PHP y cURL

Suena incoherente: usamos **curl** en nuestra línea de comandos frecuentemente para diversidad de tareas *pero no sabíamos que el lenguaje PHP tiene su propia versión llamada, cómo no,* ***curl***. El asunto es cómo se implementa y qué podemos hacer, nosotros en la línea de comandos lo acompañamos en el trabajo con **grep** pero en **PHP** la cosa es un poco diferente, ¡vamos a averiguar cómo!

Introducción.

Exactamente cómo implementaron **cURL** en **PHP** no lo sabemos, tal vez **PHP** haga llamado directo al **curl** y recoja sus resultados, sirviendo entonces como intermediario solamente... No tiene sentido pero si analizamos que muchos sitios web compartidos con otros dominios tienen **PHP** pero a uno no lo dejan ingresar por la línea de comandos *entonces ¡vaya que si vale la pena el trabajo intermedio!*

Por otra parte puede ser que especulamos demasiado y los buenos desarrolladores de **PHP** tienen su propia versión... *¡pero recordad que trabajamos con Software Libre que permite la modificación y reutilización del código!* Pero echemos una leve ojeada al comando **curl.**

Comando curl.

Ya sabemos como va es esto del software libre: hay más de 1400 colaboradores en **curl** y el proyecto está alojado en GitHub desde el año 2010; aceptan contribuciones bajo la forma de "pull requests": en resumen uno copia y trabaja con esa copia privada modificandola -y mejorandola, por supuesto- y devolviendo el trabajo al original para su aprobación. No se garantiza que se acepte toda la propuesta, e incluso ellos podrán tomar ciertas cosas propuestas y otras no, o en el mejor de los casos todo vaya bien y nos convirtamos en el colaborador 1401... y así haremos nuestra contribución al Patrimonio Tecnológico de la Humanidad.

https://twitter.com/digilizar/status/799663277395628033

Pero ¿quiénes desarrollan **curl? Daniel Stenberg** -sueco, habitante de Estocolmo- es el principal desarrollador y comenzó el proyecto por allá en 1996 basado en el trabajo de un programador brasileño llamado **Rafael Sagula**. Esta sencilla herramienta herramienta carioca fue bautizada como **httpget** con unas cuantas centenares de líneas y Stenberg las amplió para liberarla en 1997 con el nuevo nombre de "HTTP proxy support". Pero con el advenimiento de "nuevos" protocolos al proyecto tales como GOPHER y FTP pronto dejó de tomar sentido llamarlo solamente "HTTP"... así que ese mismo año vio la luz la versión número dos.

Para 1998 más protocolos y capacidades fueron agregados así que volvió a cambiar de nombre

para la versión 4.0 -manteniendo la numeración de versiones- así que el 20 de marzo de ese año marca el nacimiento formal de **curl.** El nuevo nombre hace alusión al programa del lado del cliente, de allí la letra "c" inicial. Las otras tres letras os lo podéis imaginar ya: **Uniform Resource Locator -URL-** o Localizador de Recursos Normalizados.

El mismo Stenberg reconoce que hay muchos proyectos con el mismo nombre **curl** pero para la época que ellos lanzaron la versión cuatro no había (o conocían) de otros proyectos con el mismo nombre. Es por ello que nosotros al principio especulamos del origen de **curl** en el lenguaje **PHP**, pero es que ¡Incluso existe un curl desarrollado en software privativo! (no le haremos publicidad pues no nos pagan por ello, buscadlo vosotros mismos con **DuckDuckGo**). Tan famoso es que ya se considera como verbo en idioma inglés y muchos creen que es un protocolo pues lo tratan como tal pero ya sabemos que en realidad es una navaja suiza con muchísimas funciones.

curl para la línea de comandos.

En principio el comando **curl** fue desarrollado para *scripts* ya que sus entradas y salidas utilizan las ya famosas *stdin* y *stdout*. No ampliaremos más porque en este blog hallaréis información sobre su uso con la línea de comandos. El artículo que motivó la publicación de esta entrada está en el siguiente "tuit" y precisamente es para la línea de comandos -y pretendemos que corra en **PHP curl**-:

https://twitter.com/ks7000/status/869140408672059393

También hemos recopilado con el paso del tiempo otros usos notables del curl en la línea de comandos:

https://twitter.com/ks7000/status/853346982982815744

https://twitter.com/ks7000/status/853228108748472320

curl para su uso como librería.

A partir del año 2000 fueron desarrolladas, **o mejor dicho, el código existente fue migrado completamente como librerías y a partir de allí se le hizo una interfaz de usuario para la línea de comandos.** Está escrito en lenguaje C y aún hoy en día dichas bases siguen sustentando el proyecto con la ventaja que se puede reutilizar en otros proyectos y compilarlos todos juntos. *Seguimos sospechando que precisamente eso hicieron en **PHP** pero esa es nuestra humilde opinión que como reafirmamos: en software libre este comportamiento es un honor a su filosofía de desarrollo:* ejecutar, estudiar, distribuir, mejorar y redistribuir.

Versiones actuales y desarrolladores estrellas.

Nosotros en nuestro GNU/Linux Ubuntu 16 tenemos instalada la versión 7.40.00 pero en realidad la última versión estable es la 7.54.0.

Si bien son muchísimos los colaboradores, es justo mencionar los que más han contribuido en los últimos años de una manera marcada y constante:

- Daniel Stenberg.
- Steve Holme.
- Jay Satiro.
- Dan Fandrich.
- Marc Hörsken.
- Kamil Dudka.
- Alessandro Ghedini.
- Yang Tse.
- Günter Knauf.
- Tatsuhiro Tsujikawa.
- Patrick Monnerat.
- Nick Zitzmann.

Disipación de toda duda.

En este punto de nuestra investigación nuestras sospechas se hacen realidad: uno de los primero lenguajes en adoptar la librería **curl** es precisamente **PHP** el cual motoriza un 25% de las páginas web a nivel mundial. El extracto , en inglés, reposa en el libro electrónico -liberado con licencia "Creative Commons"- "Everything about curl":

> The libcurl binding for PHP was one of, if not the, first bindings for libcurl to really catch on and get used widely. It quickly got adopted as a default way for PHP users to transfer data and as it has now been in that position for over a decade and PHP has turned out to be a fairly popular technology on the Internet (recent numbers indicated that something like a quarter of all sites on the Internet uses PHP).
> A few really high-demand sites are using PHP and are using libcurl in the backend. Facebook and Yahoo are two such sites.

La traducción hecha por nosotros al idioma castellano:

El software de enlace (librerías) de **curl** para **PHP** fue uno, cuidado sino, el primero de los enlaces que realmente captura y lo usa ampliamente. Rápidamente fue adoptado como una manera por defecto para transferir datos y se ha mantenido en esa posición por más de una década mientras tque **PHP** se ha convertido en una tecnología bastante popular en internet (cifras recientes indican que algo asó como una cuarta parte de todos los sitios web en internet utilizan **PHP**).

Realmente unos pocos sitios web de alta demanda están usando **PHP** acompañado de las librerías de enlace de **curl** corriendo entre bamabalinas. Facebook y Yahoo son dos de tales sitios.

Protocolos soportados por curl.

HTTP, HTTPS, FTP, FTPS, GOPHER, TFTP, SCP, SFTP, SMB, TELNET, DICT, LDAP, LDAPS, FILE, IMAP, SMTP, POP3, RTSP y RTMP (esos son todos, por ahora -y vienen más, no se detiene el proyecto-).

PHP curl.

Ahora si que pasamos a hablar del software que nos ocupa en este vuestro sitio web de compartición del conocimiento. Como ya hemos hablado bastante de historia y teoría pasamos directamente a describir cómo trabajar con **PHP curl:**

1. Inicializar curl con **curl_init().**
2. Pasarle los parámetros - esencial es la URL - con **curl_setopt().**
3. Retribuir y mostrar -o guardar- con **curl_exec().**
4. Cerrar y liberar recursos con **curl_close().**

Nuestro primer ejemplo práctico.

Lo siguiente que haremos es abrir una ventana terminal, tomar nuestro editor de texto favorito y escribir el siguiente código para ser guardado en un archivo que llamaremos **php_curl.php**:
[cc lang="php" tab_size="2" lines="80
[/cc]

Por supuesto, llamad vuestro archivo como queráis, guardadlo en vuestra carpeta donde ejecute

vuestro servidor PHP y dadle los permisos de lectura y ejecución necesarios. En la primera línea inicializamos, en la segunda línea le pasamos la dirección web deseada -un dominio que devuelve nuestra dirección IP asignada por nuestro ISP-, en la tercera línea lo ejecutamos y en la cuarta línea cerramos y liberamos recursos. Una explicación más detallada a continuación.

Inicialización.

Debemos crear una instancia y guardarla para futuras referencias, este objeto basado en **curl** nosotros lo llamamos **$objCurl** y este nombre es el que debemos pasar a los otros comandos. El comando **curl_init()** solamente acepta un parámetro, la URL que es un dato imprescindible, tal vez debido a ser tan importante fue el único que establecieron en este comando. Aunque en el ejemplo no lo colocamos por razones didácticas, de ahora en adelante "para que no se nos olvide" lo estableceremos siempre en la primera línea del *script* o guion del programa.

Configuración.

Este es el comando más denso, notad que le damos el nombre de *configuración* porque su nombre así lo sugiere: **curl_setopt()**, osea "setopt" -> "set options". Acepta tres parámetros, separados por comas:

- Primero debemos indicarle el objeto que contiene la inicialización de **curl**, en nuestro caso la variable **$objCurl.**
- Segundo le pasaremos el tipo de valor que le pasaremos en el tercer parámetro, en nuestro caso la URL y la constante que lo define -nombrada de manera nemotécnica- es **CURLOPT_URL**.
- El tercer parámetro es el valor en sí mismo de lo que indicamos en el segundo parámetro.

Debemos indicaros que usamos una sola línea para configurar, pero pronto veremos que esta parte es la más abultada por la infinidad de datos y tipos de datos que podemos pasar. *Es mejor que vayaís preparando para aprender que esta sección siempre será multilínea y que debemos indentarla y escribirla lo más explícito posible para corregir a futuro de forma fácil nuestro código.*

Ejecución.

Simplemente le decimos a **PHP** que hemos terminado de establecer lo que queremos obtener -*o lo que queremos enviar, ya veremos más adelante*- y acepta un parámetro que sigue siendo el objeto que hemos creado y configurado. En este punto ya os recordaremos que estamos trabajando con funciones y este comando devuelve *verdadero* o *falso* para indicar si tuvo éxito (para comparar la respuesta usaremos siempre "==" para estar seguros del tipo de variable y el resultado obtenido). En el ejemplo no colocamos ninguna variable que reciba el resultado del que hablamos como función ¿dónde está lo que nos interesa? En realidad el valor numérico (verdadero o falso) es lo que podemos guardar en una variable, el resto del resultados se va al **stdout** y eso será lo que

enviaremos al navegador web que solicitó nuestro primer guión PHP sobre curl. Debido a esto recibiremos el mismo código HTML de la página que estamos solicitando (pero con los enlaces web absolutos cambiados).

Si probáis con descargar diferentes páginas, unas funcionarán, otras no, todo depende de si la página es estática, dinámica, si tiene JavaScript, en fin, cantidad de cosas que pueden NO ser compatibles con **PHP curl**: por eso debemos aprender las muchísimas opciones de configuración que nos sean útil en cada trabajo que se nos presente para ganarnos así el pan nuestro de cada día como Dios manda.

Liberación de recursos.

Os podrá sonar que nos contradecimos con lo siguiente: la función **curl_close()** se encarga de cerrar, como su nombre indica *pero no devuelve resultado acerca del trabajo que le mandamos a realizar. Consideramos que esto es un "bug" porque a la hora de detectar dónde fugan recursos en nuestro servidor deberemos recurrir a otras herramientas apartes de PHP, pudiendo ser evitado esto con una sencilla rutina de nuestra parte e indicar que está sucediendo y qué está mal.* Ah, perdón, casi lo olvidamos: el parámetro único que acepta es el objeto que creamos -y queremos destruir- con **curl_init().**

Nuestro primer ejemplo práctico pero mejorado.

Un sencillísima rutina de control de errores nos puede ahorrar muchísimos dolores de cabeza.

> Nota: hay mejores métodos para el manejo de excepciones, pero aprendamos poco a poco. Ah, y nosotros lo llamamos "control de errores" y eso tampoco es el nombre correcto (excepciones) pero así nos abstraemos.

Por ello reescribiremos nuestro ejemplo pero con condicionales **if~else**:

```
[cc lang="php" tab_size="2" lines="80" widht="10%"]
$objCurl = curl_init("http://www.soporteweb.com/");
if ($objCurl == true) {
$resul = curl_exec ($objCurl);
if ($resul == true){
curl_close ($objCurl);
} else {
echo "No se pudo ejecutar PHP curl
";
}
} else {
echo "No se pudo inicializar PHP curl.
```

```
";
}
?>[/cc]
```

Notad que pasamos de una buena vez el URL en en el inicio con **curl_init().** Además guardamos el valor del resultado en una variable, evaluamos el valor que tiene y ejecutamos o mostramos un mensaje apropiado *según la respuesta obtenida*. Muchos programadores y programadoras piensan que hacer nuestro software de esta manera, aparte de ayudarnos a nosotros mismo, abre las puertas a los *hackers* cuando se presentan excepciones -o errores, como les llamamos- porque "develan mucho". **Ese razonamiento es válido en el software privativo pero en el software libre *no tiene asidero alguno porque cualquier hacker tiene acceso al código fuente, así que hagamos la depuración fácil para nosotros mismos.***

Guardando el resultado en una "variable aparte".

Como explicamos **PHP curl** envía al *stdout* la respuesta y asu vez eso lo pasamos al navegador, ¿qué tal si analizamos primero lo que recibimos y luego lo reenviamos? *Por ejemplo, podríamos "acomodar" los enlaces absolutos de las imágenes que contiene la página web que llamamos (URL).* Para ello vamos a emplear **culr_setopt()** con CURLOPT_RETURNTRANSFER establecido en el valor 1, mirad:

```
[cc lang="php" tab_size="2" lines="80" widht="10%"] $objCurl =
curl_init("http://www.soporteweb.com");
if ($objCurl == true) {
curl_setopt ($objCurl, CURLOPT_RETURNTRANSFER, 1);
$resul = curl_exec ($objCurl);
if ($resul == true){
curl_close ($objCurl);
print "Nuestra direccion IP es: ";
 print $resul;
} else {
echo "No se pudo ejecutar PHP curl
";
}
} else {
echo "No se pudo inicializar PHP curl.
";
}
?>[/cc]
```

Y así "manipulamos" el resultado con el texto que coloreamos en verde; ahora vamos un paso más allá. 8-)

Guardando el resultado en un archivo.

Aunque ya tengamos el resultado deseado en una "variable" supongamos que estamos creando un robot (o *bot* como los mientan ahora en este siglo) y queremos guardar el resultado en un archivo ¿qué tiene que ver **PHP curl** con esto si ya sabemos el lenguaje **PHP** y sabemos como hacerlo aparte?

Pues resulta que con CURLOPT_FILE en **curl_setopt()** permite guardarlo en un archivo, *pero antes tenemos que abrir el archivo y pasarle la "referencia" a **PHP curl**:*

```
[cc lang="php" tab_size="2" lines="80" widht="10%"] $objCurl =
curl_init("http://www.soporteweb.com/");
if ($objCurl == false) {
print "No se pudo inicializar PHP curl.
";
} else {
$nom_arch = 'PHP_curl.html';
$arch = fopen( $nom_arch , "w");
if ($arch == false ) {
print "No se pudo abrir el archivo '".$nom_arch."'
";
curl_close ($objCurl);
} else {
$resp = curl_setopt ($objCurl, CURLOPT_RETURNTRANSFER, 1);
if ( $resp == false ) {
print "No se pudo establecer CURLOPT_RETURNTRANSFER.
";
curl_close ($objCurl);
} else {
$resp = curl_setopt ($objCurl, CURLOPT_FILE, $arch);
if ( $resp == false ) {
print "No se pudo establecer CURLOPT_FILE.
";
curl_close ($objCurl);
} else {
$resul = curl_exec ($objCurl);
if ($resul == false) {
echo "No se pudo ejecutar PHP curl.
";
} else {
print "El archivo fue guardado con el nombre 'PHP_curl.html'.
";
// curl_close NO devuelve resultado:
```

```
curl_close ($objCurl);
}
}
}
}
}
?>[/cc]
```

> Si tenéis problemas para escribir el archivo en disco REVISAD vuestros derechos de escritura para PHP especialmente para el grupo "www-data".

Así de sencillo, con las correspondientes rutinas de depuración para nosotros. Como tal vez se vea un poco largo hemos coloreado las partes importantes para que sigáis el hilo. ¿Que por qué nos hemos ido por la negación siempre de primero? Pues revisando la función **fopen()** verificamos que si es exitosa devuelve una referencia al archivo *más no un valor de tipo booleno -verdadero-.* ***En cambio si falla la función siempre devuelve un valor boleano falso.*** Lo de seguir usando la negación por defecto es para llevar el mismo estilo parejo al resto del código.

Mejorando la claridad de la sintaxis con ayuda de exit().

Una función "vieja" en **PHP** es la función **die()** que es totalmente equivalente a **exit()** que es más "elegante". El chiste del asunto es emplear la conjunción **or** (operador lógico "o") para unir la acción que queremos realizar con la función **exit()**. Dicha función permite como parámetro una cadena de texto o un entero entre cero y 254 (el 255 está reservado para uso exclusivo de **PHP**). Nos gusta mejor la opción de pasarle una cadena de texto con el mensaje que queremos presentar al usuario ¡o a nosotros mismos!

Otro punto importante es que la función **exit()** aparte de mostrar mensaje y cerrar adecuadamente los objetos en memoria *es que sale inmediatamente y no ejecuta el resto de las líneas que quedan a partir de su llamada.*

Veamos entonces cómo emplearlo en nuestra labor que hoy nos ocupa *(para no alargar tanto cada línea* ***fijaos el uso del punto y coma para separar bloques de código multilíneas****):*

```
                        $objCurl             =             curl_ini
t("http://www.soporteweb.com/")
      or exit("No se pudo inicializar PHP curl.  ");
```

```
$nom_arch = 'PHP_curl.html';
 $arch = fopen( $nom_arch , "w")
or exit("No se pudo abrir el archivo '".$nom_arch."'
");

 curl_setopt ($objCurl, CURLOPT_RETURNTRANSFER, 1)
or exit("No se pudo establecer CURLOPT_RETURNTRANSFER.
");

 curl_setopt ($objCurl, CURLOPT_FILE, $arch)
or exit("No se pudo establecer CURLOPT_FILE.
");

 curl_exec ($objCurl)
or exit("No se pudo ejecutar PHP curl.
");

 print "El archivo fue guardado con el nombre 'PHP_curl.html'.
";

// curl_close NO devuelve resultado:
 curl_close ($objCurl);
?>[/cc]
```

¡Qué gran mejora en la legibilidad! Debemos aclarar que **exit()** de manera implícita llama a los procedimientos de cierre y liberación de recursos de memoria *(destructor* lo nombran en lenguaje C++) así que como ven es una vía muy pragmática para nuestros propósitos.

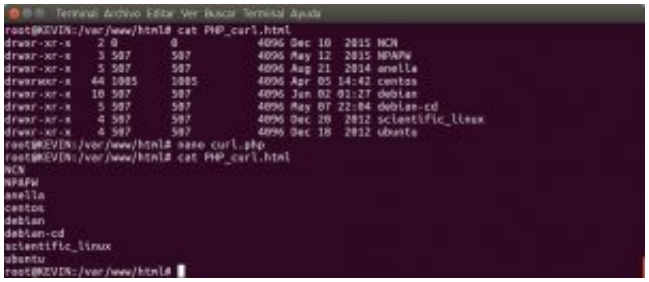
Si queréis probar como funciona la rutina con control de excepciones, colocad una dirección web que no exista y mirad que sucede, ¡experimentad! También os recomendamos probar otros protocolos, por ejemplo FTP en un servidor público como por ejemplo "ftp://ftp.cesca.es/" donde veremos un listado de archivos disponibles para ser descargados.

Probando otros protocolos: FTP.

En el párrafo anterior os sugerimos probar el protocolo FTP y no hubo nada que cambiar en el código que teníamos -aparte de la URL, por supuesto-. Pero **PHP curl** tiene sus opciones y a continuación pasamos a revisar alguna de ellas. La primera que revisaremos limita el listado obtenido de archivos y directorios, osea, no muestra los permisos de cada uno de ellos. La sintaxis en el comando **curl_setopt()** -y todas las opciones utilizan esta función- es la siguiente:

[cc lang="php" tab_size="2" lines="80" widht="10%"]curl_setopt($objCurl,

CURLOPT_FTPLISTONLY, 1)
or exit("No se pudo establecer CURLOPT_FTPLISTONLY");
[/cc]



Si el servidor FTP no fuera público o exigiera una conexión tipo anónimo usaremos esto:

curl_setopt($objCurl, CURLOPT_USERPWD, "anonimo:su@correo-e.com") or exit("No se pudo establecer CURLOPT_USERPWD");[/cc] Con esta novedad de introducir credenciales de conexión nos encontramos con otro tipo de manejo de excepciones: dado el caso las credenciales no funcionen (mala escritura, contraseña vencida, usuario inexistente, etc) dentro de l obejto **PHP** **curl** se guardará en una instancia aparte el resultado del intento de conexión, para ellos contamos con: [cc lang="php" tab_size="2" lines="80" widht="10%"] echo curl_error($objCurl);[/cc] https://twitter.com/souzace/status/868869012414840832

Pasando valores por medio del método POST.

Finalmente llegamos al meollo del asunto: poder pasar a un servidor web una o más variables por medio del método POST a un servidor web. Para propósitos de aprendizaje diremos que es uno de los métodos más populares debido a su cierta privacidad ya que el usuario no puede ver el enlace como en el método GET y tampoco es "cacheable" por los navegadores web.

Primero haremos una rchivo que nos muestre todas las variables que reciba y las liste por pantalla, de manera increíble solo necesita una sola líneas de código (guardar con el nombre de "*curl_post.php*"):

```
    var_dump($_POST);  ?>[/cc]
```

Una vez tengamos este archivo ya sea en nuestro servidor local o en nuestro servidor remoto podremos comenzar a escribir el siguiente script:

```
                    $objCurl                    =                    curl_i
nit("http://localhost/curl_post.php"
)   or exit("No se pudo inicializar PHP curl  ");
```

```
 curl_setopt($objCurl, CURLOPT_POST, 1)
or exit("No se pudo establecer CURLOPT_POST");

//Pasamos las variables que nos interesan al servidor
 curl_setopt($objCurl, CURLOPT_POSTFIELDS, "var1=uno&var2=dos&var3=tres")
or exit("No se pudieron establecer las variables en CURLOPT_POSTFIELDS");

 curl_exec ($objCurl)
or exit("No se pudo ejecutar PHP curl.");

 curl_close ($curl);
?>[/cc]
```

Debido a los mensaje que incluimos en cada **exit()** y con el coloreado del código no hay nada que explicar... excepto unas cuantas acotaciones:

- Las variables que pasamos en el método POST deben estar por pares unidas con el *ampersand* y separadas por medio de un signo de igualdad.
- Los espacios no se permiten, debemos pasar "%20" ***o mejor dicho*** debemos darle un formato especial antes de enviarlo. El comando que nos ayudará será **urlencode()** y para nosotros castellanohablantes es importante representar bien los acentos, etc.

```
[cc lang="php" tab_size="2" lines="80" widht="10%"] //Pasamos las variables que nos interesan al
servidor
$var_post = urlencode("var1=Venezuela&var2=América del Sur&var3=Güigüe");
 curl_setopt($objCurl, CURLOPT_POSTFIELDS, $var_post)
or exit("No se pudieron establecer las variables en CURLOPT_POSTFIELDS");[/cc]
```

Fuera de tema: "Tamper data for Mozilla Firefox".

Adam Judson es un desarrollador que tiene -nos tiene- como usuarios a más de 80 mil personas en su trabajo de programación desde el año 2007. Él desarrolló "Tamper Data" para este popular navegador, una herramienta para, según sus propias palabras "pruebas de seguridad en aplicaciones web por medio de la modificación de los parámetros POST".

Dicha herramienta la podemos instalar en nuestro navegador (requiere reinicio del navegador y no es compatible con "Google Web Accelerator") y con ella podremos visitar cualquier sitio web que utilize envio de datos POST y al activarlo en una ventana aparte mostrará *sin molestar ni influir en nada* el tráfico saliente de nuestro ordenador por medio de nuestro navegador web Firefox. ¿Por qué anuncia pruebas de seguridad? Porque nosotros muy bien sabemos como programadores que el código fuente de nuestras aplicaciones web están a la vista de los usuarios ¿pero que tal lo que no se ve? Pues acá entra "Tamper Data": por más que coloquemos validación HTML5 y JavaScript del lado del cliente, si no utilizamos HTTPS el tráfico entre el usuario y nuestro servidor puede ser modificado por terceros. **Y aunque si usaramos HTTPS tampoco podemos confiarnos de nuestros usuarios**, verbigracia la misma herramienta que estamos recomendando puede modificar los datos si en la venta que se nos abre hacemos click en "modificar". Al activar este botón, cuando sale algo de nuestro ordenador nos pregunta si deseamos modificar algún dato, ¡y allí podremos echar por tierra toda la hermosa programación hecha en HTML5 y javaScript!

> Este comentario es "fuera de tema": para combatir esto último lo que debemos hacer es, en lenguaje PHP -o vuestro lenguaje utilizado- volver a validar los datos recibidos desde el usuario, ¡PERO HAY MÁS! Es obligatorio agregar "disparadores" y "restricciones" incrustados en nuestra base de datos (recomendamos PostgreSQL por su capacidad de incluso aceptar guiones en lenguaje Python) para volver a validar datos al agregar los datos a nuestro motor informático. Más detalles escapan de esta entrada, pero este resumen es el mejor que podemos expresar al respecto.

Ejecutando PHP curl en modo de depuración.

"Verbose" en idioma inglés describe un modo de hablar más de lo necesario. Pero en nuestro mundo del software libre nuestra búsqueda del conocimiento es insaciable, "solo sabemos que no sabemos nada" por ello le dedicamos esta sección empinada a ahondar en **PHP curl**. Aparte de aprender más y mantener nuestro cerebro haciendo ejercicios, *debido a la gran cantidad de protocolos y opciones es sumamente fácil para nosotros el cometer errores solicitando opciones inexistentes o incompatibles entre ellas, si son varios los parámetros que les pasemos.*

Para activar el modo de depuración -vamos que la traducción al castellano de "verboso" no cuela-

en **PHP curl** echamos mano de la alternativa **CURLOPT_VERBOSE** en la ya consabida función **curl_opt()** ¡pero esperen -como dice el "infomercial" de T.V.- hay más! **PHP curl**, dijimos, lo tenemos claro, emplea *stdin* y *stdout* que son los más conocidos **pero también utiliza *stderr*** *¡para algo que no es error, sino DEPURACIÓN!* Teoricamente no debería ser pero en la práctica, el mundo real esto es *de facto* que lo hacen -y haremos-: capturar el *stderr* para llevar los mensajes de depuración hacia un archivo de escritura y de adición. Este archivo si no está creado, lo hace, y lo abre para agregar dichos datos y si existe le adiciona al final, de tal manera que llevamos una [especie de bitácora] para nuestro estudio y control.

```
                                $ObjCurl        =        curl_init()
    or exit("No se pudo inicializar PHP curl.  ");
```

```
 curl_setopt ($ObjCurl, CURLOPT_URL, "http://www.soporteweb.com")
or exit("No se pudo establecer la URL.
");

 curl_setopt($ObjCurl, CURLOPT_VERBOSE, 1)
or exit("No se pudo establecer el modo de depuración.
");

$verbose_arch = fopen('verbose.txt', 'a');
 curl_setopt($ObjCurl, CURLOPT_STDERR, $verbose_arch)
or exit("No se pudo establecer CURLOPT_STDERR
.");

 curl_exec ($ObjCurl)
or exit("No se pudo ejecutar PHP curl.
");

 curl_close ($ObjCurl);
?>[/cc]
```

Y veremos algo parecido a esto:

Explicamos que:

- Las líneas que comienza con un asterisco son mensajes informativos de PHP curl.
- Las líneas que comienzan con "**>**" es información que se le envia al servidor web, ftp, etc.
- Las líneas que comienzan con "