Seguridad Informática: EuropioCode, un sistema de codificación de caracteres basado en el modelo SAC de 3 capas

¿Cuántas veces escuchamos y repetimos la frase "no existen los sistemas invulnerables"? Lo que nunca nos detenemos a pensar es que si no existen, es solo porque no nos detenemos a crearlos. Entonces ¿por qué no comenzar a desarrollar sistemas invulnerables? Aquí los primeros pasos para asegurar el 60% de un sistema informático.

Hace poco más de un mes, liberé la versión beta 1 de **EuropioCode**, un simple y sencillo **sistema de codificación de caracteres basado en secuencias alfanuméricas**, el cual presenté en el sitio Web de **Debian Hackers**[1].

Codificar caracteres con secuencias alfanuméricas, es la forma más confiable de permitir al usuario, el ingreso de caracteres no alfanuméricos, sin poner en riesgo la seguridad de la aplicación ni limitar las funcionalidades y prestaciones de la misma.

EuropioCode posee una doble capa de seguridad:

1) A nivel cliente (librería JavaScript);
2) A nivel aplicación (librería PHP);

Estas dos capas de seguridad, son el complemento ideal para **ModSecurity**, conformando así un **modelo de seguridad SAC** *(Server-Application-Client)*, **único capaz de alcanzar el mayor nivel de seguridad**, permitiendo sistemas informáticos invulnerables.

ModSecurity, el módulo de seguridad para Apache, debe estar instalado con las reglas de núcleo /core rules/ creadas por OWASP.

1 http://www.debianhackers.net/prevenir-ataques-xss-e-inyecciones-de-codigo-y-sql-con-europiocode

Modelo de Seguridad SAC

El modelo de seguridad SAC está basado en una triple capa de seguridad de 3 niveles: 1) **Servidor**; 2) **Aplicación**; y 3) **Cliente**.

Como modelo de seguridad, no solo es uno de los más antiguos sino que además es el único que tradicionalmente ha sido capaz de otorgar **garantías de invulnerabilidad sobre los sistemas informáticos** -siempre que el modelo sea implementado de la manera correcta-.

Un error frecuente en seguridad

A pesar de todo lo anterior, **el modelo de seguridad SAC**, generalmente solo es implementado por grandes compañías pero **subestimado por la mayoría de empresas y personas independientes**.

El **desarrollador Web** que da sus primeros pasos en la programación, suele hacerlo desde una óptica de usuario, **centrándose solo y únicamente en la seguridad del lado del cliente**, puesto que es la única que el usuario final (su cliente y en definitiva, él mismo) puede percibir sin necesidad de realizar los esfuerzos que el razonamiento deductivo (al cual no está acostumbrado) le demanden. Lamentablemente, este tipo de incipientes profesionales **desarrollan aproximadamente entre el 70% y el 80% de los sitios Web y *blogs* del mercado**.

> *Más del 70% de los sitios Web (excluyendo aplicaciones) solo implementan seguridad del lado del cliente.*

Luego nos enfrentamos al promedio de los **programadores de nivel intermedio y avanzado**, que confiados de «haberlo visto todo», **solo se concentran en la seguridad de la aplicación**, subestimando el control del lado del cliente (puesto que saben positivamente que el mismo es perfectamente vulnerable pero olvidan que no por ello es inútil) y desconociendo los mecanismos de abarcar la seguridad del lado del servidor, por lo que la misma, la delegarán en terceros que generalmente también la desconocen o no ponen en ella el esfuerzo mínimo necesario. Nuevamente debemos lamentarnos, ya que el promedio de los programadores **desarrolla aproximadamente el 80% de las aplicaciones del mercado**.

> *El 80% de las aplicaciones Web no cuenta con la seguridad necesaria del lado del cliente ni del servidor.*

Finalmente nos encontramos con los **administradores de sistemas**, quienes por lógica, **centrarán su atención en la seguridad a nivel servidor**, delegando la tarea de programación en aplicaciones prefabricadas que por lo general, han sido desarrolladas por programadores promedio de los que hablábamos en el párrafo anterior. Aquí nos encontramos con que prácticamente el 100% de los administradores de sistemas, recurre a aplicaciones prefabricadas (CMS) pero la buena noticia es que por lo general, no lo hacen como servicio a terceras personas o empresas y si lo hacen, **no superan el 5% de los blogs del mercado** (entiéndase que me refiero a los blogs que los administradores de sistemas montan para las empresas para las cuales trabajan y que no me estoy refiriendo a los blogs que montan para sí mismos).

Con suerte, podremos encontrar **un 20% de aplicaciones** que **implementan el modelo de seguridad SAC** y sin embargo, una tercera parte de estas, no lo implementa de la forma más óptima.

ModSecurity Core Rules y el fin de los falsos positivos

Hace poco tiempo, conversando con mi colega de *Debian Hackers*[2] **David Hernández**[3], surgía en medio de la charla el tema sobre *«los falsos positivos que el módulo* ***ModSecurity***[4] *de* ***Apache*** *suele arrojar»*. Hablando informalmente, comentábamos nuestro «enojo» con los mismos pero sin embargo, **la mayoría de las veces los falsos positivos de ModSecurity son producto de errores de diseño a nivel de la aplicación**. Es decir que **la mayoría de las veces, no se trata de falsos positivos**.

Los falsos positivos de ModSecurity por intentos de ataques del tipo SQL Injection o Cross Site Scripting se acaban evitando los errores de diseño en la aplicación y esto se logra, codificando las entradas del usuario mediante algún sistema alfanumérico.

A raíz de esto, que me propuse hacer algo al respecto y así nació **EuropioCode**.

EuropioCode en pocas palabras

EuropioCode no es más que un **algoritmo para codificar cadenas de caracteres en código alfanumérico** fácilmente decodificable y reversible. Esto, permite brindar al usuario la capacidad de utilizar campos de formulario con texto enriquecido -hasta incluso, escribir código fuente puro en cualquier lenguaje- y enviar el mismo codificado alfanuméricamente, evitando así los falsos positivos de ModSecuity por supuestos intentos de ataques de tipo ***SQL Injection*** o ***Cross Site Scripting (XSS)***.

De hecho, **esta técnica de codificación alfanumérica NO se utiliza en sí para evitar falsos positivos en ModSecurity** sino que por el contrario, **es la forma correcta de transferir datos de forma segura para prevenir ataques** por diversos tipo de inyecciones de código.

Si por ejemplo, se quiere permitir al usuario el ingreso de comillas dobles, incluso aunque se las reemplazase por su entidad HTML correspondiente (`&quot;`), sería un error de diseño permitir su envío en estado puro (ya sea en su carácter correspondiente como en su entidad HTML) puesto que **técnicamente se estaría permitiendo la inyección de código en la aplicación**, pues la forma correcta de permitir el ingreso de caracteres especiales, jamás puede contemplar la inyección de código.

Sin embargo, si dejásemos que el usuario ingresara esas comillas dobles pero nuestro sistema las transformase de forma inmediata en código alfanumérico (que luego tuviese la capacidad de interpretar y mostrar como al usuario como comillas dobles), estaríamos solucionando un requerimiento sin poner en riesgo la seguridad de nuestro sistema.

ECODQUOT podría ser el equivalente a las comillas dobles unicodes " y al " de HTML para nuestro sistema.

2 www.debianhackers.net

3 www.daboblog.com

4 www.modsecurity.org

Y eso, es lo que hace **EuropioCode**: transformar los caracteres no alfanuméricos en secuencias alfanuméricas que luego sea capaz de interpretar.

Codificación Europio

Antes de utilizar EuropioCode es muy importante saber que aún se encuentra en fase beta y solo contempla un número limitado de caracteres *unicode*, como se muestra en la tabla de la siguiente tabla.

Tabla de codificación de caracteres unicode

CARACTER UNICODE	EUROPIOCODE			CARACTER UNICODE	EUROPIOCODE		
	PREFIJO	CÓDIGO	SUFIJO		PREFIJO	CÓDIGO	SUFIJO
!	ECODG	33	ECODC	"	ECODG	34	ECODC
#		35		$		36	
%		37		&		38	
'		39		(		40	
)		41		*		42	
+		43		,		44	
-		45		.		46	
/		47		:		58	
<		60		=		61	
>		62		?		63	
@		64		[		91	
\		92		]		93	
^		94		_		95	
`		96		{		123	
|		124		}		125	
~		126		€		128	
(espacio)		160		¡		161	
£		163		«		171	
´		180		·		183	
»		187		¿		191	
Ç		199		ç		231	
Á		193		É		201	
Í		205		Ó		211	
Ú		218		Ü		220	
Ñ		209		á		225	
é		233		í		237	
ó		243		ú		250	
ü		252		ñ		241	
(tabulado)		09		;		59	
(salto de línea)		ECODS					

Tabla de codificación alfanumérica de caracteres

Como se puede apreciar, la codificación EuropioCode se basa en un **prefijo** (ECODG) y un **sufijo** (ECODC) predeterminados y entre medio, el **número que conforma el código hexadecimal** del carácter correspondiente. La única excepción es el salto de línea el cual es representado por el código alfabético ECODS.

La combinación **PREFIJO + NÚMERO DEL CÓDIGO HEXADECIMAL + SUFIJO** es la que permite diferenciar al intérprete, los caracteres alfanuméricos intencionales del código Europio.

Tabla de codificación de etiquetas para texto con formato enriquecido

La codificación Europio está preparada para admitir el formato de texto enriquecido, facilitando la codificación -y consiguiente decodificación- de etiquetas HTML. Para ello, todo texto introducido que sea *preformateado* mediante etiquetas HTML, siempre que se permita, será convertido sobre la base del siguiente esquema:

ETIQUETA HTML	EUROPIOCODE		ETIQUETA HTML	EUROPIOCODE	
	PREFIJO	CÓDIGO		PREFIJO	CÓDIGO
	pFt	00		pFt	12
		01			13
<i>		02	<h1>		14
		03	<h2>		15
<u>		04	<h3>		16
<strike>		05	<h4>		17
<sub>		06	<h5>		18
<sup>		07	<h6>		19
<p>		08	<code>		20
<blockquote>		09	<pre>		21
<hr>		10	 		22
		11	<small>		23

Tabla de código Europio para texto enriquecido

De la tabla anterior, se debe tener en cuenta que:

- Ninguna de las etiquetas anteriores debe aceptar atributos;
- Si una etiqueta HTML poseyera atributos, no sería interpretada como etiqueta de texto enriquecido, sino como caracteres no alfanuméricos a ser codificados;
- Aquellas etiquetas que requieren una etiqueta extra de cierre, se codificarán con los mismos códigos de la tabla anterior, pero antecediendo la vocal **e** al sufijo (`pFt00` será la codificación de la etiqueta mientras que `pFte00` lo será para)

Asimismo, se facilita una **codificación especial para etiquetas que deban indefectiblemente aceptar atributos**, como es el caso particular de la etiqueta **<a>** (ver tabla en la página siguiente).

Por favor, tener en cuenta que en caso de pretender codificar enlaces en formato HTML, solo será admitida la siguiente sintaxis:

```
<a href="url"[ target="_blank|_self|top"]>anchor text</a>
```

Ejemplos de **enlaces válidos** serían los siguientes:

```
<a href="pagina.html">Visitar enlace</a>
<a href="http://www.duckduckgo.com">Buscar en DuckDuckGo</a>
<a href="pagina.html" target="_self">Visitar enlace</a>
<a href="http://www.duckduckgo.com" target="_blank">Buscar en DuckDuckGo</a>
```

Sin embargo, **ninguno de los siguientes enlaces serán válidos como tales**:

```
<a target="_self" href="pagina.html">Visitar enlace</a>
<a href="http://www.duckduckgo.com" title="Buscar en DuckDuckGo">Buscar en DuckDuckGo</a>
<a href="pagina.html" class="mi_estilo">Visitar enlace</a>
```

Tabla de codificación de caracteres en hipervínculos

Dentro de los hipervínculos, etiquetas y atributos son codificados sobre la base de los códigos de la siguiente tabla:

CARACTER / GRUPO DE CARACTERES	EUROPIOCODE	CARACTER / GRUPO DE CARACTERES	EUROPIOCODE	CARACTER / GRUPO DE CARACTERES	EUROPIOCODE
<a href=	aH0n2	**target=_**	tG0n7	**-**	gN6n1
.	p01nt	**~**	nN0n5	**://**	pT7n3
/	bB0n1	**"**	*null*	**>**	fT0x1
****	eT0n1				

De esta forma, el enlace:

```
<a href="http://www.duckduckgo.com" target="_blank">Buscar en DuckDuckGo</a>
```

Enlace con espacios en blanco sin codificar (77 Bytes)

sería codificado como:

```
aH0n2httppT7n3wwwp01ntduckduckgop01ntcomECODG160ECODCtG0n7blankfT0x1BuscarECODG160ECODCenECODG160ECODCDuckDuckGoeT0n1
```

Enlace con espacios en blanco codificado (118 Bytes)

Un ejemplo similar, pero con enlaces sin espacios en blanco, se ve más compensado en la codificación:

```
<a href="http://www.duckduckgo.com">DuckDuckGo</a>
```

Enlace sin espacios en blanco antes de ser codificado (51 Bytes)

```
aH0n2httpT7n3wwwp01ntduckduckgop01ntcomfT0x1DuckDuckGoeT0n1
```

Enlace sin espacios en blanco ya codificado (61 Bytes - un 20% más)

Como observación, una forma de optimizar la codificación (que me apunto como un TODO), sería que el código **tG0n7**, incluyera al espacio en blanco previo al atributo *target* en la tabla de equivalencias para **target=_**

Uso de EuropioCode

Para que codificar cadenas de texto mediante EuropioCode no se convierta en una tarea tediosa, se puede recurrir a las librerías JavaScript (cliente) y PHP (aplicación) que codifican mediante EuropioCode teniendo resueltas de antemano, todas las funcionalidades necesarias.

Se puede obtener la última versión del paquete completo de librerías, haciendo un `branch` del repositorio oficial en Launchpad[5]:

```
bzr branch lp:~eugeniabahit/europiocode/trunk
```

Por favor, notar que **Launchpad** utiliza **Bazaar**[6], el sistema de control de versiones desarrollado por **Canonical** que forma parte oficial del **proyecto GNU**[7].

EuropioCode JS

europio_code.js https://bazaar.launchpad.net/~eugeniabahit/europiocode/trunk/view/head:/europio_code.js
La librería cliente, es un JavaScript capaz de codificar grandes cadenas de texto plano y texto enriquecido.

Su **uso**, está especialmente indicado para la codificación de grandes áreas de texto (`textarea`) en formularios HTML.

La forma de **implementación** sugerida, es mediante la codificación de dichos campos al ser invocado el envío del formulario, es decir, en el evento `onsubmit` tal como se muestra en el primer ejemplo de la página siguiente.

5 https://launchpad.net/
6 https://www.gnu.org/software/bazaar/
7 https://www.gnu.org/

```
window.onload = function() {
    europiocode = new EuropioCode();
    document.getElementById('id_del_formulario').onsubmit = function() {
        europiocode.encode('id_del_campo');
    };
};
```

La librería JavaScript posee dos **métodos de codificación**:

```
EuropioCode.encode()
Para codificar texto plano

EuropioCode.encode_preformat()
Para codificar texto enriquecido respetando etiquetas de formato
```

Cualquiera de las dos funciones puede ser llamada en el evento `onsubmit` y tras codificar el campo (o los campos) indicados, pasarán a un estado de solo lectura antes de ser enviados.

Vale aclarar que pueden codificarse tantos campos como sea necesario. No obstante se aconseja su uso con precaución, ya que **la codificación del texto no comprime el tamaño de las cadenas**, sino que actúan de forma contraria, insumiendo un mayor número de bytes.

El siguiente, podría ser ejemplo de implementación de EuropioCode JS para codificar el formulario utilizado por un CMS el cual es usado para agregar una nueva entrada a un *blog*. El mismo, supone tres campos de texto de tipo `textarea` con las ID: `etiquetas`, `keywords` y `entrada`:

```
window.onload = function() {

    europiocode = new EuropioCode();

    document.getElementById('nueva_entrada').onsubmit = function() {
        europiocode.encode('etiquetas');
        europiocode.encode('keywords');
        europiocode.encode_preformat('entrada');  // permite tags HTML
    };

};
```

EuropioCode PHP

europio_code.php

https://bazaar.launchpad.net/~eugeniabahit/europiocode/trunk/view/head:/europio_code.php

Por el momento, solo se dispone de una librería oficial a nivel aplicación, desarrollada en PHP. Por ello, me gustaría invitar a programadores y programadoras de diversos lenguajes como Python, Perl, Ruby, Java (entre otros), a motivarse y desarrollar una librería en el lenguaje de preferencia.

EuropioCode PHP es capaz no solo de decodificar sino también de decodificar, revertir, limpiar y purgar una cadena de texto. Pero veremos esto en detalle para poder comprenderlo mejor.

EuropioCode::encode()

Se aconseja su uso como refuerzo de seguridad en el envío de formularios. Asimismo, se aconseja la persistencia de cadenas codificadas mediante archivos de texto. Para persistencia en base de datos, ver más adelante el método clean.

```
Codifica una cadena de texto de la misma forma que lo hace la librería cliente.
Se puede utilizar como refuerzo de la seguridad a nivel aplicación, ya que todo lo que se
implemente del lado del cliente, puede ser eludido con relativa facilidad.
Su uso no afecta a una cadena que ya se encuentre codificada.

Uso:
    EuropioCode::encode(string $cadena)

Retorna:
    La cadena codificada

Ejemplo de implementación:
    $input = isset($_POST['input']) ? EuropioCode::encode($_POST['input']) : '';
```

Resultados obtenidos:

```
# Entrada del usuario: <b>Hola Mundo!</b>
$input = isset($_POST['input']) ? EuropioCode::encode($_POST['input']) : '';
print $input;
/*
Salida:
ECODG60ECODCbECODG62ECODCHolaECODG160ECODCMundoECODG33ECODCECODG60ECODCECODG47ECODCbECODG62E
CODC
*/
```

EuropioCode::decode()

Su uso está indicado solo para la lectura y decodificación de texto previamente codificado, preferentemente almacenado en un archivo de texto.

```
Decodifica una cadena previamente codificada. La decodificación NO se realiza al estado
natural de la cadena antes de codificarse, sino que todos los caracteres no alfanuméricos
previamente codificados, son convertidos a sus entidades HTML hexadecimales
correspondientes.
```

Uso:
    EuropioCode::decode(*string* $cadena)

Retorna:
    La cadena decodificada a entidades HTML

Ejemplo de implementación:
```
$encode_content = file_get_contents('entrada-1573');
$decode_content = EuropioCode::decode($encode_content);
```

Resultados obtenidos:

```
$encode_content = file_get_contents('entrada-1573');
print $encode_content;
/*
Salida:
ECODG60ECODCbECODG62ECODCHolaECODG160ECODCMundoECODG33ECODCECODG60ECODCECODG47ECODCbECODG62E
CODC
*/

$decode_content = EuropioCode::decode($encode_content);
print $decode_content;
/*
Salida:
&#60;b&#62;Hola Mundo&#33;&#60;&#47;b&#62;
*/
```

EuropioCode::decode_preformat()

Se aconseja su uso para lectura y decodificación de archivos de texto enriquecido previamente codificados con encode_preformat

Decodifica una cadena de texto enriquecido, previamente codificada con encode_preformat. La decodificación NO se realiza completamente al estado natural de la cadena. Solo, se decodificarán a estado natural, aquellas etiquetas HTML permitidas.

Uso:
    EuropioCode::decode_preformat(*string* $cadena)

Retorna:
    La cadena enriquecida decodificada

Ejemplo de implementación:
```
$encode_content = file_get_contents('entrada-1574');
$decode_content = EuropioCode::decode_preformat($encode_content);
```

Resultados obtenidos:

```
$encode_content = file_get_contents('entrada-1574');
print $encode_content;
/*
Salida:
pFt00EtiquetaECODG160ECODCadmitidapFte00ECODSECODG60ECODCdivECODG62ECODCEtiquetaECODG160ECOD
CnoECODG160ECODCadmitidaECODG60ECODCECODG47ECODCdivECODG62ECODC

En estado natural, se hubiese visto como:
<b>Etiqueta admitida</b>
<div>Etiqueta no admitida</div>
*/

$decode_content = EuropioCode::decode_preformat($encode_content);
print $decode_content;
/*
Salida:
<b>Etiqueta admitida</b>
&#60;div&#62;Etiqueta no admitida&#60;&#47;div&#62;
*/
```

EuropioCode::clean()

Se aconseja su uso para almacenamiento persistente en bases de datos.

Elimina caracteres no alfanuméricos de una cadena decodificada, reemplazando acentos, cedillas y virgulillas de eñes de vocales, ce y eñes respectivamente.
Su uso es especialmente útil para almacenar resúmenes de archivos en una base de datos.

Uso:

```
EuropioCode::clean(string $cadena)
```

Retorna:

La cadena sin caracteres no alfanuméricos

Implementación:

```
$encode_content = file_get_contents('entrada-1575');
$decode_content = EuropioCode::decode($encode_content);
$clean_content = EuropioCode::clean($decode_content);
```

Resultados obtenidos:

```
$encode_content = file_get_contents('entrada-1575');
print $encode_content;
/*
Salida:
UnECODG160ECODCECODG241ECODCandECODG250ECODCECODG160ECODCqueECODG160ECODCvivECODG237ECODCaEC
ODG160ECODCenECODG160ECODCBraziliaECODG160ECODCseECODG160ECODCmudECODG243ECODCECODG160ECODCa
ECODG160ECODCValeECODG160ECODCdoECODG160ECODCAECODG231ECODCo

En estado natural, se hubiese visto como:
Un ñandú que vivía en Brazilia se mudó a Vale do Aço
*/
```

```
$decode_content = EuropioCode::decode($encode_content);
print $decode_content;
/*
Salida:
Un &#241;and&#250; que viv&#237;a en Brazilia se mud&#243
; a Vale do A&#231;o
*/

$clean_content = EuropioCode::clean($decode_content);
print $clean_content;
/*
Salida:
Un nandu que vivia en Brazilia se mudo a Vale do Aco
*/
```

EuropioCode::purge()

Se aconseja su uso para extracción de palabras claves.

Limpia una cadena decodificada y luego elimina palabras repetidas y de longitud menor a 3 caracteres. Su uso es especialmente útil para obtener las palabras claves de un archivo o contenido determinado.

Uso:

```
EuropioCode::purge(string $cadena)
```

Retorna:

La cadena limpiada y sin palabras repetidas ni de longitud inferior a 3 caracteres

Implementación:

```
$encode_content = file_get_contents('entrada-1576');
$decode_content = EuropioCode::decode($encode_content);
$clean_content = EuropioCode::purge($decode_content);
```

Resultados obtenidos:

```
$encode_content = file_get_contents('entrada-1576');
print $encode_content;
/*
Salida:
UnECODG160ECODCECODG241ECODCandECODG250ECODCECODG160ECODCqueECODG160ECODCvivECODG237ECODCaEC
ODG160ECODCenECODG160ECODCBraziliaECODG160ECODCseECODG160ECODCmudECODG243ECODCECODG160ECODCa
ECODG160ECODCValeECODG160ECODCdoECODG160ECODCAECODG231ECODCoECODG160ECODCporqueECODG160ECODC
enECODG160ECODCBraziliaECODG160ECODCnoECODG160ECODChabitabaECODG160ECODCningECODG250ECODCnEC
ODG160ECODCotroECODG160ECODCECODG241ECODCandECODG250ECODCECODG46ECODC

En estado natural, se hubiese visto como:
Un ñandú que vivía en Brazilia se mudó a Vale do Aço porque en Brazilia no habitaba ningún
otro ñandú.
*/

$decode_content = EuropioCode::decode($encode_content);
print $decode_content;
/*
Salida:
```

```
Un &#241;and&#250; que viv&#237;a en Brazilia se mud&#243
; a Vale do A&#231;o porque en Brazilia no habi
taba ning&#250;n otro &#241;and&#250;&#46;
*/

$clean_content = EuropioCode::purge($decode_content);
print $clean_content;
/*
Salida:
nandu que vivia brazilia mudo vale aco porque habitaba ningun otro
*/
```

EuropioCode::reverse()

Se aconseja su uso SOLO Y ÚNICAMENTE, previa modificación del tipo MIME como texto plano en los encabezados HTTP. Especialmente útil, para ofrecer descargas de código fuente en estado puro. POR FAVOR, NO UTILIZAR ESTE MÉTODO EN PRODUCCIÓN, SI SE CARECE DE CONOCIMIENTOS SOBRE SEGURIDAD INFORMÁTICA A NIVEL SERVIDORES.

Vuelve una cadena codificada a su estado puro tal como se encontraba antes de la codificación.

Uso:

```
EuropioCode::reverse(string $cadena)
```

Retorna:

La cadena en estado original puro

Implementación:

```
$encode_content = file_get_contents('sourcecode-2718');
$decode_content = EuropioCode::decode($encode_content);
header("Content-Type: text/text; charset=utf-8");
print EuropioCode::reverse($decode_content);
```

Resultados obtenidos:

```
$encode_content = file_get_contents('sourcecode-2718');
/*
Contenido del archivo: sourcecode-2718
ECODG60ECODCECODG63ECODCphpECODSECODSfunctionECODG160ECODCfooECODG40ECODCECODG36ECODCparamEC
ODG61ECODCNULLECODG41ECODCECODG160ECODCECODG123ECODCECODSECODG160ECODCECODG160ECODCECODG160E
CODCECODG160ECODCifECODG40ECODCECODG33ECODCisECODG95ECODCnullECODG40ECODCECODG36ECODCparamEC
ODG41ECODCECODG41ECODCECODG160ECODCreturnECODG160ECODCsplitECODG40ECODCECODG36ECODCparamECOD
G44ECODCECODG160ECODCchrECODG40ECODC10ECODG41ECODCECODG41ECODCECODG59ECODCECODSECODG125ECODC
ECODSECODSECODG63ECODCECODG62ECODC
*/

$decode_content = EuropioCode::decode($encode_content);
/*
&#60;&#63;php
```

```
function foo&#40;&#36;param&#61;NULL&#41; &#123;
    if&#40;&#33;is&#95;null&#40;&#36;param&#41;&#41; return sp
lit&#40;&#36;param&#44; chr&#40;10&#41;&#41;&#59;
&#125;

&#63;&#62;
*/

header("Content-Type: text/text; charset=utf-8");
print EuropioCode::reverse($decode_content);
/*
Salida:

<?php

function foo($param=NULL) {
    if(!is_null($param)) return split($param, chr(10));
}

?>

*/
```