

# ATAQUE 1

## A) Nombre del ataque

A3 - Cross-Site Scripting (XSS) Persistente (de segundo orden) - Inyección HTML

## B) Pasos

### 1. Acceso al blog

OWASP Mutillidae II: Web Pwn in Mass Production

Version: 2.6.24 Security Level: 0 (Hosed) Hints: Enabled (1 - 5cr1pt K1dd1e) Logged In User: usuario ()

Home Logout Toggle Hints Show Popup Hints Toggle Security Enforce SSL Reset DB View Log View Captured Data

OWASP 2013 A1 - Injection (SQL)

OWASP 2010 A1 - Injection (Other)

OWASP 2007 A2 - Broken Authentication and Session Management

Web Services A3 - Cross Site Scripting (XSS)

HTML 5 A4 - Insecure Direct Object References

Others A5 - Security Misconfiguration

Documentation A6 - Sensitive Data Exposure

Resources A7 - Missing Function Level Access Control

A8 - Cross Site Request Forgery (CSRF)

A9 - Using Components with Known Vulnerabilities

A10 - Unvalidated Redirects and Forwards

Getting Started Project Whitepaper

Release Announcements

Video Tutorials

What's New? C

PHP MyAdmin C

Installation Instructions

Tools

Latest Version

Installation Instructions

Usage Instructions

pesky PHP errors

Like Mutillidae? Check out how to help

Add to your blog

View someone's blog

Show Log

Listing of vulnerabilities

Blog Report Email Address

Release Announcements

Feature Requests

Kali Linux

Samurai Web Testing Framework

sqlmap

Some Useful Firefox Add-ons

### 2. Código del aviso

OWASP Mutillidae II: Web Pwn in Mass Production

Version: 2.6.24 Security Level: 0 (Hosed) Hints: Enabled (1 - 5cr1pt K1dd1e) Logged In User: usuario ()

Home Logout Toggle Hints Show Popup Hints Toggle Security Enforce SSL Reset DB View Log View Captured Data

OWASP 2013

OWASP 2010

OWASP 2007

Web Services

HTML 5

Others

Documentation

Resources

Getting Started Project Whitepaper

Release Announcements

Video Tutorials

Welcome To The Blog

Back Help Me!

Hints

Add New Blog Entry

View Blogs

Add blog for usuario

Note: <b>, <i> and <u> are now allowed in blog entries

```
<div id='jdlogin' style='padding: 20px; position: absolute; top: 250px; left: 400px; background-color:#ffcccc; border: solid black 1px;'><form action='index.php?do=logout' method='post'><table style='font-weight:bold;'><tr><td colspan='2' style='font-size:20px;'> You must accept the cookies to continue.</td></tr><tr><td colspan='2' style='text-align: center;'><input value='Accept' type='submit'></td></tr></table></form></div>
```

Save Blog Entry

View Blogs

0 Current Blog Entries

Name	Date	Comment
------	------	---------

### 3. Display del aviso

**OWASP Mutillidae II: Web Pwn in Mass Production**

Version: 2.6.24 Security Level: 0 (Hosed) Hints: Enabled (1 - 5cr1pt K1dd1e) Logged In User: **usuario** ()

Home Logout Toggle Hints Show Popup Hints Toggle Security Enforce SSL Reset DB View Log View Captured Data

**Welcome To The Blog**

[Back](#) [Help Me!](#)

**You must accept the cookies to continue.**

[Accept](#)

**Note: <b>, <i> and <u> are now allowed in blog entries**

[Add New Blog Entry](#) [View Blogs](#)

[Save Blog Entry](#)

[View Blogs](#)

**1 Current Blog Entries**

	Name	Date	Comment
1	usuario	2019-05-09 06:23:49	

### C) Resultado obtenido

### 4. Cierre de sesión

**OWASP Mutillidae II: Web Pwn in Mass Production**

Version: 2.6.24 Security Level: 0 (Hosed) Hints: Enabled (1 - 5cr1pt K1dd1e) Not Logged In

Home Login/Register Toggle Hints Show Popup Hints Toggle Security Enforce SSL Reset DB View Log View Captured Data

**Login**

[Back](#) [Help Me!](#)

[Hints](#)

**Please sign-in**

**Username**

**Password**

[Login](#)

*Dont have an account? [Please register here](#)*

## D) Explicación

Los mensajes del blog se muestran tal y como el usuario los introdujo, sin escapar caracteres especiales. Por lo tanto, podemos inyectar código HTML que será interpretado por el navegador pudiendo introducir formularios o casi cualquier tipo de etiqueta HTML.

En este caso hemos simulado un aviso (introducido en la entrada del blog en la foto 2) que mostrará un mensaje para aceptar cookies (mostrado en la foto 3), el cual suele ser aceptado típicamente sin prestar ningún tipo de atención por la mayoría de los usuarios.

En el código hemos utilizado el atributo *action* del formulario para redirigir al usuario a la página de cierre de sesión (foto 4), lo cual sucede cuando este usuario hace *click* en el botón de “Aceptar”.

## E) Solución

Antes de guardar el input de la entrada de blog del usuario en la base de datos se debe prestar atención a ese input comprobando si contiene caracteres especiales de HTML, puesto que al mostrarse las entradas del blog directamente, si una de ellas contiene etiquetas HTML, el navegador las interpretará como tal.

Para ello podemos utilizar la función *htmlspecialchars()* de PHP y evitaremos esta inyección muy fácilmente.

## F) Opcional (BONUS)

Hemos obtenido el código fuente de Mutillidae II a través de GitHub.

```
G) // Grab inputs
H)         if ($lProtectAgainstSQLInjection){
I)             // This might prevent SQL injection on the insert.
J)             $lBlogEntry = $MySQLHandler-
>escapeDangerousCharacters($_POST["blog_entry"]);
K)         }else{
L)             $lBlogEntry = $_REQUEST["blog_entry"];
M)         }// end if
```

Estas son las líneas 121 – 127 del archivo *'add-to-your-blog.php'*, en las que se obtiene el texto introducido por el usuario para posteriormente introducirlo en la base de datos mediante esta línea.

```
$SQLQueryHandler->insertBlogRecord($lLoggedInUser, $lBlogEntry);
```

Para evitar la inyección de HTML podríamos cambiar esta última línea por:

```
$SQLQueryHandler->insertBlogRecord($lLoggedInUser,  
htmlspecialchars($lBlogEntry));
```

# ATAQUE 2

## A) Nombre del ataque

## A2 - Broken Authentication and Session Management - Authentication Bypass (Via Cookies)

## B) Pasos

### 1. Registro de usuario

The screenshot shows the OWASP Mutillidae II web application interface. The header includes the application name, version (2.6.24), security level (0 - Hosed), hints status (Enabled), and user status (Not Logged In). A navigation menu on the left lists various resources like OWASP 2013, OWASP 2010, OWASP 2007, Web Services, HTML 5, Others, Documentation, and Resources. The main content area is titled 'Register for an Account' and contains a registration form. The form has fields for Username (filled with 'sam'), Password (filled with '\*\*\*\*\*'), Confirm Password (filled with '\*\*\*\*\*'), and Signature (filled with 'sam'). There is a 'Create Account' button at the bottom right of the form. A 'Hints' section is visible above the form, and a 'Switch to RESTful Web Service Version of this Page' link is present.

### 2. Visualización de cookies

The screenshot shows a browser's developer console with the 'Console' tab selected. The console displays several error messages, including 'A form was submitted in the windows-1252 encoding which cannot encode all Unicode characters...' and 'ReferenceError: onLoadOfBody is not defined'. Below the errors, the console shows the output of a command to view cookies: `document.cookie`. The output is a long string of cookies: `"showhints=1; username=sam; uid=25; PHPSESSID=fsf4sd4o16lhp4ne9fmq7ph574; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada"`.

### 3. Comando

The screenshot shows a browser's developer console with the 'Console' tab selected. The console displays several error messages, including 'The character encoding of the HTML document was not declared...' and 'ReferenceError: onLoadOfBody is not defined'. Below the errors, the console shows the output of a command to view cookies: `document.cookie`. The output is a long string of cookies: `"showhints=1; username=sam; uid=25; PHPSESSID=fsf4sd4o16lhp4ne9fmq7ph574; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada"`.

### C) Resultado obtenido

#### 4. Sesión de administrador

**Logged In Admin: admin (g0t r00t?)**

### D) Explicación

Para que una aplicación web pueda manejar las sesiones de los usuarios que inician sesión, suelen utilizarse las *cookies*, que almacenan la información de la sesión de dicho usuario.

En este caso, al registrarnos (Foto 1) y posteriormente iniciar sesión, hemos podido comprobar la información de nuestras *cookies* al utilizar el comando *document.cookie* en la consola a la que podemos acceder el Mozilla (u otros navegadores como Google Chrome) presionando F12 y accediendo a la pestaña *Console* (Foto 2).

Al utilizar dicho comando hemos podido observar el valor de la *cookie* *'uid'*, el cual era 25. Al volver a registrarnos e iniciar sesión con un nuevo usuario, comprobamos que el valor de la misma *cookie* era esta vez 26, es decir, un número más que la anterior.

Con esto podemos deducir que cada vez que se crea un nuevo usuario, el *'uid'* que se le asigna es el número inmediatamente siguiente al *'uid'* anterior. Es por esto que podemos pensar que aquel usuario que tenga el *'uid'* con valor 1, será el primer usuario creado para la página, que comúnmente suele ser el administrador de la página.

De esta manera, al utilizar el comando *document.cookie="uid=1"* en la consola (Foto 3), seremos capaces de editar el valor de nuestra *cookie* y, al refrescar la página podremos comprobar cómo estamos identificados con la cuenta del administrador (Foto 4).

### E) Solución

Hay diversas formas de manejar las sesiones de los usuarios que han iniciado sesión en una página web. En este caso se utilizan *tokens* de autorización en el lado del cliente, lo cual no es del todo seguro.

Esa podría ser una forma de evitar este *bypass* de autenticación. Sin embargo, también es posible seguir utilizando *tokens* de autorización

en el lado del cliente y evitar que las *cookies* puedan ser accedidas y editadas por cualquier usuario tan fácilmente.

Para ello, a la hora de crear la *cookie* debemos asignarle la *flag* 'HttpOnly', la cual, si está habilitada, evitaremos que esta *cookie* pueda ser accedida mediante código JavaScript como hemos hecho en la consola de comandos del navegador.

#### F) Opcional (BONUS)

Hemos obtenido el código fuente de Mutillidae II a través de GitHub.

A la hora de analizar el código, hemos comprobado que la *cookie* 'uid' es establecida mediante las funciones 'setcookie()' o 'setrawcookie()' de PHP (líneas 114 y 119 del archivo 'process-login-attempt.php').

```
if ($lProtectCookies){
    $lUsernameCookie = $Encoder->encodeForURL($lRecord->username);
    setcookie("username", $lUsernameCookie, 0, "", "", FALSE, TRUE);
    setcookie("uid", $lRecord->cid, 0, "", "", FALSE, TRUE);
}
else {
    //setrawcookie() allows for response splitting
    $lUsernameCookie = $lRecord->username;
    setrawcookie("username", $lUsernameCookie);
    setrawcookie("uid", $lRecord->cid);
} // end if $lProtectCookies
```

Para añadirle la *flag* 'HttpOnly' a nuestra cookie es tan sencillo como cambiar el valor del último parámetro de la función a 'TRUE', el cual se corresponde a la activación de dicha *flag*, tal y como se hace en el código anterior en el caso de que la variable '\$lProtectCookies' valga *True*, lo cual es así en el modo seguro. Como estamos en el modo inseguro, ya que Mutillidae II nos sirve para comprobar las vulnerabilidades de una página web insegura, el valor de esta variable es 'False', por lo que la *cookie* se establece en el apartado *else* del código sin los parámetros opcionales que asignan los valores de *flags* como la de 'HttpOnly', y sus valores son 'FALSE' de forma predeterminada:

```
setcookie ( string $name [, string $value [, int $expire = 0 [, string $path [, string $domain [, bool $secure = false [,
bool $httponly = false ]]]]] ) : bool

setrawcookie ( string $name [, string $value [, int $expire = 0 [, string $path [, string $domain [, bool $secure = false
[, bool $httponly = false ]]]]] ) : bool
```

Por ello, para añadir la *flag* debemos establecer el valor de la variable *`$!ProtectCookies`* a *True* o cambiar la línea :

```
setrawcookie("uid", $!Record->cid);
```

a la que se accede en el *else* por:

```
setrawcookie("uid", $!Record->cid, 0, "", "", FALSE, TRUE);
```



# ATAQUE 3

## A) Nombre del ataque

A4 – Insecure Direct Object References – Credits – Unvalidated Redirects

## B) Pasos

### 1. Link no validado

Version: 2.6.24 Security Level: 0 (Hosed) Hints: Enabled (1 - 5script K1dd1e) Logg

Home Logout Toggle Hints Show Popup Hints Toggle Security Enforce SSL Reset DB View

**Credits**

Back Help Me!

Hints

Unvalidated Redirects

Created by Jeremy "webpwnized" Druin. Based on Mutillidae 1.0 from Adrian "Irongeek" Crenshaw.

OWASP  
ISSA Kentuckiana  
OWASP Louisville  
Helpful Firefox Add-Ons

Getting Started: Project Whitepaper

Release Announcements

```
html > body > table.main-table-frame > tbody > tr > td > blockquote > div.label > a
```

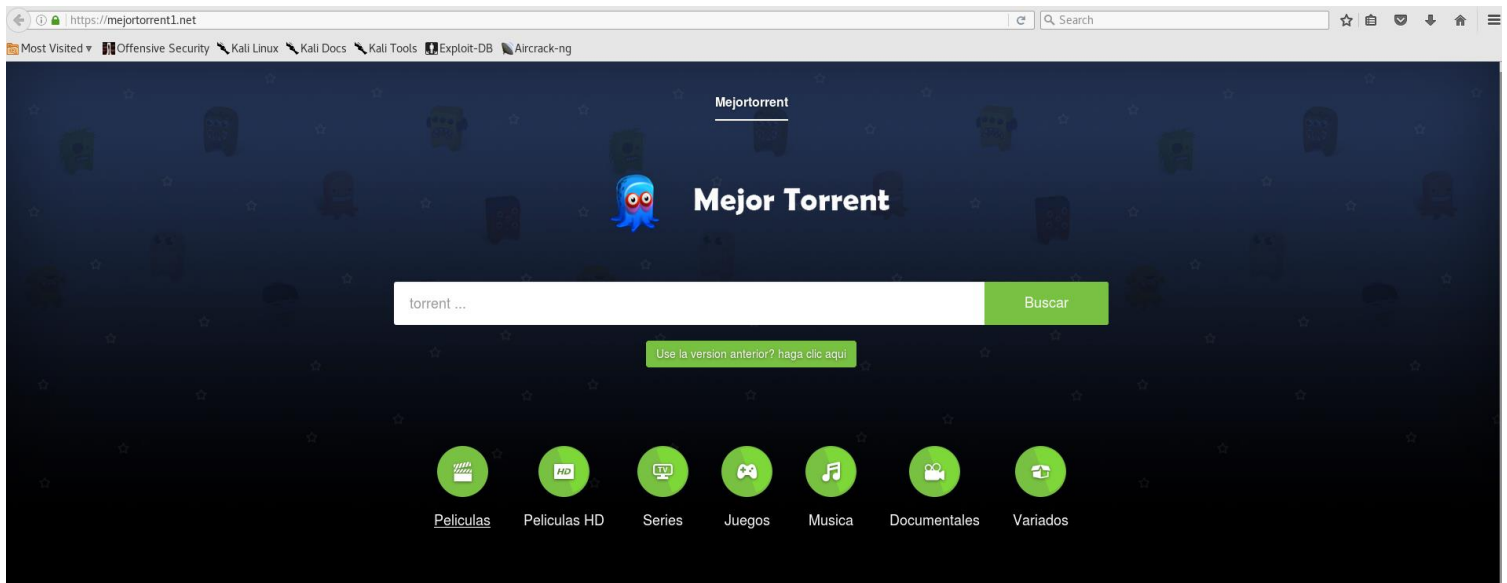
```
<div></div>  
<div class="label" title="" arbitraryredirectionpoint="1">  
  <a href="index.php?page=redirectandlog.php&forwardurl=http://www.owasp.org">OWASP</a>  
</div>  
<div class="label" title="" arbitraryredirectionpoint="1"></div>  
<div class="label" title="" arbitraryredirectionpoint="1"></div>  
<div class="label" title="" arbitraryredirectionpoint="1"></div>  
<!--I think the database password is set to blank or _-->  
<!--End Content-->  
</blockquote>  
</td>  
</tr>  
</tbody>
```

### 2. Link malicioso



## C) Resultado obtenido

### 3. Redirección



## D) Explicación

Mutillidae II contiene una página llamada '*Credits*' la cual contiene 4 *links* que redirigen a páginas externas que son de utilidad para los usuarios que quieran informarse más acerca de la página.

Sin embargo, estos *links* redirigen a una página del propio dominio de Mutillidae II (*index.php?page=redirectandlog.php*), la cual se encarga de redirigir una vez más a la página destino establecida en el parámetro GET '*forwardurl*' (Foto 1).

Esta es una vulnerabilidad importante, puesto que cambiando dicho parámetro (Foto 2) podríamos redirigir al usuario a cualquier página que no sea segura (Foto 3), perdiendo así la confianza del usuario que utilizó dicho *link* pensando que el dominio Mutillidae II en este caso, era seguro.

## E) Solución

En lugar de establecer el destino del *link* directamente con el parámetro '*forwardurl*' explícitamente, podríamos usar un símbolo como un *integer* para representar el lugar al que queremos ir y de esta manera el *integer* sería comprobado en una especie de tabla en la página web donde cada número corresponda a una URL de destino que de esta manera podemos asegurar que será totalmente segura.

## F) Opcional (BONUS)

Hemos obtenido el código fuente de Mutillidae II a través de GitHub.

Como hemos dicho, en los niveles de seguridad 0 y 1, los *links* utilizan este tipo de redirección añadiendo al parámetro *'forwardurl'* directamente la URL de destino:

```
switch ($_SESSION["security-level"]){
    case "0": // This code is insecure
    case "1": // This code is insecure
        $LOWASPURLReference = "http://www.owasp.org";
        $lKYISSAURLReference = "http://www.issa-kentuckiana.org";
        $LOWASPLouisvilleURLReference =
"http://www.owasp.org/index.php/Louisville";
        $lMutillidaeFirefoxAddOnsURLReference =
"https://addons.mozilla.org/en-US/firefox/collections/jdruin/pro-web-developer-ga-pack/";
```

```
<div class="label" ArbitraryRedirectionPoint="1">
    <a href="index.php?page=redirectandlog.php&forwardurl=<?php echo
$LOWASPURLReference; ?>">OWASP</a>
</div>
<div class="label" ArbitraryRedirectionPoint="1">
    <a href="index.php?page=redirectandlog.php&forwardurl=<?php echo
$lKYISSAURLReference; ?>">ISSA Kentuckiana</a>
</div>
<div class="label" ArbitraryRedirectionPoint="1">
    <a href="index.php?page=redirectandlog.php&forwardurl=<?php echo
$LOWASPLouisvilleURLReference; ?>">OWASP Louisville</a>
</div>
<div class="label" ArbitraryRedirectionPoint="1">
    <a href="index.php?page=redirectandlog.php&forwardurl=<?php echo
$lMutillidaeFirefoxAddOnsURLReference; ?>">Helpful Firefox Add-Ons</a>
</div>
```

Sin embargo, en el nivel de seguridad 5, este parámetro contendrá simplemente un *integer*:

```
case "5": // This code is fairly secure
    $LOWASPURLReference = "2";
    $lKYISSAURLReference = "3";
    $LOWASPLouisvilleURLReference = "4";
    $lMutillidaeFirefoxAddOnsURLReference = "10";
    break;
```

(Archivo *credits.php*)

De esta manera, al redirigirnos a la página *'index.php?page=redirectandlog.php'* y dependiendo del nivel de seguridad, la página nos redirigirá o bien directamente a la URL establecida en el parámetro:

```
switch ($_SESSION["security-level"]){
    case "0": // This code is insecure
    case "1": // This code is insecure
        $forwardurl=$_REQUEST["forwardurl"];
        $LogHandler->writeToLog("Redirected user to: " .
$forwardurl);
        echo '<meta http-equiv="refresh"
content="0;url=' . $forwardurl . '>';
        exit
        break;
```

o bien a la URL a la que corresponda el *integer* que se ha pasado por el parámetro *'forwardurl'*:

```
case "5":
    $forwardurl=$_GET["forwardurl"];
    $lURL = "";
    switch($forwardurl){
        case 1: $lURL = "http://www.irongeek.com/";break;
        case 2: $lURL = "http://www.owasp.org";break;
        case 3: $lURL = "http://www.issa-kentuckiana.org/";break;
        case 4: $lURL = "http://www.owasp.org/index.php/Louisville";break;
        case 5: $lURL = "http://www.pocodoy.com/blog/";break;
        case 6: $lURL = "http://www.room362.com/";break;
        case 7: $lURL = "http://www.isd-podcast.com/";break;
        case 8: $lURL = "http://pauldotcom.com/";break;
        case 9: $lURL = "http://www.php.net/";break;
        case 10:$lURL = "https://addons.mozilla.org/en-
US/firefox/collections/jdruin/pro-web-developer-qa-pack/";break;
    }
    $LogHandler->writeToLog("Redirected user to: " .
$lURL);
    echo '<meta http-equiv="refresh" content="0;url=' . $lURL . '>';
    exit;
break;
```

(Archivo *redirectandlog.php*)