantidad de tramas Ethernet a los switches, con el objetivo de agotar la capacidad de sus tablas internas (denominadas TCAM). Al llenarse esas tablas, el equipo comenzaría a realizar broadcasting, es decir, a enviar las tramas a todos los puertos del equipo, y por lo tanto, se haría factible llevar a cabo el sniffing. Algunos fabricantes ofrecen herramientas de protección para MAC Flooding, por ejemplo la funcionalidad Port Security, de Cisco [19].

Man in the Middle

Debido a que en la actualidad prácticamente no se utilizan hubs y que la técnica de MAC Flooding no siempre puede llevarse a cabo con éxito, para lograr “escuchar” el tráfico de la red, los atacantes deben realizar primero otro ataque, denominado MITM (Man-in-the-middle). Se conoce con este nombre (MITM) a las técnicas para interceptar una comunicación de red, sin que las partes estén al tanto de esto. Existen distintas formas de llevar a cabo estos ataques:

ARP Spoofing

El objetivo del protocolo ARP es permitir a un dispositivo conectado a una red LAN obtener la dirección MAC de otro dispositivo conectado a la misma red LAN cuya dirección IP es conocida. Este protocolo está documentado en la RFC 826.

La suplantación de ARP (o ARP spoofing) es una técnica que consiste en enviar mensajes ARP falsos, con la finalidad de asociar la dirección MAC del atacante con la dirección IP del host atacado, normalmente el default gateway. De esta forma, cualquier tráfico dirigido a la dirección IP de ese host, será enviado al atacante, en lugar de a su destino real. El atacante, entonces, puede capturar el tráfico y posteriormente enviarlo al destino real. Incluso podría alterar los datos antes de enviarlos.

ICMP Redirect

La funcionalidad ICMP Redirect, detallada en la RFC 792, permite que un router pueda indicarle a un host, que existe un camino más corto para alcanzar un destino. El caso típico es el de dos routers conectados a la misma red LAN. Si un host tiene como default gateway el Router 1, y pretende acceder a una red que está detrás del Router 2, entonces el Router 1 enviará un mensaje ICMP Redirect al host para que las peticiones posteriores se encaminen directamente hacia el Router 2. Los ataques MITM basados en ICMP Redirect consisten en el envío de este tipo de mensajes con información falsa, para que los paquetes sean destinados hacia el host del atacante.

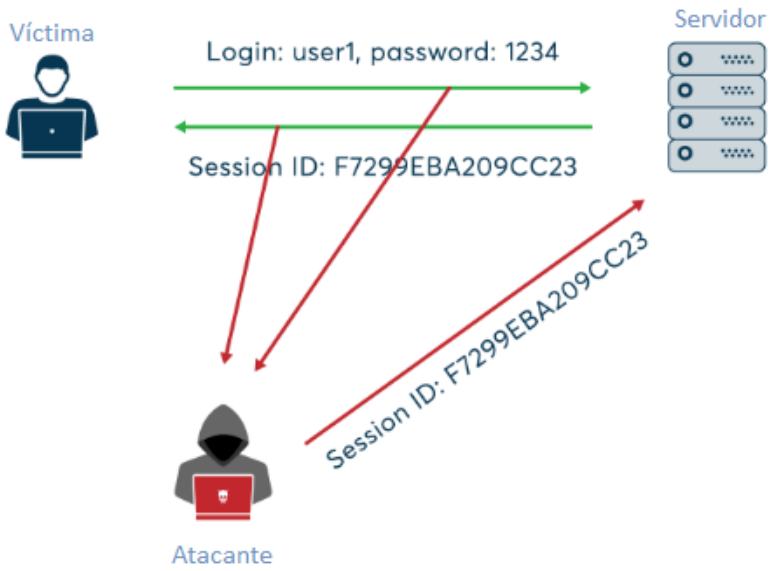
DHCP Spoofing

En una red que utiliza DHCP, cuando un host requiere una dirección IP envía un broadcast, para que un servidor DHCP reciba el mensaje y posteriormente le responda con la dirección IP asignada a dicho host. El ataque MITM basado en DHCP consiste en conectar un servidor DHCP falso en la red, que responda las peticiones de los clientes con datos de Gateway y DNS modificados, para que el tráfico sea dirigido a equipos del atacante.

DNS Spoofing

Este tipo de ataques consisten en explotar vulnerabilidades de los servidores de DNS, para poder modificar sus cachés con información falsa. De esta forma, un servidor DNS corporativo que fue atacado, podría entregar direcciones falsas para ciertos dominios a todos los hosts de la red que lo soliciten. Una vez más, ese tráfico sería dirigido a hosts del atacante, que luego de interceptarlo, podría reenviarlo a los destinos reales, modificarlo, o incluso generar una denegación de servicio.

Todas estas técnicas persiguen el mismo objetivo: capturar tráfico. En el caso de sesiones web, el atacante podría robar las cookies intercambiadas entre clientes y servidores y de esa forma llevar a cabo el secuestro de la sesión.



3.2.3 Sniffing en sitios seguros (HTTPS)

En el caso de sitios web que funcionen bajo HTTPS, la escucha de tráfico sería inútil, ya que aún en el caso de que un atacante lograra capturar el tráfico, no podría interpretarse, por estar cifrado. No obstante, muchos sitios web utilizan la estrategia de cifrar únicamente el tráfico de la página de acceso al sistema (login), dejando al resto del sitio sobre HTTP. Esto era algo muy común hace algunos años, y si bien a priori resulta razonable, pues el único momento en que un usuario ingresa sus credenciales de acceso es en el proceso de login, ¿qué ocurre con las cookies?

Una vez que el usuario está autenticado, las cookies son intercambiadas durante el resto de la sesión, y si dicho tráfico fuese capturado por un atacante, entonces podría llevar a cabo el secuestro de dicha sesión. En la actualidad, cada vez más sitios web emplean HTTPS en todas las páginas que componen la aplicación.

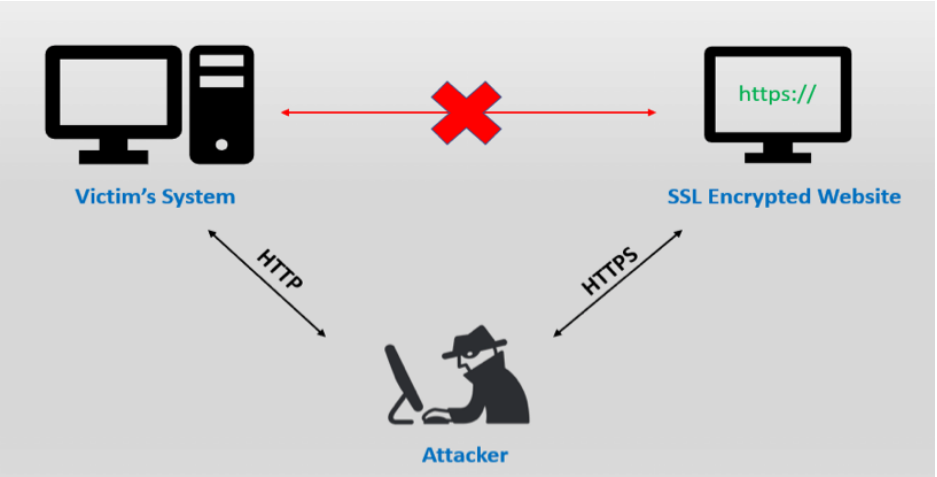
SSL Strip

Esta técnica tiene como objetivo quitar la capa de seguridad a las conexiones HTTPS, aprovechando el primer intento de conexión a un sitio web.

Generalmente, un usuario tipea en su navegador web una dirección, por ejemplo “www.sitio.com”. Eso hace que se intente establecer una conexión HTTP. El servidor, al recibir esa petición, envía una respuesta de redirección (Respuesta 301 o 302), para convertir esa petición a “https://www.sitio.com”. Y por lo tanto, a partir de ese momento, la conexión queda cifrada.

En esa primera conexión es donde entra en juego SSL Strip, que básicamente se ocupa de convertir cada petición HTTPS de la víctima hacia HTTP, pudiendo así capturar el tráfico. El servidor web no se entera de esto, ya que la conexión es realmente originada desde la

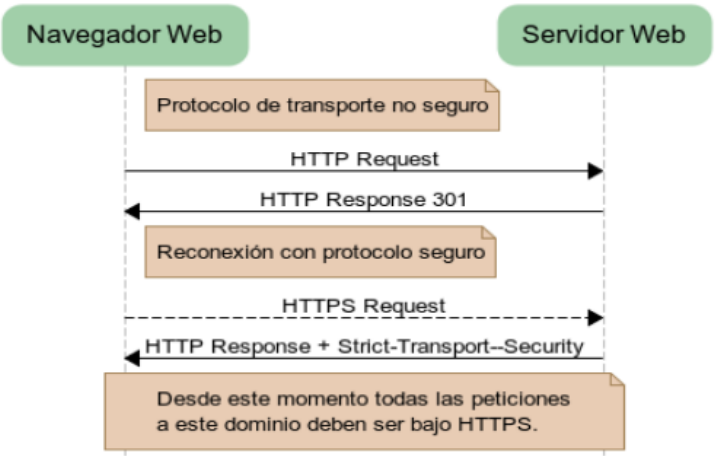
máquina del atacante, usando HTTPS, pero entre ésta y la víctima se realiza vía HTTP. Desde luego, para poder llevar a cabo este tipo de ataques, se necesita, además, de alguna de las técnicas de MITM mencionadas anteriormente, para poder tomar control del tráfico de la red.



HSTS

Para hacer frente a este tipo de ataques, surge en el año 2012 la especificación HSTS, mediante la RFC 6797 [23]. Este mecanismo consiste en el envío, por parte de los servidores web, de un campo en el encabezado HTTP, denominado `Strict-Transport-Security`, para notificar al navegador web de que solo deberá utilizar conexiones seguras hacia dicho servidor (HTTPS).

Esta técnica, intenta cubrir los ataques de SSL Strip, pero solo tiene efecto para aquellos sitios que ya han sido visitados al menos una vez utilizando HTTPS, ya que si la primera conexión se realiza mediante un canal inseguro, la comunicación podría verse comprometida. Algunos navegadores web incluyen una lista pre-cargada con algunos sitios web conocidos que soportan HSTS, lo que contribuye a minimizar el riesgo, aunque, claramente, esto no puede escalar a todos los sitios web.



3.3 Session Prediction

Esta técnica hace referencia a la posibilidad de que un atacante pueda predecir el valor de una cookie de sesión, o token, y posteriormente realizar ataques de fuerza bruta para lograr acceso a una aplicación web.

Muchas aplicaciones web utilizan cadenas de texto relativamente sencillas de predecir como valores de ID de sesión. Un atacante podría recopilar distintos tokens y posteriormente analizar su composición y aleatoriedad. Existen herramientas que pueden ayudar a este propósito, como Burp Suite [20].

A partir del análisis del proceso de generación de identificación de sesiones, el atacante podría predecir valores válidos y acceder a la aplicación vulnerable. Por ejemplo, supongamos que una aplicación web genera una cookie de sesión cuando un usuario ingresa al sistema, y dicha cookie posee como valor una cadena de texto como “user74”. Esa cadena podría estar formada por el texto “user” más el número de identificación del usuario, por ejemplo. Entonces, un atacante podría intentar acceder al sistema manipulando dicho valor con otros identificadores, como “user46”, “user04”, etc.

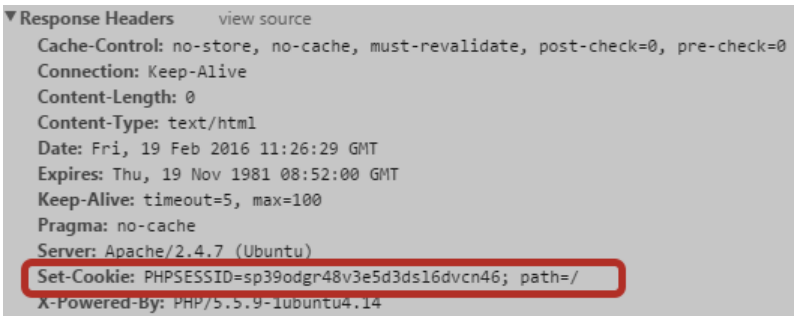
3.4 Session Fixation

A diferencia de la técnica de predicción de sesión, la fijación de sesión permite que el atacante pueda indicar de antemano cuál será el token utilizado en una sesión lícita. Por ejemplo, un atacante podría enviar un link por correo electrónico o cualquier otra vía a la víctima, invitándola a acceder al sitio web vulnerable, de esta forma:

```
http://ejemplo.com/SID=12345678
```

Aquí se aprecia que en el link se está pre-fijando el identificador de sesión. Si el servidor web no realiza la correspondiente validación, el usuario legítimo se autenticará usando ese identificador, y por lo tanto el atacante solo deberá esperar a que la víctima se conecte para tener acceso a la aplicación web. En el caso de que el servidor web no acepte identificadores de sesión que no hayan sido generados previamente, aún así se podría llevar a cabo el ataque.

Por ejemplo, un atacante se conecta con sus credenciales al sitio web y toma nota del valor de la cookie de sesión generada por la aplicación:



Posteriormente, se envía a la víctima el link para acceder al sitio web, pero esta vez con el identificador de sesión que recibió el atacante:

```
http://ejemplo.com/SID=sp39odgr48v3e5d3ds16dvcn46
```

Entonces, cuando la víctima se autentique en el sitio web, el ID utilizado será el mismo que le "fijó" el atacante, quien podrá acceder al sitio como si se tratara del usuario víctima.

Tanto la técnica de fijación de sesión como la de predicción de sesión, pueden mitigarse con implementaciones robustas del manejo de sesiones, por ejemplo, utilizando identificadores aleatorios, que hayan sido generados del lado del servidor, con validez limitada en el tiempo, entre otras recomendaciones, como las indicadas en la guía *Session Management Cheat Sheet* [25], de la Fundación OWASP.

3.5 Cross-Site Request Forgery (CSRF)

Los ataques de tipo CSRF (del inglés Cross-Site Request Forgery), también conocidos como XSRF, se basan en engañar a la víctima para que realice acciones no deseadas sobre un sitio web al cual accedió en forma lícita con sus credenciales de acceso, explotando el hecho de que los sitios web confían en las solicitudes HTTP/S de los usuarios [21]. A diferencia de los ataques de tipo XSS, en CSRF un atacante intenta hacer que una víctima realice una acción sin necesidad de usar un script.

CSRF ataca la funcionalidad de destino que causa un cambio de estado en el servidor, como cambiar la dirección de correo electrónico o contraseña de la víctima, o comprar algo. Obligar a la víctima a recuperar datos no beneficia a un atacante porque el atacante no recibe la respuesta, la víctima sí. Como tal, los ataques CSRF apuntan a solicitudes de cambio de estado. Las aplicaciones web que no realizan la verificación de las solicitudes del cliente, pueden estar expuestas a ataques de este tipo, aún si dichas peticiones están bien formadas y son consistentes, ya que el servidor no puede distinguir entre una petición no intencional y una solicitud auténtica.

Un ejemplo típico para ilustrar esta técnica es el de la eliminación de registros de una base de datos, sin que el usuario víctima esté al tanto de lo que está haciendo. Supongamos una aplicación web vulnerable a CSRF, a la que se conecta un administrador para eliminar un

usuario (por ejemplo, el usuario con identificador 114). La petición GET podría ser similar a ésta:

```
http://www.sitio.com.ar/usuarios/eliminar/114
```

Ahora bien, un atacante podría generar otro sitio web, digamos ***www.sitio2.com.ar***, que incluya algún link como el siguiente:

```
<img src="http://www.sitio.com.ar/usuarios/eliminar/75">
```

Por lo tanto, si el usuario administrador navega por el sitio ***www.sitio2.com.ar*** estando previamente conectado al sitio ***www.sitio.com.ar***, cuando se intente cargar la imagen, automáticamente se procederá a eliminar el usuario con identificador 75, sin siquiera estar al tanto de esto.

Como se puede apreciar, este tipo de ataques no permite obtener información por parte del atacante, pero sí aprovechar la “confianza” que la aplicación tiene sobre el usuario autenticado, para poder llevar a cabo acciones dirigidas, como la del ejemplo precedente, en el que se logra eliminar un registro de una base de datos, sin que el usuario esté al tanto de dicha acción. Para prevenir este tipo de ataques, también debemos acudir a las buenas prácticas de desarrollo de software, fundamentalmente a la utilización de tokens aleatorios en las URLs. La fundación OWASP ofrece una guía [26], que incluye recomendaciones y ejemplos para la mitigación de CSRF.

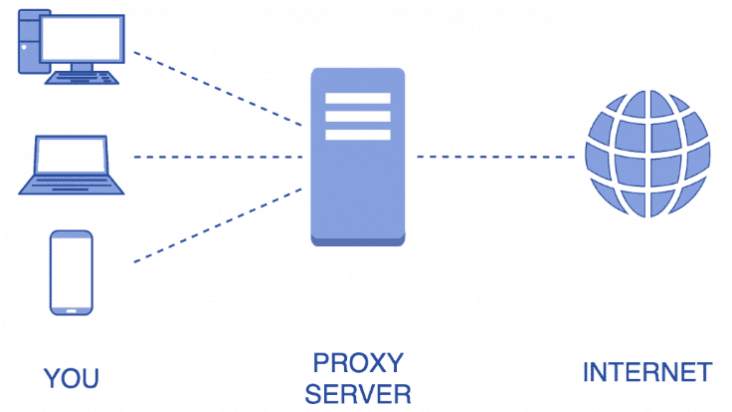
A lo largo de los años, se fueron buscando mecanismos de mitigación para cada una de las técnicas expuestas, como se mencionó en cada uno de los apartados anteriores. Sin embargo, estos mecanismos dependen, en casi todos los casos, de la calidad del desarrollo del software de la aplicación web que los usuarios consumen. En definitiva, si un usuario accede a un sitio web que no está desarrollado siguiendo las mejores prácticas de seguridad, puede estar expuesto a ataques.

En el siguiente capítulo se presenta una propuesta diferente, orientada a minimizar el riesgo que supone el eventual robo de cookies. Es decir, asumiendo que un atacante lograse obtener una cookie de sesión, se buscará que aun así no se vea comprometida la seguridad de los usuarios en el uso de aplicaciones web.

Capítulo 4: Uso de un proxy local

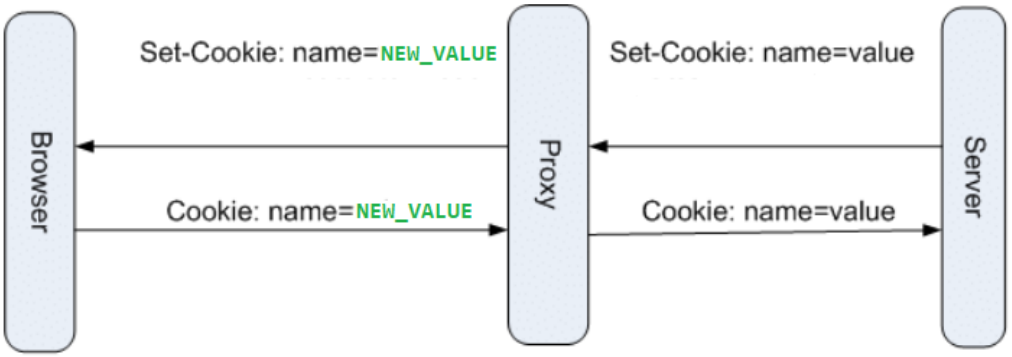
En este capítulo se presenta una alternativa innovadora, que permite que el robo de cookies de los navegadores web no afecte la seguridad de los usuarios en el uso de las aplicaciones web, utilizando un servidor proxy local instalado en el equipo cliente (*endpoint*).

Los servidores Proxy se utilizan como intermediarios en una comunicación entre dos hosts. Muchas empresas utilizan proxies para ofrecer diversas funcionalidades, como control de acceso, registro del tráfico, restricción a determinados tipos de tráfico, mejora de rendimiento, anonimato de la comunicación, caché web, etc.



Ahora bien, ¿cómo puede ayudar un proxy local a evitar el robo de cookies?

El tráfico originado por el navegador web va destinado al proxy, quien posteriormente inicia una nueva conexión contra el sitio web requerido por el usuario. Por lo tanto, el proxy podría interceptar y eventualmente modificar las cabeceras HTTP, alterando las cookies intercambiadas. El objetivo final es que el navegador web reciba cookies con valores modificados por el proxy, y de esta forma, aun en el caso de que dichas cookies sean robadas de los almacenes de los navegadores, no serán de utilidad para los atacantes.



Desde luego, el proxy deberá conservar la información que relaciona o mapea las cookies "reales" de las "ficticias".

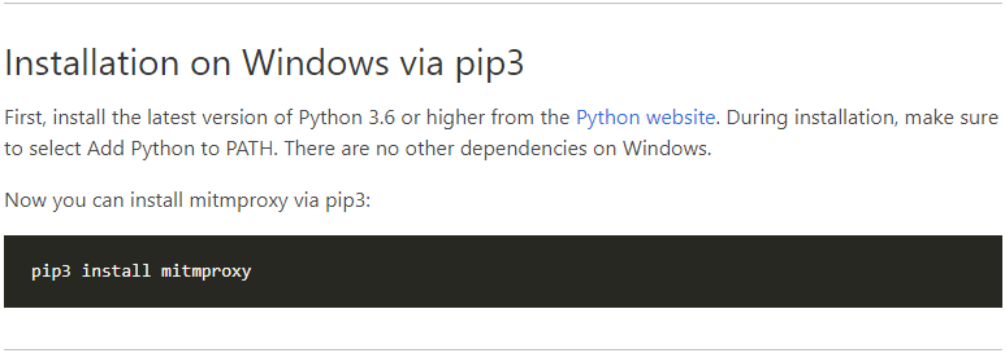
En este capítulo se pretende demostrar la factibilidad técnica de realizar esta manipulación de cookies, lo que daría lugar a una implementación real en un futuro trabajo.

4.1 Instalación y configuración

El producto elegido es **mitmproxy** [27], por ser un software open-source, con soporte para HTTP y HTTPS, con una interfaz web muy útil para realizar debug y particularmente por la posibilidad de extender su funcionalidad mediante el desarrollo de *addons*.

Instalación

El producto ofrece distintos métodos de instalación y es compatible con distintos sistemas operativos. Se decidió usar el método de instalación como paquete de Python, lo que permitirá, a futuro, la creación de un *addon* que permita modificar los headers HTTP.



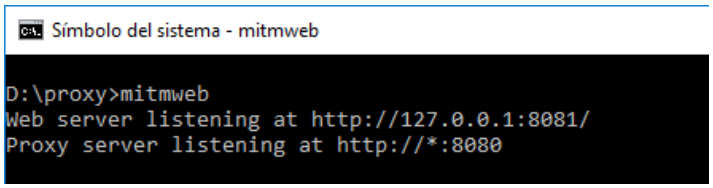
Puesta en funcionamiento

Una vez finalizada la instalación, se pueden utilizar las siguientes herramientas:

mitmdump: Permite ver y/o manipular el tráfico que pasa por el proxy, mediante una interfaz de línea de comandos.

mitmweb: Permite ver y/o manipular el tráfico que pasa por el proxy, mediante una interfaz web. A los efectos de las pruebas de funcionamiento, se usará esta interfaz.

Python API: Permite el desarrollo de scripts para automatizar la manipulación de tráfico.



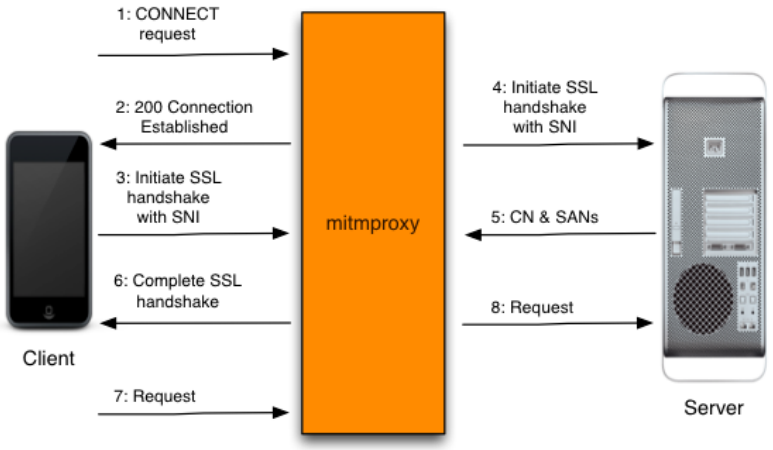
Se invoca el binario mitmweb

Como se puede ver, el proxy queda a la espera de peticiones en el puerto TCP/8080.
Por otra parte, queda disponible la consola web del proxy en *http://localhost:8081.* A continuación, se debe configurar el navegador web para que utilice el proxy local recién instalado. En este caso, se utilizará el navegador Firefox:

4.2 Gestión de certificados digitales

Hasta aquí, cuando el usuario ingrese a cualquier sitio web HTTP, las peticiones se enviarán al proxy. Éste será quien establezca las conexiones con el servidor web y envíe las respuestas al cliente, pudiendo inspeccionar e incluso modificar dicho tráfico.

Pero, ¿qué ocurre con sitios que funcionan sobre HTTPS? El proxy no podría inspeccionar ni modificar el contenido de los paquetes que pasen por él, ya que la información estaría cifrada. La solución que implementa el proxy consiste en realizar el proceso de cifrado/descifrado dos veces: por un lado, entre el cliente y el proxy y por otro, entre el proxy y el servidor web.

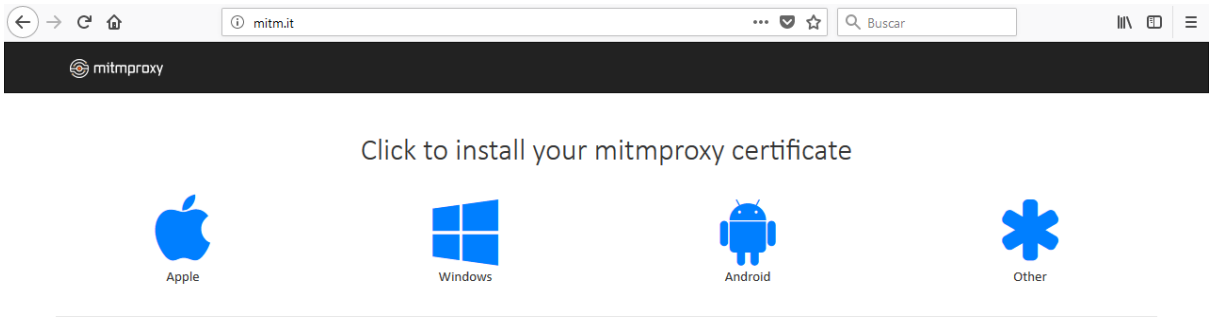


HTTPS explícito

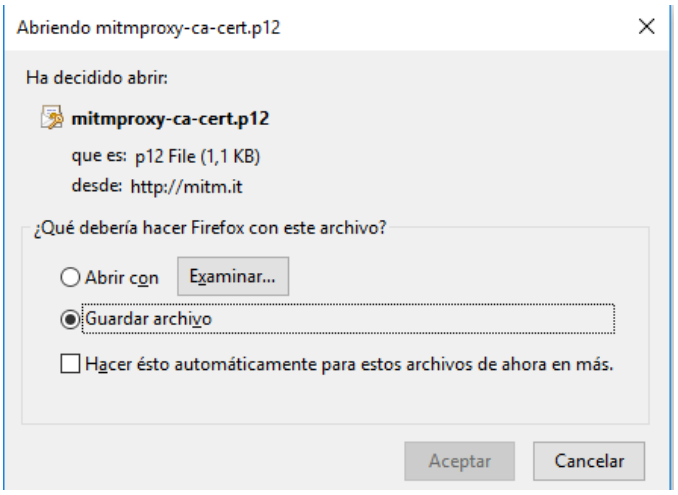
Esta solución posee un inconveniente: ¿Quién firma el certificado digital que el proxy le presenta al cliente en la comunicación que se da entre ellos? El proxy genera certificados bajo demanda firmados por él mismo, para cada uno de los dominios que visita el cliente. Eso implica que el cliente debe confiar en el proxy en su rol de Autoridad de Certificación (CA), es decir, tener instalado el certificado del proxy en el almacén de certificados de confianza (autoridades de certificación) que utilice su navegador.

Obtención del certificado

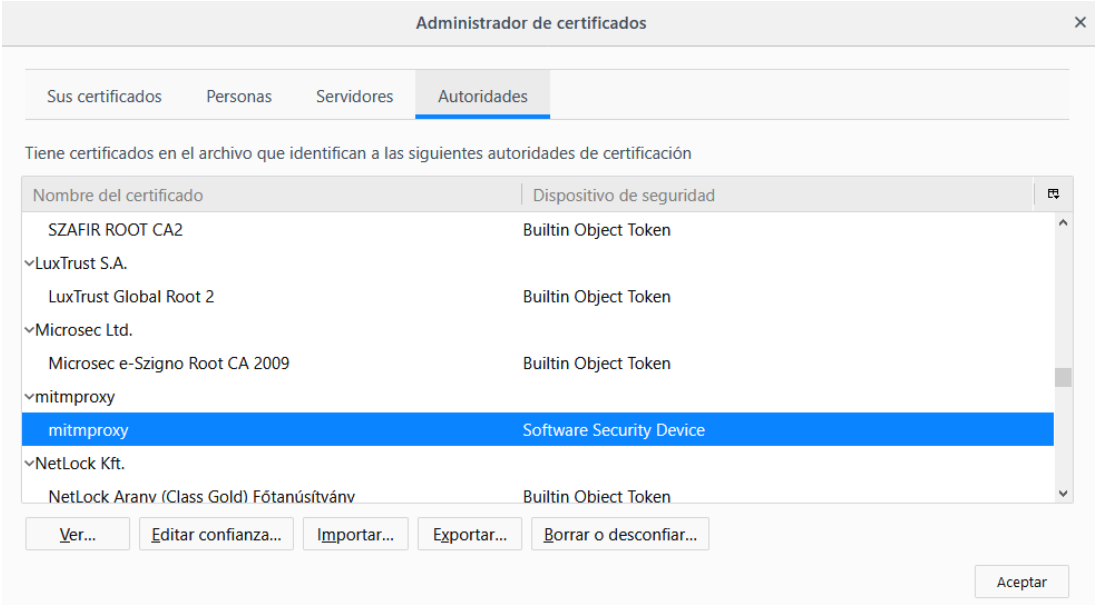
El producto cuenta con una página web integrada que permite la descarga del certificado.
Para acceder, se debe ingresar con el mismo navegador que está configurado para usar el proxy a la siguiente dirección: *http://mitm.it*



Según el sistema operativo seleccionado se muestran las instrucciones para la instalación del certificado, y se ofrece el archivo para su descarga:



A modo de ejemplo, se muestra cómo queda instalado el certificado en el almacén que posee el navegador Firefox (este navegador utiliza un almacén propio de certificados):



El siguiente paso es la prueba de funcionamiento del proxy, para finalmente verificar la factibilidad de modificar los encabezados HTTP/HTTPS, en particular, los relacionados al intercambios de cookies.

4.3 Pruebas de funcionamiento

El objetivo de estas pruebas es verificar el correcto funcionamiento del proxy en distintos escenarios. Se muestra el acceso al sitio y la captura de la primera petición del cliente vista desde la consola web del proxy:

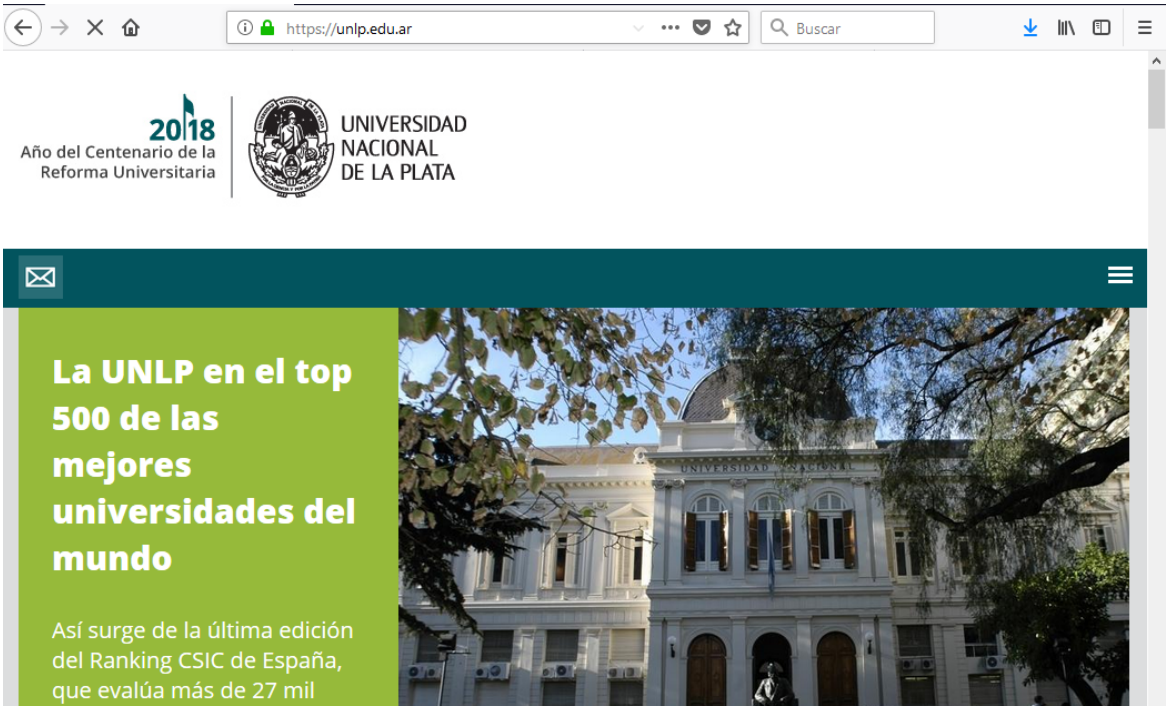
Sitio HTTP1/1: http://sedici.unlp.edu.ar

Request | Response | Details

```
GET http://sedici.unlp.edu.ar/ HTTP/1.1
```

Host	sedici.unlp.edu.ar
User-Agent	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept	text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language	es-AR,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding	gzip, deflate
Cookie	_ga=GA1.3.648965942.1527352151; _gid=GA1.3.953086879.1532808925; JSESSIONID=B6D973CAFFAE3936BA8EF254FAA3321E; uvts=7nhjw0RN6u9aOaoA
Connection	keep-alive
Upgrade-Insecure-Requests	1

Sitio HTTPS/1.1 con certificado válido: https://unlp.edu.ar



Request | Response | Details

```
GET https://unlp.edu.ar/ HTTP/1.1
```

Host	unlp.edu.ar
User-Agent	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept	text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language	es-AR,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding	gzip, deflate, br
Cookie	_ga=GA1.3.648965942.1527352151
Connection	keep-alive
Upgrade-Insecure-Requests	1

Sitio HTTPS/2 con certificado válido: https://www.facebook.com

Request | Response | Details

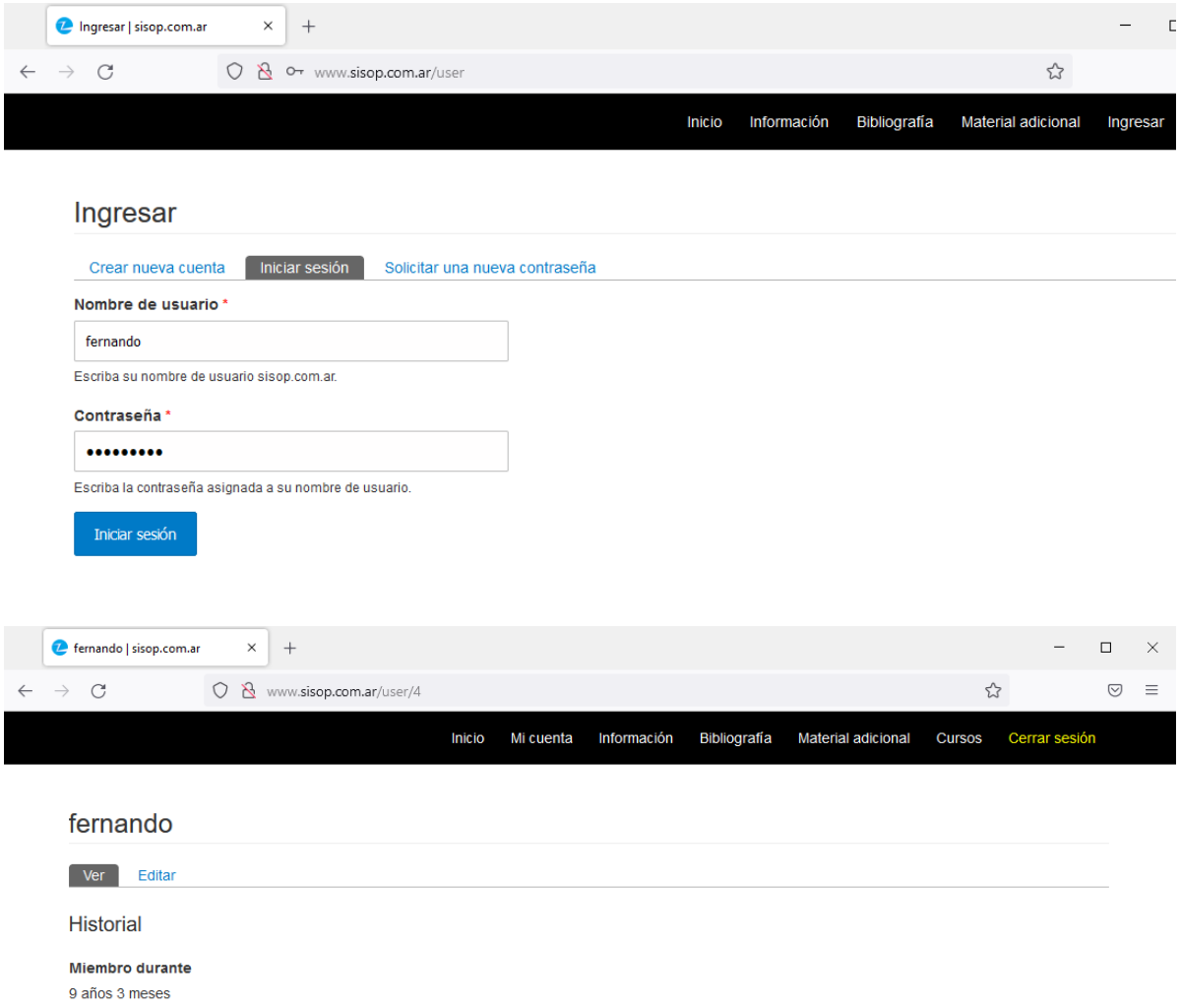
```
GET https://www.facebook.com/ HTTP/2.0
```

:authority	www.facebook.com
user-agent	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:60.0) Gecko/20100101 Firefox/60.0
accept	text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
accept-language	es-AR,es;q=0.8,en-US;q=0.5,en;q=0.3
accept-encoding	gzip, deflate, br
upgrade-insecure-requests	1
cookie	fr=0pCoAfNPmDdjjeBuE..BbO4Jz.Ml.AAA.0.0.BbXPRJ.AWWRiWBA; sb=SfRcW31Kuju6BV8DlDm7GvCp; _js_datr=SfRcW7lMo-R9ehjti85uVfd2; _js_reg_fb_ref=https%3A%2F%2Fwww.facebook.com%2F; _js_reg_fb_gate=https%3A%2F%2Fwww.facebook.com%2F; wd=1049x615

Alterando encabezados HTTP(S)

Una vez verificado el funcionamiento de la navegación web utilizando el proxy local, se realiza una prueba de comportamiento al manipular una cookie de sesión:

1. Se accede a un sitio web, y se realiza la autenticación de un usuario

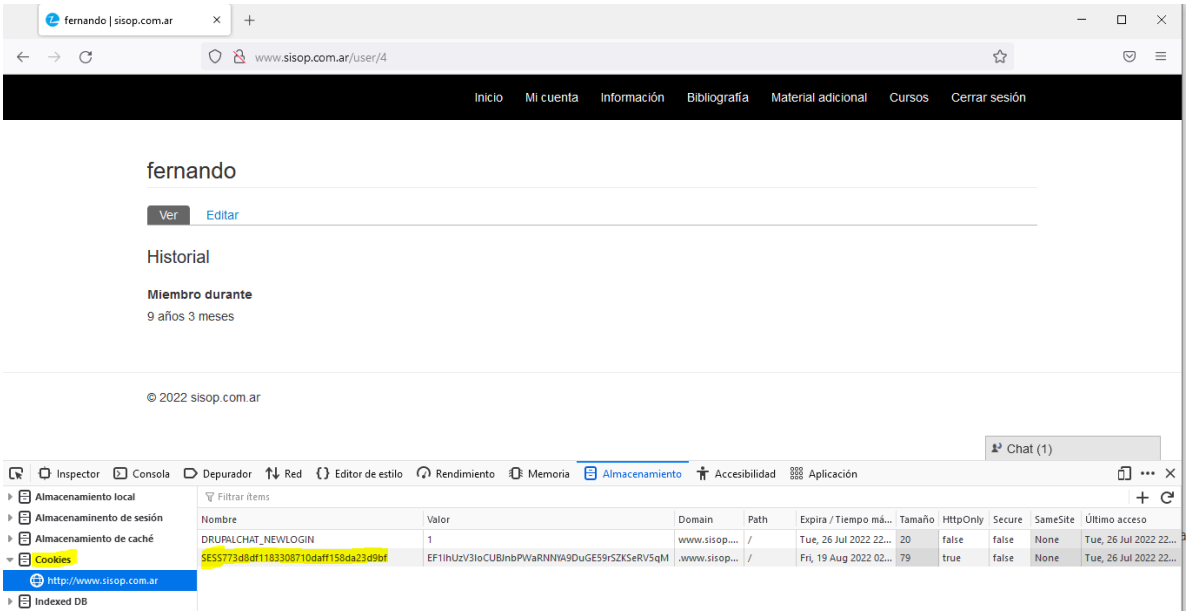


2. Se verifica en la consola del proxy el envío del mensaje POST desde el navegador y posterior respuesta del servidor, incluyendo una cookie de sesión.

   Envío del formulario de login:

   Respuesta del servidor:

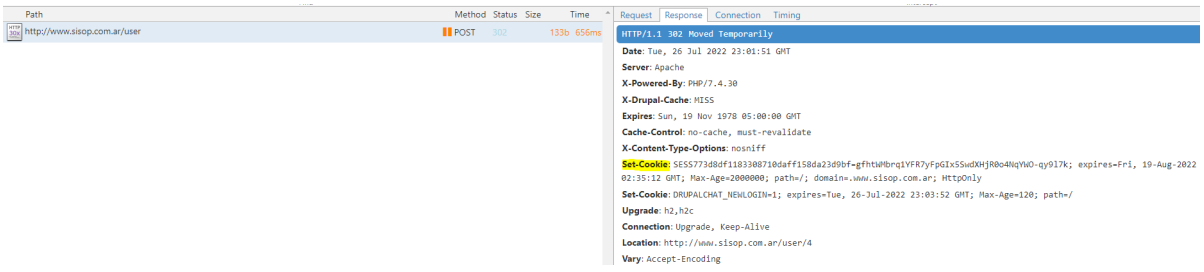
3. Se valida que la cookie de sesión efectivamente quedó almacenada en el navegador web.



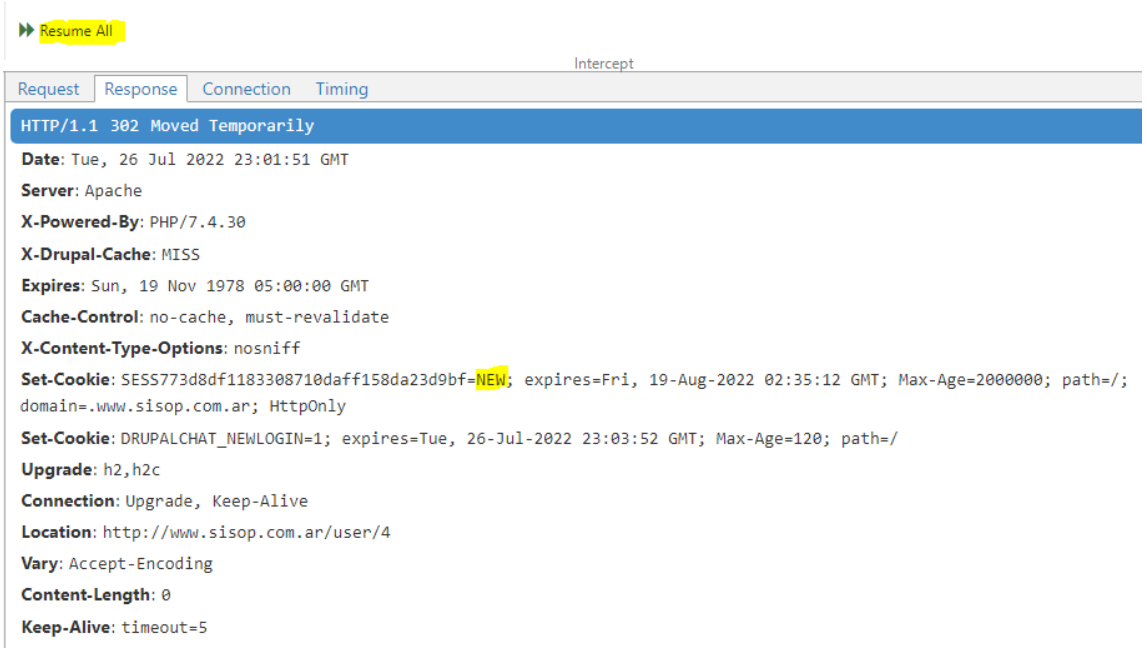
Ahora, utilizando la capacidad de alterar los encabezados HTTP del proxy, se repite el caso de prueba anterior, pero alterando la cookie recibida antes de que llegue al navegador.

4. Se configura el proxy para que cuando reciba un campo “Set-Cookie” en el encabezado HTTP, se detenga, para poder manipular la cabecera:

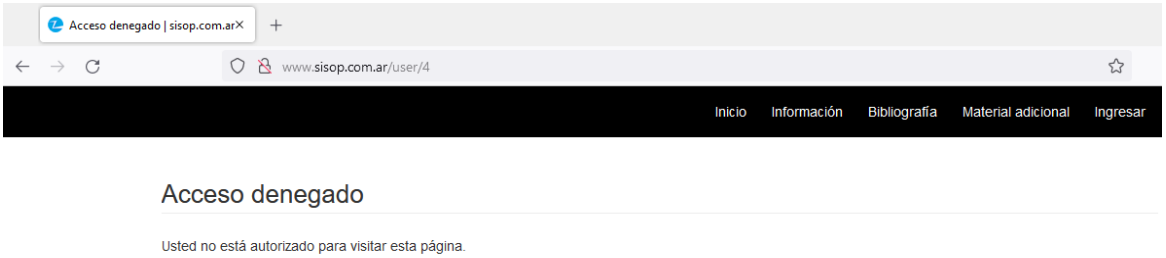
5. Cuando el usuario se intenta loguear al sitio, el proxy detiene la transacción:



6. Se modifica la cookie, cambiando su valor por “NEW” y luego se presiona “Resume” para que el tráfico continúe su camino hacia el navegador



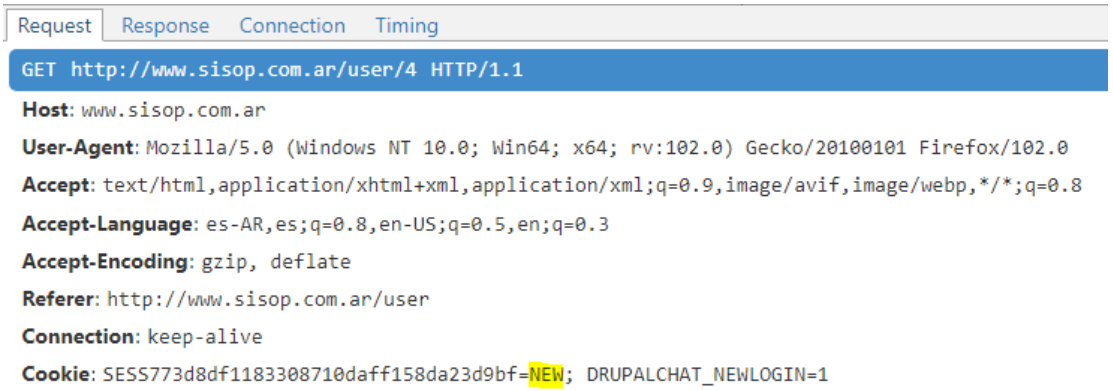
7. La respuesta que ve el usuario es la siguiente:



Ahora bien, ¿por qué el usuario ve esta página de Acceso denegado?

Luego de que el valor de la cookie fue modificado, ésta llegó normalmente al navegador, y la petición posterior incluyó esa cookie (con valor “NEW”). El servidor, al recibir esa cookie, verifica que no coincide con ninguna cookie generada previamente por éste, entonces responde con la página de acceso denegado, y elimina la cookie de sesión. Esto puede observarse en la última petición y respuesta:

Petición:



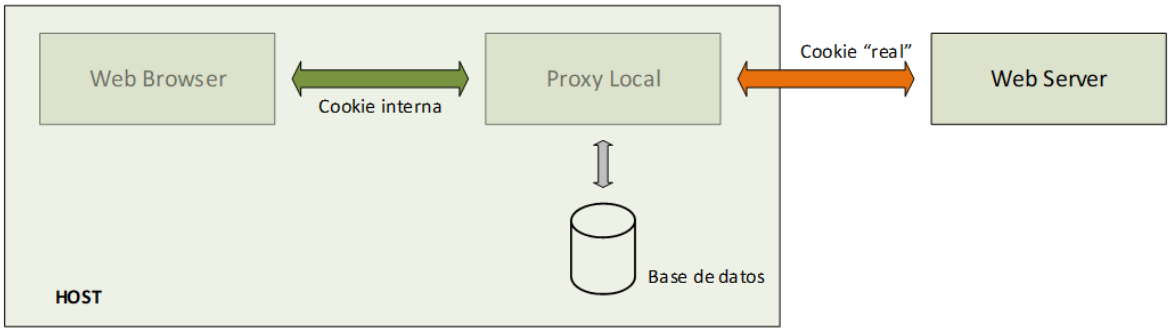
Respuesta:

Request | Response | Connection | Timing

```
HTTP/1.1 403 Forbidden
Date: Tue, 26 Jul 2022 23:19:41 GMT
Server: Apache
X-Powered-By: PHP/7.4.30
X-Content-Type-Options: nosniff
Content-Language: es
X-Frame-Options: SAMEORIGIN
Permissions-Policy: interest-cohort=()
X-Generator: Drupal 7 (http://drupal.org)
Cache-Control: public, max-age=0
Expires: Sun, 19 Nov 1978 05:00:00 GMT
Vary: Cookie,Accept-Encoding
Set-Cookie: SESS773d8df1183308710daff158da23d9bf=deleted; expires=Thu, 01-Jan-1970 00:00:01 GMT; Max-Age=0; path=/;
domain=.www.sisop.com.ar; HttpOnly
Upgrade: h2,h2c
Connection: Upgrade, Keep-Alive
Etag: "1658877581-0"
Last-Modified: Tue, 26 Jul 2022 23:19:41 GMT
Keep-Alive: timeout=5
Transfer-Encoding: chunked
Content-Type: text/html; charset=utf-8
```

Esta prueba demuestra la posibilidad de alterar los encabezados HTTP tanto en mensajes de solicitud como de respuesta (Request / Response), utilizando un proxy local, y de esta forma, poder realizar la manipulación del valor de las cookies intercambiadas entre los navegadores y los servidores web, para que aun en caso de que las cookies sean robadas de los almacenes de datos de los navegadores, sus valores no resulten de utilidad para los eventuales atacantes.

Para lograr esto, se propone el desarrollo de un software que, apoyado en el proxy local, tenga como principal función el almacenamiento de las cookies que eventualmente envíen los servidores web, y la generación de nuevos valores aleatorios asociados a las cookies originales. Se tendría un “mapeo” entre las cookies reales y las generadas por el software.



Capítulo 5: Conclusiones y trabajos futuros

La utilización de cookies para el manejo de sesiones web se extendió muy rápidamente, en particular desde la llegada de la denominada Web 2.0. Con el paso de los años, se fueron actualizando y mejorando las versiones de los protocolos que transportan la información entre servidores y navegadores web, sin embargo, el uso de cookies sigue estando vigente.

El robo de cookies es una problemática compleja, ya que un atacante que sea capaz de obtener una cookie de sesión de un usuario, podrá utilizarla y entonces suplantar su identidad. Muchos sitios web que manejan información sensible de los usuarios, como por ejemplo entidades bancarias, generan cookies con una validez de apenas unos pocos minutos, para minimizar los riesgos, pero aun así, un ataque por robo de cookies sigue siendo posible.

5.1 Conclusiones

La presente propuesta, no evita que una cookie sea robada, pero la manipulación de la misma mediante el proxy local, permite que esas cookies almacenadas en los navegadores web no posean los valores reales esperados por los servidores web. De este modo, deja de tener sentido la explotación de técnicas para obtener las cookies con ataques de tipo XSS.

Por otro lado, con la utilización del proxy local, no se requieren adaptaciones en los navegadores web ni en los servidores web, ya que el intercambio de cookies se sigue dando. Esto posibilita el acceso a cualquier sitio web, pues se mantiene el mecanismo de manejo de sesiones, incluyendo el tiempo de expiración de la sesión, y la no dependencia de una ubicación física específica para mantener dicha sesión.

5.2 Trabajos futuros

En un trabajo posterior, se pretende realizar el desarrollo de un software que, basado en el proxy local presentado aquí, permita realizar el intercambio de cookies, en forma automática y transparente para el usuario, almacenando internamente la relación *cookie real-cookie ficticia*.

Adicionalmente, se podría avanzar en trabajos futuros sobre las siguientes temáticas relacionadas:

- Evaluación del mecanismo propuesto para el caso de cookies que no manejan sesiones, como cookies de personalización de contenidos y/o cookies de terceros.

- Manejo de sesiones utilizando Websockets [28]

- Análisis de seguridad de las sesiones de usuarios, cuando se utiliza JSON Web Tokens [29] cómo técnica de autenticación.

Bibliografía

[1] Hypertext Transfer Protocol
https://tools.ietf.org/html/rfc7230

[2] HTTP Over TLS
https://tools.ietf.org/html/rfc2818

[3] HTTP State Management Mechanism
https://tools.ietf.org/html/rfc6265.html

[4] The Secure Sockets Layer (SSL) Protocol
https://tools.ietf.org/html/rfc6101

[5] The Transport Layer Security (TLS) Protocol
https://tools.ietf.org/html/rfc5246

[6] The Open Web Application Security Project (OWASP)
https://www.owasp.org

[7] OWASP Top Ten Project
https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

[8] Diseño y desarrollo de un mecanismo más seguro de manejo de sesiones web (N. Macia, UNLP, 2017)
http://sedici.unlp.edu.ar/handle/10915/60821

[9] Evolución del protocolo HTTP
https://developer.mozilla.org/es/docs/Web/HTTP/Basics_of_HTTP/Evolution_of_HTTP

[10] Usage statistics of HTTP/2 for websites
https://w3techs.com/technologies/details/ce-http2/all/all

[11] Apache Server – KeepAliveTimeout
https://httpd.apache.org/docs/2.4/mod/core.html#keepalivetimeout

[12] HTTP State Management Mechanism
https://datatracker.ietf.org/doc/html/rfc6265

[13] Lou Montulli
https://es.wikipedia.org/wiki/Lou_Montulli

[14] John Giannandrea
https://es.wikipedia.org/wiki/John_Giannandrea

[15] Asymmetric algorithms
https://cryptography.io/en/latest/hazmat/primitives/asymmetric/index.html

[16] Internet X.509 Public Key Infrastructure Certificate
https://datatracker.ietf.org/doc/html/rfc5280

[17] Wireshark
https://www.wireshark.org/

[18] Document Object Model (DOM)
https://es.wikipedia.org/wiki/Document_Object_Model

[19] Cisco Port Security
https://community.cisco.com/t5/networking-documents/how-to-configure-port-security-on-cisco-catalyst-switches-that/ta-p/3132907

[20] Burp Suite
https://portswigger.net/burp/communitydownload

[21] Alcorn, W., Frichot, C. & Orrù, M. (2014). The Browser Hacker's Handbook. Indianapolis : John Wiley & Sons, Inc

[22] Criptografía simétrica
https://cryptography.io/en/latest/hazmat/primitives/symmetric-encryption/

[23] HTTP Strict Transport Security (HSTS)
https://en.wikipedia.org/wiki/HTTP_Strict_Transport_Security

[24] Cross Site Scripting Prevention Cheat Sheet
https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

[25] Session Management Cheat Sheet
https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html

[26] Cross-Site Request Forgery Prevention Cheat Sheet
https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

[27] mitmproxy - An interactive HTTP Proxy
https://mitmproxy.org/

[28] The WebSocket Protocol
https://datatracker.ietf.org/doc/html/rfc6455

[29] JSON Web Token (JWT)
https://www.rfc-editor.org/rfc/rfc7519