

Instituto Tecnológico de San Juan del Río



(Tópicos De Ciberseguridad)

Insecure-webapi

P R E S E N T A:

Luis Josué Chávez Andrade 20590237
Manuel Hernández Rayoso 20590255
Hugo Leonardo Robles Torres 20590288

PERIODO
ENERO-JUNIO 2025



Insecure-webapi

Tabla de Correcciones

Código OWASP	Vulnerabilidad	Solución Aplicada
A01	Broken Access Control	Verificación de propiedad con JOIN
A02	Cryptographic Failures	Reemplazo de MD5 por password_hash()
A03	SQL Injection (Imagen)	Consultas preparadas
A04	Insecure Design (Imágenes)	Validación de extensiones y Base64
A05	Security Misconfiguration	Mover archivos sensibles fuera del directorio publico
A04	Insecure Design (id Imágen)	No exponer información interna innecesaria
A04	Insecure Design (validar email)	Validar con <code>filter_var()</code>
A04	Insecure Design (Verificar que la informacion sea base64 valida)	El código corregido toma el string 'data' y lo asigna a <code>base64_str</code> .
A03	SQL Injection (Registrar)	Consultas preparadas
A03	SQL Injection (Login)	Consultas preparadas

A01:2021 - Broken Access Control (Verificación de usuarios)

Código Antiguo:

```
241 // Buscar imagen y enviarla
242 try {
243     $R = $db->exec('Select name,ruta from Imagen where id = '.$idImagen);
244 }
```

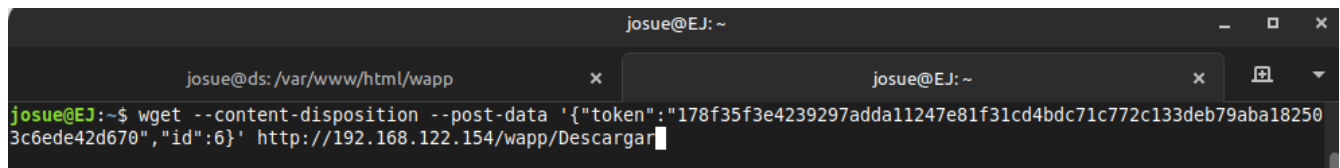
Código Corregido:

```
229
230 // ===== CORRECCIÓN: A01:2021 - Broken Access Control (Verificación de usuarios) ===== //
231 try {
232     $image = $db->exec('SELECT i.name, i.ruta
233                        FROM Imagen i
234                        JOIN AccesoToken at ON i.id_Usuario = at.id_Usuario
235                        WHERE i.id = ? AND at.token = ?',
236                        [$data['id'], $data['token']]);
237
238     if (empty($image)) {
239         echo '{"R":-4,"error":"Imagen no encontrada o no tienes permisos"}';
240         return;
241     }
242 // =====
```

Explicación: Lo que nos permite este código es implementar la seguridad y comprueba que el usuario tiene permisos para ver la imagen usando su token aun así se sepa el id no podrá hacer la descarga sin el token. Se implementa una consulta separada usando “?” Lo que no permite evitar inyecciones de código malicioso. Se utiliza un marcador de posición “?” que nos permite que los valores de la consulta se inserten después y no al momento.

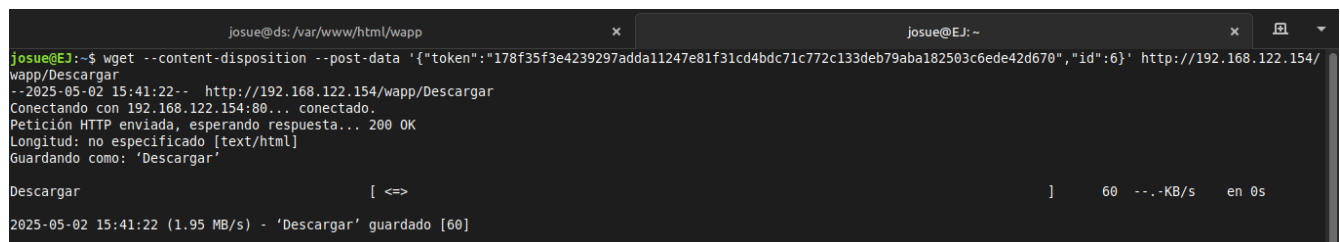
Comprobación:

1- Colocamos el token del usuario para descargar la imagen que el mismo subió, en este caso se coloca un id de imagen que no coincide con el del usuario.



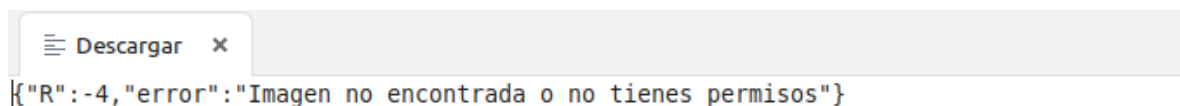
```
josue@EJ: ~
josue@ds: /var/www/html/wapp
josue@EJ:~$ wget --content-disposition --post-data '{"token":"178f35f3e4239297adda11247e81f31cd4bdc71c772c133deb79aba182503c6ede42d670","id":6}' http://192.168.122.154/wapp/Descargar
```

2- Al momento de ejecutar el comando no descargar un archivo de texto donde nos indica lo siguiente:



```
josue@EJ:~$ wget --content-disposition --post-data '{"token":"178f35f3e4239297adda11247e81f31cd4bdc71c772c133deb79aba182503c6ede42d670","id":6}' http://192.168.122.154/wapp/Descargar
--2025-05-02 15:41:22-- http://192.168.122.154/wapp/Descargar
Conectando con 192.168.122.154:80... conectado.
Petición HTTP enviada, esperando respuesta... 200 OK
Longitud: no especificado [text/html]
Guardando como: 'Descargar'

Descargar [ <> ] 60 --.-KB/s en 0s
2025-05-02 15:41:22 (1.95 MB/s) - 'Descargar' guardado [60]
```



```
Descargar x
{"R":-4,"error":"Imagen no encontrada o no tienes permisos"}
```

A02:2021 - Cryptographic Failures(Usu de MD5)

Código Antiguo:

```
64 try {
65     $R = $db->exec('insert into Usuarios values(null, "'.$jsB['uname'].'", "'.$jsB['email'].'", md5("'"$jsB['password']."'"))');
66 } catch (Exception $e){
```

Código Corregido:

```
60
61 // ===== CORRECCIÓN: A02:2021 - Cryptographic Failures(Usu de MD5) ===== //
62 try {
63
64     $hash = password_hash($data['password'], PASSWORD_BCRYPT);
65     $stmt = $db->prepare('INSERT INTO Usuarios VALUES (null, ?, ?, ?)');
66     $stmt->execute([$data['uname'], $data['email'], $hash]);
67     echo '{"R":0}';
68 }
69 // =====
```

Explicación:

Para generar los password se utilizo un almacenamiento seguro utilizando password_hash + password_bcrypt que es un algoritmo moderno y resistente a ataques de fuerza bruta y el cual también utiliza una consulta separada usando "?" la cual nos permite evitar ataques de Injection de código y manejo estructurado de errores la cual captura las excepciones sin detalles sensibles. Y la cual también se cambian los parámetros de la función getToken() con funciones criptográficas mas segura y genera password impredecibles esto quiero decir que un atacante se le va hacer muy difícil descifrar la contraseña.

Función getToken() corregida:

```
8
9 // ===== CORRECCIÓN 1 (Token) =====//
10 function getToken() {
11     $part1 = bin2hex(random_bytes(10));
12     $part2 = bin2hex(random_bytes(16));
13     $part3 = bin2hex(random_bytes(10));
14     return $part1 . $part2 . $part3;
15 }
```

Comprobación:

En este caso podemos ver como en múltiples pruebas registramos usuarios a la base de datos pero en este caso se cambio la cryptation de las contraseña en ve de usar md5 usamos password_hash(), el usuario 5 y 6 son los que tienen la nueva cryptation y ya no están usando md5 porque es menos inseguro.

```
MariaDB [webapps]> select * from Usuarios;
```

id	uname	email	password
1	usuario1	correo@srv.com	827ccb0eea8a706c4c34a16891f84e7b
2	usuario2	correo2@srv.com	827ccb0eea8a706c4c34a16891f84e7b
4	usuario3	correo3@srv.com	827ccb0eea8a706c4c34a16891f84e7b
5	usuario4	correo4@srv.com	827ccb0eea8a706c4c34a16891f84e7b
6	usuario5	correo5@srv.com	\$2y\$10\$0Rx6Rp.GEJVosg04ucoDFeIR0HsSnbPZPrCDK/4V9QB3/vmIryi.
7	usuario6	correo6@srv.com	\$2y\$10\$AKVQqJa4exbDA0WA6i.RT0pVE5vm.zqKMwp2LmP9V1GbcZC8qhHD0

A04:2021 - Insecure Design (Validar Extencion)

Código Antiguo:

```
178
179     $id Usuario = $R[0]['id Usuario'];
180     file_put_contents('tmp/'.$id_Usuario,base64_decode($jsB['data']));
181     $jsB['data'] = '';
```

Código Corregido:

```
144 // ===== CORRECCIÓN: A04:2021 - Insecure Design (Validar Extencion) ===== //
145 $allowed_extensions = ['png', 'jpg', 'jpeg']; // Extensiones permitidas
146 $ext = strtolower(pathinfo($data['name'], PATHINFO_EXTENSION));
147 if (!in_array($ext, $allowed_extensions) || $ext !== strtolower($data['ext'])) {
148     die('{ "R": -4 }');
149 }
150 // =====
```

Explicación:

Este código nos permita la filtración de archivos que aparte de que sean en base64 también del tipo de archivo que este se envía desde el servidor y verifica que el archivo sea válido con la extensión en formato de imagen ya sea .png .jpg .jpeg evitando que contengan algún script se utiliza `pathinfo` para traer la extensión y `strtolower` que evita la malformación de extensiones como .jPg .pNg y evitar que el atacante filtre extensiones no válidas.

Comprobación:

En este caso para evitar que archivos se filtren al servidor y contengan algún script o así se optó por permitir solo extensiones como .png .jpg .jpeg .gif .webp que son formatos comunes de imagen.

```
josue@EJ:~/Documentos/josue/ITSJR/TEMAS DE CIBERSEGURIDAD/UNIDAD 3/file_json$ curl -v -X POST -H "Content-Type: application/json" -d @file.json http://192.168.122.154/wapp/Imagen
Note: Unnecessary use of -X or --request, POST is already inferred.
* Trying 192.168.122.154:80...
* Connected to 192.168.122.154 (192.168.122.154) port 80
> POST /wapp/Imagen HTTP/1.1
> Host: 192.168.122.154
> User-Agent: curl/8.5.0
> Accept: */*
> Content-Type: application/json
> Content-Length: 69214
>
* We are completely uploaded and fine
< HTTP/1.1 200 OK
< Date: Fri, 02 May 2025 21:58:33 GMT
< Server: Apache/2.4.62 (Debian)
< X-Powered-By: Fat-Free Framework
< X-Frame-Options: SAMEORIGIN
< X-XSS-Protection: 1; mode=block
< X-Content-Type-Options: nosniff
< Pragma: no-cache
< Cache-Control: no-cache, no-store, must-revalidate
< Expires: Thu, 01 Jan 1970 00:00:00 +0000
< Vary: Accept-Encoding
< Transfer-Encoding: chunked
< Content-Type: text/html; charset=UTF-8
<
* Connection #0 to host 192.168.122.154 left intact
{"R": -4}josue@EJ:~/Documentos/josue/ITSJR/TEMAS DE CIBERSEGURIDAD/UNIDAD 3/file_json$
```

Sucedio esto porque en el archivo file.json no especifico bien la extensión de mi imagen:

```
1 {
2     "token": "178f35f3e4239297adda11247e81f31cd4bdc71c772c133deb79aba182503c6ede42d670",
3     "name": "m.pdf",
4     "data": "/9j/4AAQSkZJRgABAQEAyABgAAD/2wBDAAAGBGgcBGQgHBWcJCQgKDBUODAsLDBKSEw8VHsgHx4bHR
5     "ext": "pdf"
6 }
```

A04:2021 - Insecure Design (Id imagen)

Código Antiguo:

```
192
193     echo "{\R\":0,\D\":\".$idImagen.\"}";
194
```

Código Corregido:

```
193 // ===== CORRECCIÓN: A04:2021 - Insecure Design (Id imagen) ===== //
194     echo '{"R":0,"D":"Imagen subida correctamente"}';
195 // =====
```

Explicación: El id es visible cualquier atacante puede hacer un ataque de fuerza bruta y lo que se evita es que en la salida en ves de mostrar el .idImagen como se muestra en el código de la api web vulnerable y el otro código solo lanzamos un mensaje de “Imagen subida correctamente” y evitamos lo siguiente: *fuerza bruta con /Descargar?id=1, 2, 3...).*

Comprobación:

En el caso del id de la imagen solo se opto por quitar la variable y simplemente cuando la imagen fuera subida al servidor solo se mostrara una leyenda como la que se muestra:

```
josue@EJ: ~/Documentos/josue/ITSJR/TOPICOS DE CIBERSEGURIDAD/UNIDAD 3/file_json
josue@EJ:~/Documentos/josue/ITSJR/TOPICOS DE CIBERSEGURIDAD/UNIDAD 3/file_json$ curl -v -X POST -H "Content-Type: application/json" -d @file.json http://192.168.122.154/wapp/Imagen
Note: Unnecessary use of -X or --request, POST is already inferred.
* Trying 192.168.122.154:80...
* Connected to 192.168.122.154 (192.168.122.154) port 80
> POST /wapp/Imagen HTTP/1.1
> Host: 192.168.122.154
> User-Agent: curl/8.5.0
> Accept: */*
> Content-Type: application/json
> Content-Length: 69214
* We are completely uploaded and fine
< HTTP/1.1 200 OK
< Date: Fri, 02 May 2025 22:21:36 GMT
< Server: Apache/2.4.62 (Debian)
< X-Powered-By: Fat-Free Framework
< X-Frame-Options: SAMEORIGIN
< X-XSS-Protection: 1; mode=block
< X-Content-Type-Options: nosniff
< Pragma: no-cache
< Cache-Control: no-cache, no-store, must-revalidate
< Expires: Thu, 01 Jan 1970 00:00:00 +0000
< Vary: Accept-Encoding
< Transfer-Encoding: chunked
< Content-Type: text/html; charset=UTF-8
* Connection #0 to host 192.168.122.154 left intact
{"R":0,"D":"Imagen subida correctamente"}josue@EJ:~/Documentos/josue/ITSJR/TOPICOS DE CIBERSEGURIDAD/UNIDAD 3/file_json$
```

En la base de datos también cambia el nombre de la imagen para que no se guarde con el id:

12	m	img/img_6815385bc66128.79035343.jpg	7
13	m	img/img_6815395958e575.20716671.jpg	7
14	m	img/img_68154570d94c63.96819016.jpg	7

A04:2021 - Insecure Design (Validación de email)

Código Antiguo:

```
56 //////////////////////////////////////////////////
57 $R = array_key_exists('uname',$jsB) && array_key_exists('email',$jsB) && array_key_exists('password',$jsB);
58 // TODO chequear si estan vacios los elemento del json
59 // TODO control de error
```

Código Corregido:

```
54 // ===== CORRECCIÓN: A04:2021 - Insecure Design (Validación de email) ===== //
55 if (!filter_var($data['email'], FILTER_VALIDATE_EMAIL)) {
56     die('{"R":-6}');
57 }
58 // =====
```

Explicación: En primer lugar lo que se trata de evitar es el almacenar email inválidos corrompe la integridad de los datos de los usuarios y del servidor en general para ello se utiliza algo un poco moderno que es `filter_var()` + `FILTER_VALIDATE_EMAIL` que es un filtro que nos permite validar la estructura de un correo electrónico valido y este va hacer aceptado o rechazado por `FILTER_VALIDATE_EMAIL`.

Comprobación:

En este caso validamos lo que es el email para evitar que se envíe caracteres que no coincidan con la estructura de un correo electrónico:

```
jose@ds:/var/www/html/wapp
jose@EJ:~$ curl -v -X POST -H "Content-Type: application/json" -d '{"uname":"usuario7","email":"<12correo7@srv.com>","password":"12345"}' http://192.168.122.154/wapp/Registro
Note: Unnecessary use of -X or --request, POST is already inferred.
* Trying 192.168.122.154:80...
* Connected to 192.168.122.154 (192.168.122.154) port 80
> POST /wapp/Registro HTTP/1.1
> Host: 192.168.122.154
> User-Agent: curl/8.5.0
> Accept: */*
> Content-Type: application/json
> Content-Length: 73
>
< HTTP/1.1 200 OK
< Date: Fri, 02 May 2025 22:28:39 GMT
< Server: Apache/2.4.62 (Debian)
< X-Powered-By: Fat-Free Framework
< X-Frame-Options: SAMEORIGIN
< X-XSS-Protection: 1; mode=block
< X-Content-Type-Options: nosniff
< Pragma: no-cache
< Cache-Control: no-cache, no-store, must-revalidate
< Expires: Thu, 01 Jan 1970 00:00:00 +0000
< Vary: Accept-Encoding
< Transfer-Encoding: chunked
< Content-Type: text/html; charset=UTF-8
<
* Connection #0 to host 192.168.122.154 left intact
{"R":1}jose@EJ:~$
```

En este caso tiene que aparecer para que se pueda registrar en la base de datos:

```
<
* Connection #0 to host 192.168.122.154 left intact
{"R":0}jose@EJ:~$
```

Registro en la base de datos con un correo valido:

7	usuario6	correo6@srv.com	\$2y\$10\$AKVQqJa4exbDA0WA6i.RT0pVE5vm.zqKMwp2lmP9V1GbcZC8qhHD0
8	usuario7	correo7@srv.com	\$2y\$10\$lgaGEldemqfiQtri4WxpV0VX3m1lz.rZA7H20ku1EDWvcWr4RPYGj

A04:2021 - Insecure Design (Verificar que la informacion sea base64 valida)

Código Antiguo:

```
178
179     $id_usuario = $R[0]['id_usuario'];
180     file_put_contents('tmp/'.$id_usuario,base64_decode($jsB['data']));
181     $jsB['data'] = '';
```

Código Corregido:

```
154
155 // = CORRECCIÓN: A04:2021 - Insecure Desing - (Verificar que la informacion sea base64 valida) = //
156 $base64_str = $data['data'];
157 if (!preg_match('/^[a-zA-Z0-9\\/\r\n+]*={0,2}$/', $base64_str)) {
158     die('{"R": -5}');
159 }
160 // ===== //
```

Explicación: En el código está decodificándolo y guardándolo en un archivo lo que puede ser peligroso si no se valida adecuadamente antes de subir al servidor. El código corregido toma el string 'data' y lo asigna a base64_str con !preg_match para verificar si el archivo no contiene algo fuera de lo común o no esta bien descodificado en base64 la imagen y aun así se rechaza el archivo para que no se suba al servidor como también la longitud del string en base64.

Comprobación:

En esta parte comprobamos que lo que estamos enviando por el archivo file.json en este caso es de decodificación en base64 de la imagen que nosotros declaramos dentro del archivo con la variable "data" si esta mal decodificado el archivo en base64 de la imagen este no lo va a aceptar y no se va a enviar la imagen y el error seria {"R": -5}:

```
josue@EJ: ~/Documentos/josue/ITSJR/TEMAS DE CIBERSEGURIDAD/UNIDAD 3/file_json
josue@EJ:~/Documentos/josue/ITSJR/TEMAS DE CIBERSEGURIDAD/UNIDAD 3/file_json$ curl -v -X POST -H "Content-Type: application/json" -d @file.json http://192.168.122.154
/wapp/Imagen
Note: Unnecessary use of -X or --request, POST is already inferred.
* Trying 192.168.122.154:80...
* Connected to 192.168.122.154 (192.168.122.154) port 80
> POST /wapp/Imagen HTTP/1.1
> Host: 192.168.122.154
> User-Agent: curl/8.5.0
> Accept: */*
> Content-Type: application/json
> Content-Length: 69194
>
* We are completely uploaded and fine
< HTTP/1.1 200 OK
< Date: Fri, 02 May 2025 22:39:13 GMT
< Server: Apache/2.4.62 (Debian)
< X-Powered-By: Fat-Free Framework
< X-Frame-Options: SAMEORIGIN
< X-XSS-Protection: 1; mode=block
< X-Content-Type-Options: nosniff
< Pragma: no-cache
< Cache-Control: no-cache, no-store, must-revalidate
< Expires: Thu, 01 Jan 1970 00:00:00 +0000
< Vary: Accept-Encoding
< Transfer-Encoding: chunked
< Content-Type: text/html; charset=UTF-8
<
* Connection #0 to host 192.168.122.154 left intact
{"R": -5}josue@EJ:~/Documentos/josue/ITSJR/TEMAS DE CIBERSEGURIDAD/UNIDAD 3/file_json$
```


Explicación de las Injection (SQL): Los parámetros (?) se envían al servidor de BD de forma independiente a la consulta, imposibilitando la interpretación de código malicioso, al separar la consulta con (?) que esto son marcadores de posición los valores se envían después del array y así la base de datos trata los valores como datos y no como código SQL.

A03:2021 - Injection (SQL en Registrar)

Código Antiguo:

```
64     try {
65         $R = $db->exec('insert into Usuarios values(null,
66                     "'. $jsB['uname']. '", "'. $jsB['email']. '",
67                     md5('". $jsB['password']. "')');
68     }
```

Código Corregido:

```
65     // ===== CORRECCIÓN: A03:2021 - Injection (SQL REGISTRAR) ===== //
66     $stmt = $db->prepare('INSERT INTO Usuarios VALUES (null, ?, ?, ?)');
67     $stmt->execute([$data['uname'], $data['email'], $hash]);
68     // =====
```

A03:2021 - Injection (SQL en Imagen)

Código Antiguo:

```
171
172     try {
173         $R = $db->exec('select id_usuario from AccesoToken where token = "'. $TKN. '"');
174     } catch (Exception $e) {
175         echo '{"R":-2}';
176         return;
177     }
```

Código Corregido:

```
146     // ===== CORRECCIÓN: A03:2021 - Injection (SQL) ===== //
147     $user = $db->exec('SELECT id_usuario FROM AccesoToken WHERE token = ?', [$data['token']]);
148     if (empty($user)) {
149         die('{"R":-3}');
150     }
151     // =====
```

A03:2021 - Injection (SQL en Login)

Código Antiguo:

```
107     try {
108         $R = $db->exec('Select id from Usuarios where uname =
109                     "'. $jsB['uname']. '" and password = md5
110                     ('". $jsB['password']. "')');
111     }
```

Código Corregido:

```
95     // ===== CORRECCIÓN: A03:2021 - Injection (SQL LOGIN) ===== //
96     try {
97         $stmt = $db->prepare('SELECT id, password FROM Usuarios WHERE uname = ?');
98         $stmt->execute([$data['uname']]);
99         $user = $stmt->fetch();
100
101         if (!$user || !password_verify($data['password'], $user['password'])) {
102             die('{"R":-3}');
103         }
104     }
105     // =====
```

A05:2021 - Security Misconfiguration (Archivos sensibles)

Código Antiguo:

```
211 function($f3) {
212     $dbcnf = loadDatabaseSettings('db.json');
213     $db=new DB\SQL(
214         'mysql:host=localhost;port='.$dbcnf['port'].';dbname='.$dbcnf['dbname'],
215         $dbcnf['user'],
216         $dbcnf['password']
217     );
```

Código Corregido:

```
199
200 // ===== CORRECCIÓN: A05:2021 - Security Misconfiguration (Archivos sensibles) ===== //
201 $dbcnf = loadDatabaseSettings('config/db.json');
202 // =====
```

Explicación:

En esta parte lo que se implemento fue que el archivo db.json que es el que contiene prácticamente el inicio de sesión fue movido a otra ubicación en “config” para evitar que la informacion sensible se muestre directamente desde el navegador Ejemplo: `http://0.0.0.0/wapp/db.json`. Y prácticamente con esto evitamos que archivos que están en la posición publica de los archivos se visualicen y evitar un ataque.

Comprobación:

En este caso se opto por guardar el archivo de db.json que contiene mi inicio de sesión de mi base de datos en una carpeta llamada: config para evitar que archivos sensibles se visualices en el navegador y exponer la informacion.



`https://192.168.122.154/wapp/db.json`

Not Found

HTTP 404 (GET /db.json)

Y para evitar que los archivos sensibles que están dentro de directorios como en el caso de la api web que son img, tmp config y vendor.

```
josue@ds:/var/www/html/wapp$ ls
composer.json  composer.lock  config  img  index.php  tmp  vendor
josue@ds:/var/www/html/wapp$
```

Se configuro el archivo .htaccess donde podemos evitar ataques que lean configuraciones sensibles lo que se hace es redireccionar los archivos declarados con repuesta a R=404 y así evitamos que cuando alguien quiero ver algo dentro de un archivo desde el navegador aparezca lo siguiente.



Y solamente configuramos lo siguiente en el archivo .htaccess:

```
GNU nano 7.2
RewriteEngine On

RewriteRule ^(app|dict|ns|tmp|img|vendor|config)\|\.ini$ - [R=404,L]

RewriteCond %{REQUEST_FILENAME} !-l
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule .* index.php [L,QSA]
RewriteRule .* - [E=HTTP_AUTHORIZATION:%{HTTP:Authorization},L]
```