the original Hacker

creado por EUGENIA BAHIT

Jugando con la inteligencia

número 7

Woman Eyes creado por Mourad Mokrane - Silueta de Mujer creado por Leonardo B. Cunha

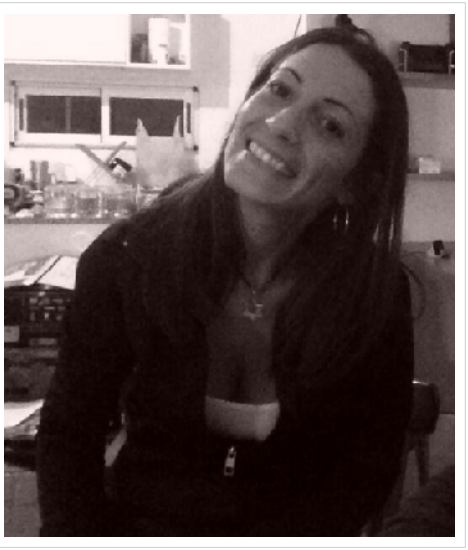
AUTHORS AGAINST DRM

SIN DRM

GNU run free run GNU

THE ORIGINAL HACKER
SOFTWARE LIBRE, HACKING y PROGRAMACIÓN, EN UN PROYECTO DE

EUGENIA BAHIT



@eugeniabahit

GLAMP HACKER Y PROGRAMADORA EXTREMA

Hacker especializada en Programación eXtrema e Ingeniería Inversa de código sobre GNU/Linux, Apache, MySQL, Python y PHP. eugeniabahit.com

Docente e instructora de tecnologías GLAMP cursos.eugeniabahit.com cursosdeprogramacionadistancia.com

Miembro de la Free Software Foundation fsf.org y The Linux Foundation linuxfoundation.org.

Creadora de python-printr, Europio Engine, JackTheStripper. Vim contributor. Fundadora de Hackers n' Developers Magazine y responsable editorial hasta octubre '13.

武士道

Buenos Aires, 30 de Junio de 2014

ÍNDICE DE LA EDICIÓN NRO7

Europio Engine Lab: URL Customizables y Slugs en Europio Engine

Llevaba tiempo Jimmy comentándome la necesidad de crear URLS personalizadas en Europio Engine ya que es uno de los requerimientos más usados en el front-end de la mayor parte de las aplicaciones Web. Unos meses atrás decidí implementar un sistema para el cual me inspiré en el de Django y para mi sorpresa, al día de hoy se convirtió en la característica más popular del engine.

Hace algunas revisiones atrás, incorporé en Europio Engine un sistema de URL personalizadas que utilizan expresiones regulares para la definición de *slugs*.

Esta idea -que como tantas otras surgió del ingenio de Jimmy Daniel Barranco- se convirtió en los últimos 2 meses en una de las características más populares del motor y por tanto, en una de las más consultadas.

Es por ello que en esta entrega explicaré como implementar dicha característica en las aplicaciones y como generar *slugs* SEO-compatibles, aprovechando la ocasión para hablarles sobre algunos *hacks* increíbles.

Comenzar a utilizar URL personalizadas

Lo primero que debemos hacer es **crear un archivo** llamado **`urls.php`** en la raíz de la aplicación. Es importante saber que este archivo no es sustituido ni reemplazado de ninguna forma cuando el *core* de Europio Engine es actualizado mediante el *updater*[1].

Trasfondo

El contenido de este archivo es ejecutado como código en estado puro por el *enrutador* de *ApplicationHandler* mediante `eval`. El *enrutador* utiliza el constructor `eval`[2] del lenguaje -destinado originalmente a evaluar código- como *hack* para evitar el uso de global y la creación de variables globales.

1 http://www.cursosdeprogramacionadistancia.com/pastebin/codigofuente/ver/16

2 http://www.php.net/manual/es/function.eval.php

El archivo `urls.php` no debe contar con etiquetas de apertura ni cierre. Solo debe contener una variable llamada `$urls` cuyo valor se un `array`:

```
$urls = array();
```

Funcionamiento del array $urls

Se trata de un *array* asociativo cuyas claves serán expresiones regulares que definan los *slugs* personalizados y sus valores asociados, la verdadera URL cuyo recurso se desee mostrar al usuario.

Un ejemplo sencillo, plantea la como *slug* la expresión regular del literal de /productos para mostrar al usuario el recurso `listar` del modelo `Producto` perteneciente al módulo de `ventas`:

```
$urls = array(
    "/^\/productos$/" => "/ventas/producto/listar"
);
```

Según el *array* anterior, los usuarios que accediesen a `examples.org/productos` llegarían al mismo recurso que accediendo a través de `example.org/ventas/producto/listar`

Ejemplos un poco más complejos, podrían requerir argumentos fijos:

```
$urls = array(
    "/^\/ofertas/" => "/ventas/producto/ver/ofertas"
);
```

O variables:

```
$urls = array(
    "/^\/producto\/[0-9|a-z|A-Z|\-]{1,}/" => "/ventas/producto/ver"
);
```

En cualquier caso, todo el *slug* será pasado como parámetro al recurso dentro de un *array*. Los elementos de dicho *array* serán cada una de las partes obtenidas tras el análisis sintáctico (*parsing*) de la URL, el cuál tomará la barra diagonal (*slash*) como parámetro divisor de la URI.

De esta forma, si el slug resultante se viese como este:

```
example.org/user/juan2013
```

Produciría el siguiente *array*:

```
array('user', 'juan2013')
```

Entonces, si hubiese estado definido como:

```
$urls = array(
    "/^\/user\/[0-9|a-z|A-Z]{1,}/" => "/usuarios/user/ver"
);
```

El *Application Handler* llamaría al recurso `ver` del modelo `user` perteneciente al módulo de `usuarios`, siendo dicha llamada equivalente a:

```
$slug_data = array('user', 'juan2013');
$controlador = new UserController();
$controlador->ver($slug_data);
```

¿Cuando se activa?

El *enrutador* del *Application Handler*, siempre intentará, en primera instancia, invocar a un módulo-modelo-recurso. Solo cuando la URI esté mal formada o sea inexistente, se fijará si existe un archivo llamado urls.php en la raíz de la aplicación y comprobará si la URI recibida coincide con alguna de las claves del array.

El *enrutador* considera que **una URI está bien formada si respeta el formato `/módulo/modelo/recurso`**`[|argumento]`. Toda URI que no respete dicho formato, será considera URI mal formada.

Slug Hack

Mostrar una vista por defecto SIN redireccionamiento

En el *array* `$urls` se debe crear la siguiente expresión regular:

```
$urls = array(
    "/^\/$/" => "/ruta/a/vista/por/defecto"
);
```

Luego, en la sección APPLICATION del archivo config.ini de la aplicación, se debe establecer el valor de la directiva DEFAULT_VIEW en /

```
DEFAULT_VIEW = /
```

Recuperar un objeto a partir de un Slug sin contar con su ID

Puede ser el caso de una URI formada solo por la denominación de un determinado objeto, como en el ejemplo del usuario /user/juan2013. En un caso como este, en el controlador puede emplearse a DataHandler[3] para recuperar al objeto sin contar con su ID, utilizando el método filter.

```
function ver($data=array()) {
    $opt = array('flags'=>FILTER_FLAG_STRIP_HIGH);
    $user = isset($data[1]) ? filter_var($data[1], FILTER_SANITIZE_SPECIAL_CHARS, $opt) : '';
    $obj_user = DattaHandler('user', DH_FORMAT_OBJECT)->filter("name=$user");
    #...
}
```


3 Ver artículo sobre «**manipulación de datos con DataHandler**» en **The Original Hacker Nº6** descargando el *paper* desde este enlace: http://library.originalhacker.org/search/datahandler

Ingeniería de Software y Estructuras de Directorios definitivas en Aplicaciones MVC Modulares

Lo padecí durante años. Durante años me dediqué a investigar y tratar de descubrir un esqueleto que resultase tan óptimo como irrefutable. La creación de Europio Engine me ayudó mucho en el hallazgo de una respuesta y ahora, tras dos años de constantes pruebas sobre algo más de 70 aplicaciones creadas, puedo contarles mi propuesta sin miedo a cometer errores.

Enredos, desprolijidades, decisiones «provisorias para siempre», son solo algunas de las tantas causas que generan el malestar y la desesperación que todo programador padece de forma permanente en su carrera como desarrollador de Software y a pesar de haber leído los mejores libros, estudiado con los mejores docentes, seguido los mejores consejos y haberse mantenido actualizado sobre los últimos avances en tecnología, la desgastante sensación nunca desaparece.

Parece una introducción irónica, casi sarcástica, pero no lo es. Es la pura realidad que vivo día a día cuando reviso algún código en GitHub, intento hacer algo de ingeniería inversa sobre alguna app o simplemente escucho padecer a mis alumnos. Pues **la organización del código fuente, la estructura interna de una aplicación y su esqueleto, representan el 80% del éxito de una aplicación**. Y si alguna de estas falla, el fracaso del Software es un hecho inminente.

La importancia contar con un sistema bien organizado

Las bases de una aplicación modular, no son solo un formalismo técnico de moda. Por consiguiente, sus principios de **portabilidad, encapsulación e independencia**, deben ser respetados sin margen de error.

Un sistema modular es aquel que se compone de un número desconocido de aplicaciones denominadas «módulos» y que permite la incorporación o eliminación de un módulo sin alterar en absoluto las aplicaciones

restantes. De esta forma, un sistema modular que hoy cuenta con un módulo de ventas y otro de costos, mañana podrá contar con un módulo extra de seguridad social sin que los módulos de ventas y costos se vean afectados.
Por otra parte, es norma inviolable, que un sistema modular cuente con un núcleo independiente, que pueda mantenerse, actualizarse y modificarse sin que ello afecte a los módulos del sistema.

Por supuesto que todo esto no se logra solo con una estructura interna coherente. Pero igual de cierto es, que contar con un esqueleto irrefutable, nos soluciona el 80% del problema.

Propuesta estructural

Como comenté al principio, tras un largo período de pruebas e investigaciones, logré dar con una estructura que al momento, a demostrado dar los mejores resultados. Esta estructura que propongo no es nueva. Parte de ella puede verse sugerida por grandes autores como **Martin Fowler**[4] y aplicada por numerosos *frameworks* en diferentes lenguajes. Yo simplemente he tomado esta arquitectura y he creado un híbrido que toma lo mejor de ésta sumando unos toques propios muy sutiles.

Exteriormente, el **esqueleto estructural de un sistema modular MVC** (independientemente del lenguaje con el que se encuentre creado), debería verse exactamente como el siguiente (algunos nombres pueden variar):

```
.
├── apps*         Aquí se almacenarán todas las aplicaciones (módulos) del sistema
├── common
│   ├── custom
│   └── libs
├── core
├── static*       Destinado al almacenamiento de archivos estáticos
└── tests
```

Los asteriscos * indican que el nombre del directorio puede variar

Sobre los nombres variables

El directorio **apps** tiene por objetivo el almacenamiento de los módulos del sistema (más adelante veremos su estructura interna). Por lo tanto, nombres como `applications` y `modules` también son válidos. Es importante que se conserve un nombre el plural, puesto que la estructura de directorios de un sistema, es también una forma de documentación. Los nombres en inglés representan una sugerencia universal que puede ser bien entendida por cualquier programador/a independientemente de su nacionalidad o procedencia.

El directorio **`static`** debe ser una carpeta fácilmente trasladable (*portable*) cuyo único objetivo sea el almacenamiento de archivos estáticos inherentes a la GUI del sistema. Por lo tanto, admitirá nombres tales como `media`, `site_media`, `static_server`, etc. dependiendo del tratamiento que dicho directorio fuese a tener.

4 http://www.martinfowler.com

```
static
├── css
├── html
├── img
├── js
├── sb-admin-v2
└── synthaxhl
```

En el ejemplo anterior, los directorios `sb-admin-v2` y `synthaxhl` corresponden a librerías de terceros y por lo tanto, su estructura interna ha sido conservada.

Sobre la finalidad de los directorios

La finalidad de las carpetas `apps` y `static` fue explicada anteriormente. Con respecto a las restantes, nos encontramos con los directorios `common` y `core`, los principales del sistemas y por ello es muy importante entender la diferencia entre ambos.

El directorio **`core`** tiene como única finalidad el almacenamiento de archivos y librerías del núcleo de la aplicación. Son archivos del núcleo, todos aquellos que resulten **indispensables para el arranque y funcionamiento del sistema** y que a la vez **puedan ser portados a otro sistema sin requerir modificaciones**. Si un archivo del `core` requiriese un mínimo cambio para servir a otro aplicación, sería un error que podría deberse a:

- La necesidad de generar una directiva de configuración variable;
- La necesidad de mover el archivo fuera del núcleo;

El directorio **`common`** suele ser similar al directorio `core` en el sentido de que todo el sistema necesita de él para funcionar. Sin embargo, a pesar de su naturaleza esencial, todos los archivos son comunes al sistema en el que se encuentran pero su portabilidad hacia otro sistema sería imposible. Es decir, que se trata de **archivos y librerías comunes al sistema que los contiene pero no a otro**, incluso aunque la similitud entre dos o más sistemas se vea afectada por la casualidad de dicha similitud, a reutilizar archivos de un `common`.

Finalmente nos encontramos con la carpeta **`tests`**, claramente destinada a las pruebas unitarias del sistema. Es sumamente importante que aunque el equipo de desarrollo aún no haya logrado implementar técnicas de programación avanzadas como TDD o pruebas unitarias, incluya esta carpeta en los sistemas que diseñe, pues servirá en un futuro al tiempo que sirve como forma de respuesta rápida a si se cuenta o no con *tests*. Al ver esa carpeta vacía, con una sola vista rápida, el programador podrá saber que el sistema no cuenta con pruebas unitarias y que no ha sido desarrollado mediante TDD.

Directorios por dentro

La estructura interna de cada directorio, varía de acuerdo al mismo y a factores inherentes al proyecto. Sin embargo, ciertas normas y sub-carpetas deberán ser respetadas como se describe a continuación. Comenzaré

por lo más simple e iré hacia lo más complejo.

Lo más aconsejable para el directorio **`tests`** es que éste replique la misma estructura de directorios que la del SUT. Esto permitirá que al momento de empaquetar el sistema para su distribución, puedan suprimirse los *tests* sin más esfuerzo que eliminar el directorio correspondiente.

La estructura interna del directorio **`static`** debe indefectiblemente quedar a criterio de los diseñadores gráficos y *maquetadores*. Si no se contase con la experiencia de profesionales gráficos en el proyecto, puede emplearse una subestructura tradicional (que no por ser tradicional será la más óptima) que garantice un orden mínimo y al mismo tiempo, facilite su comprensión. Tradicionalmente, una estructura interna óptima, sería aquella que basase su organización en el tipo de archivos. Para ello, se podría contar con una carpeta por tipo de archivos estáticos: `html`, `css`, `js`, `img`, `audio`, `video`, `pdf`, etcétera.

Cuando se trabajase con plantillas o esqueletos gráficos prediseñados como *Frameworks* o soluciones tales como *Bootstrap*, será aconsejable colocar dentro del directorio estático la carpeta original de dicha librería, respetando su estructura interna intacta ya que esto simplificará el proceso de futuras actualizaciones.

La estructura interna del directorio **`core`**, claramente depende de forma directa, del núcleo que se utilice y de cómo éste haya sido creado. Si se tiene previsto desarrollar un núcleo propio, lo más aconsejable sería organizar los subdirectorios por responsabilidades y/o tipo de herramienta. Esto es equivalente a modularizar el núcleo. Algunos ejemplos serían: *orm*, *sesiones*, *helpers*, *interfaces*, etc.

El directorio **`common`**, sin dudas, es el que mayor tendencia a la desorganización genera y sin embargo, es el que más necesita de una rigurosa disciplina en su estructura. Como les digo a mis alumnos, *«la carpeta common debe organizarse con un régimen militar»*.

La carpeta `common` suele poder resolver sus necesidades con 2 subdirectorios: `custom` y `libs` y jamás debería tener archivos en la raíz, salvo contadas excepciones bien justificadas. A lo sumo, la estructura interna más compleja que pueda verse en la raíz, podría verse como la siguiente:

```
common/
├── core
├── custom
├── libs
├── tools
└── plugins
```

En esta estructura, cada subdirectorio cumpliría una función específica:

- **`common/core:`** respetando la estructura interna del *core* del sistema, su responsabilidad sería la de almacenar cualquier extensión del núcleo. Ejemplos de ellos podrían ser clases heredadas del objeto de algún ORM o -más común- herencia de helpers.

- **common/custom:** esta es la carpeta más importante y paradójicamente la menos utilizada. La tendencia universal del programador, es descargar librerías o herramientas de terceros y confundir la libertad que el Software Libre nos da con una práctica de programación. Que un Software sea libre y permita modificar el código no implica que exista una buena práctica de programación que diga «modifica los archivos originales y arruina la posibilidad de mantenerlos actualizados». Los archivos de terceros JAMÁS deben modificarse. Hacerlo obliga a mantener la misma versión del software, por el resto de los días sin posibilidad de instalar actualizaciones, con todos los riesgos de seguridad que ello implica. El software de tercero siempre queda intacto y es en esta carpeta *«custom»* que haremos los cambios necesarios. ¿Cómo? Sobreescribiendo lo menos posible aquellas clases o funciones que se deseen personalizar.

  Una buena técnica y muy práctica consiste en crear una `CustomSomeClass` que herede de una `SomeClass` existente y sobrescriba el método que se quiere modificar, invocando previamente (siempre que sea necesario) al método `parent` correspondiente. Cuando se tratase de una función, simplemente se crearía una `CustonSomeFunction` para una `SomeFunction`.

- **common/libs:** son todas las librerías de terceros.

- **common/tools:** son todos aquellos guiones (*scripts*) propios creados con finalidades puntuales, como pueden ser generadores de *backups*, *cleaners*, funciones para envío de correo electrónico, *scripts* ejecutados por el `cron`, etc.

- **common/plugins:** los complementos suelen ser un tema muy discutido y el hacerlo tiene argumentaciones válidas desde ambas partes. Un complemento es una aplicación completa (como un módulo del sistema) pero que en vez de poder ser utilizada por un usuario es utilizada por el programador en el desarrollo de los módulos del sistema. De esta forma, un generador de formularios accesible por el navegador, que permita al usuario crear un formulario de contacto para el *frontend* de su sistema, sería un módulo mientras que una clase generadora de formularios que el programador utilizaría en la lógica de una vista para crear el formulario que el usuario utilice para generar los suyos, sería un plugin.

  Por lo tanto, un *plugin* podría considerarse complemento del *core* (es aquí donde se discute) y la línea que marca la diferencia es muy delgada y está dada por la respuesta a la pregunta *«¿viviría una aplicación sin este complemento?»*. Un ejemplo claro está en *Europio Engine*. Al ser *Europio* un núcleo propiamente dicho, el generador de formularios se considera un *plugin* ya que se podría crear cualquier tipo de aplicación y/o sistema prescindiendo de éste. Sin embargo, si se creara un *framework* utilizando *Europio Engine*, el mismo generador de formularios que hoy se considera un *plugin*, debería formar parte de las herramientas del núcleo del *framework*.

Estructura interna de un módulo

Esta estructura fue la que más me costó hallar y más años me llevó tanto investigar como probar y evaluar resultados.

Cuando se trata de una arquitectura MVC, claramente cada módulo tendrá sus directorios `models`, `views` y `controllers`, pero ahí, no puede ni debe concluir el esqueleto.

Un módulo es un Software completo. Llamémosle micro-aplicación si se lo desea, pero no deja de ser un software que emplea un núcleo común. Como tal, tendrá sus propios requerimientos y necesidades y por lo tanto, **no podrá prescindir de los siguientes archivos y directorios**:

```
app-name/
├── models/
├── views/
│   └── templates/
├── controllers/
├── helpers/
├── config.ini
├── config.ini.dist
├── app-name.sql
├── license
└── README
```

La estructura anterior es la única que ha demostrado hasta el momento, verdadera capacidad de portabilidad.

Entre los directorios, nos podremos encontrar con los siguientes destinos:

`models/`
almacenaje de todos los modelos de la aplicación (módulo del sistema)
`views/`
almacenaje de la parte lógica de todas las vistas del módulo
`views/templates/`
en caso de que el módulo posea templates HTML personaizados (que deban ser leídos pero no servidos) es aconsejable almacenarlos en este directorio, facilitando así la portabilidad del módulo.
`Controllers/`
almacenaje de los controladores del módulo
`helpers/`
almacenaje de clases y/o funciones ayudantes propias de este módulo

Los archivos de la raíz del módulo, es aconsejable que sean solo y únicamente los propuestos, considerando que cada uno contará con la siguiente responsabilidad:

`config.ini / config.ini.dist`
Todo aquel dato que pueda ser variable dependiendo del contexto, debe *setearse* en una variable global o preferentemente, en una constante. Utilizar un archivo de configuración por secciones, es una excelente alternativa ya que la mayoría de los lenguajes, posee funciones que permiten analizar este tipo de archivos (parsing) y utilizar su información. Un ejemplo de este archivo DEBE distribuirse (portarse) en un archivo de distribución .dist

app-name.sql

El módulo necesitará sus propias tablas en el sistema. Este archivo debe contener todas y cada una de las instrucciones SQL necesarias para que con tan solo importar el archivo a una base de datos, se ejecuten las consultas requeridas para que el módulo funcione sin rodeos. Una buena práctica es ir completando este archivo en tiempo real a medida que el desarrollo demande cambios en la DB.

README

De verdad, este es el archivo más importante. Puede ser un simple archivo de texto con instrucciones cortas. Cada vez que el desarrollo demande un cambio de configuración en la plataforma, la instalación de un software, librería o herramienta de terceros, dicha información se agregará en tiempo real en este archivo, bajo el título «requerimientos e instalación»

license

Un simple archivo de texto con el nombre de la licencia bajo la cual se distribuye el módulo y preferentemente un enlace hacia el texto de la licencia, será la mejor forma de portar y distribuir micro aplicaciones.

Hack para emular la autocarga y desactivación de módulos de un sistema como política de emergencia frente a fallos inesperados

Activar y desactivar módulos de forma automática, es especialmente útil cuando en producción se hace necesario actuar de manera rápida para resolver conflictos en un módulo. De esta forma, desactivando un módulo temporalmente, nos dará a tiempo a resolver el problema en producción, sin poner en riesgo a todo el sistema.

Es 30 de junio, tengo la edición finalizada pero de pronto la necesidad de generar contenido útil se impone sobre la de cumplir con lo pactado. Entonces me dije ¿por qué no hacer una edición más «temática». Y así surgió esta imperiosa necesidad de reeditar la revista a penas horas antes de su lanzamiento, con el fin de ofrecerles unas notas que en su conjunto, les puedan servir para aplicar a todo un proyecto.

En el artículo de página 3, les comentaba sobre la estructura de directorios definitiva que tras años de pruebas e investigación, logré hallar para armar el esqueleto irrefutable de una aplicación modular.

En el proceso de investigación anterior, me esforcé por obtener resultados objetivos independientes del lenguaje. Incluso, me ocupé de informarme sobre las sugerencias que expertos en cada lenguaje proponían y de allí que pude tomar lo mejor de cada una. Y en ese camino, hallé la solución a otro problema que les trasmito a continuación.

Una gran parte de los lenguajes algebraicos interpretados, cuentan con constructores que facilitan desde la autocarga de clases (el caso de PHP con la próximamente obsoleta `__autoload`[5] y

5 http://www.php.net/manual/es/function.autoload.php

`spl_autoload_register`[6]) hasta el tratamiento de directorios como paquetes (el caso de Python con el archivo `__init__.py`).

Cada lenguaje tiene su «buenas ideas» y sus aún mejores métodos y de todos ellos es posible dejarse inspirar para obtener grandes soluciones propias que permitan mejorar la calidad de nuestros proyectos y por sobre todo, del diseño de nuestras aplicaciones.
Este *hack* que les quiero contar, permite **activar y desactivar de forma automática, determinados módulos de nuestro sistema, con solo alterar un valor de forma binaria.** Esto puede ser muy útil cuando trabajamos con soluciones modulares propias que al distribuirse pueden diferentes requerimientos modulares.

> *La solución consiste en definir una directiva con valor binario, en el archivo de configuración del módulo la cual sea verificada al arrancar el sistema actuando en consecuencia.*

De esta forma, cada módulo del sistema podría contar con una variable `AUTOLOAD` dentro de una sección `[APPNAME]` del archivo de configuración del módulo, cuyos valores posibles puedan ser `1` o `0`, `True` o `False`, `On` u `Off` o cualquier otro valor binario.

```
; Archivo: appname/config.ini

[APPNAME]
; ...
AUTOLOAD = 1  ; 1 para activar, 0 para desactivar la autocarga
; ...
```

Al arrancar el sistema, un *autoload* propio recorrerá el directorio de aplicaciones (módulos del sistema) y preguntará si el valor de AUTOLOAD para cada módulo es el equivalente a encendido (1). De ser así, el auto-cargador del sistema, procederá a importar/cargar un archivo centralizador del módulo que debería incluirse en la raíz del mismo.

De esta forma, desactivar un módulo del sistema consistirá en establecer a 0 el valor de una única variable. Lo anterior puede también ser complementado a nivel gráfico, recurriendo a la misma variable para saber si incluir o no en la interfaz gráfica al módulo correspondiente. De esta forma, visualmente se activarían o desactivarían botones y menús (entre otros complementos gráficos) con solo modificar el valor de una única variable.

6 http://www.php.net/manual/es/function.spl-autoload-register.php

Ese captcha no verifica a los humanos, sino que comprueba su miopía

No me digas si eres humano. Solo envía este formulario si no eres miope.

Si la inteligencia artificial no va acompañada de inteligencia real, de muy poca utilidad resultará.

Se supone que la Ingeniería es la ciencia encargada de encontrar soluciones a problemas existentes, en un área determinada. De esta forma, la Ingeniería de Software, se encarga de encontrar soluciones informáticas a problemas determinados.

Sin embargo, **la implementación de *«captchas»* ha supuesto la generación de un problema, resultando solucionar menos de lo que complica**.

Un requisito de inteligencia debería ser la simplicidad

Si una solución no es simple, no debería ser considerada inteligente. Pues la grandeza existe en la simplicidad.

Como usuario de un sistema, seguramente padeciste la frustración de no lograr enviar un formulario exitosamente porque has fallado acertando la respuesta al *captcha* o peor aún, porque tu tiempo de sesión ha expirado. Como programador/a **¿te planteaste alguna vez que si los *captcha* han sido creados para descartar robots cometen un error al expirar puesto que los humanos por naturaleza procesamos la información más lento que un ordenador?**

La idea de los *captcha* siempre me gustó y me resultó sumamente interesante, conceptualmente hablando. Pero ni su propuesta ni su implementación me han resultado útiles. De allí que decidí crear mis propios *captchas* en mis aplicaciones, que solo verificasen que quien pretende enviar un formulario sea humano y no, que no sea miope.

Los captchas, en su mayoría, utilizan imágenes generadas al azar a fin de que la respuesta no pueda ser leída por un ordenador. Sin embargo, la inteligencia artificial -al menos en niveles accesibles- no ha alcanzado a «razonar» como lo hacemos los humanos. Esto, significa que un ordenador, no sería capaz, por ejemplo, de

razonar lateralmente, de entender el sarcasmo o disfrutar de un chiste. Entonces ¿por qué no utilizar texto plano y preguntar de qué color es un albaricoque violeta?

Lo admito. No se qué es un albaricoque (ya lo buscaré en el diccionario), pero sí soy capaz de deducir que la respuesta correcta a la pregunta anterior es violeta. Y no necesité hacer esfuerzos, claro está.

¿Cómo crear un captcha en texto plano que solo pueda ser respondido por humanos?

Lo único que necesitas es generar tus propias preguntas y respuestas. Cuanto más fácil sea la pregunta, mejores resultados obtendrás. Se necesitaría una base de datos para hallar las respuestas por un ordenador o un algoritmo de inteligencia artificial altamente avanzado, pero aún no ha sido desarrollado (aunque sí investigado).

Las mejores preguntas serán aquellas impliquen el cumplimiento de ciertas consignas. Por ejemplo «dígame cuantos dedos hay en una mano pero no utilice números». Hasta un niño pequeño podría responder «cinco».

La base de datos

La lista de preguntas y respuestas puede ser almacenada en una base de datos tradicional, o mejor aún, en un archivo de texto plano que consuma menor cantidad de recursos.

En este último caso, se deberá decidir un formato identificador para el conjunto-par pregunta-respuesta. Mi preferido es colocar un par por línea y respuesta separada de pregunta por un *pipe*.

```
Pregunta 1 | Respuesta 1
Pregunta 2 | Respuesta 2
Pregunta 50 | Respuesta 50
```

de esta forma, el carácter divisor de pares será el salto de línea y el de pregunta-respuesta, el *pipe*.

La obtención de una pregunta

La pregunta debe ser algo elegido al azar ya que el azar dificulta la generación de *cracks*. Las funciones *random* del lenguaje, son útiles en estos casos. Un ejemplo en PHP podría verse como el siguiente:

```
# Almaceno el contenido de la base de datos en un array (salto las líneas en blanco)
$rows = file('database', FILE_SKIP_EMPTY_LINES);

# Genero un entero al azar para utilizar luego como índice
$random = rand(0, count($rows) - 1);

# Obtengo el par pregunta-respuesta
$par = $rows[$random];
```

```
# Separo la pregunta de la respuesta
list($pregunta, $respuesta) = explode('|', $par);
```

Capturar la respuesta sin requerir sesiones

Al usuario mostraremos la pregunta:

```
<?php echo $pregunta; ?>
```

Y por lógica, ofreceremos una forma de indicar la respuesta, mediante un campo de formulario:

```
<input type='text' name='respuesta'>
```

Para luego validar la respuesta (sin requerir almacenarla previamente en una sesión) necesitaremos una forma de capturar la respuesta y saber a qué pregunta pertenece. Para ello, podremos recurrir al número de índice almacenándolo en un campo oculto (aunque esto sería algo bastante vulnerable):

```
<input type='hidden' name='random_question' value='<?php echo $random; ?>'>
```

Para hacerlo menos vulnerable, podría crearse un *token* para este campo, el cual se guardase temporalmente en un archivo junto al *index* correspondiente:

```
$token = md5(uniqid(mt_rand(), True));
file_put_contents("/private/dir/$token", $random);
```

Luego, lo que sería almacenado en el *hidden*, sería el token:

```
<input type='hidden' name='random_question' value='<?php echo $token; ?>'>
```

de esta forma, se podría capturar la respuesta y la pregunta a través de su token.

Validar la respuesta

Si se ha generado un token, obtendremos el *index* de la respuesta a través de la lectura del archivo «tokenizado»:

```
$opt = array('flags'=>FILTER_FLAG_STRIP_HIGH);
$user_token = isset($_POST['random_question']) ? filter_var(
    $_POST['random_question'], FILTER_SANITIZE_SPECIAL_CHARS, $opt) : 'null';
$indice = null;
$file = "/private/dir/$user_token";
if(file_exists($file)) $indice = file_get_contents($file);
```

Una vez obtenido el índice, solo es cuestión de repetir los pasos con los que se obtuvo la pregunta para obtener la respuesta y compararla:

```
$rows = file('database', FILE_SKIP_EMPTY_LINES);
$par = $rows[$indice];
list($pregunta, $respuesta) = explode('|', $par);

if($respuesta == $respuesta_dada_por_el_usuario) {
    # todo correcto ...
} else {
    # respuesta equivocada ...
}
```

Alternativamente, dado que no necesitamos conocer la cantidad de datos alojados en el archivo, para localizar la respuesta podemos recurrir al método `seek` de `SplFileObject`:

```
$database = new SplFileObject($file);
$database->seek($indice);
$respuesta = $database->current();
```

Al finalizar toda la comprobación, es aconsejable liberar espacio en disco eliminando el archivo creado. No obstante, una tarea programada en el *cron* que se ejecute de forma periódica en busca de archivos antiguos (de un 1 día de antigüedad o más) para eliminarlos, sería un complemento ideal.

Lectura recomendada

Emulación de tokens de seguridad temporales para el registro de usuarios
http://library.originalhacker.org/search/token

ACERTIJO:

¿suicidio, homicidio o un caso para «Molder y Scully»?

Un equipo de investigadores encuentra en una habitación completamente vacía, carente de ventanas y con la única puerta cerrada con un pestillo desde el interior, a un hombre ahorcado con una soga amarrada de una biga ubicada a 4 mts del suelo y a 20 cm del techo. A los investigadores, llamó poderosamente la atención, haber encontrado el suelo lleno de agua, teniendo en cuenta que en toda la habitación, no había cañerías ni ningún otro objeto más que las paredes, el techo, el suelo, el occiso, la soga, la biga y la puerta con su pestillo. ¿Cómo logró colgarse el occiso?

Ayuda: los investigadores hallaron la respuesta en el agua encontrada en el suelo de la habitación.

Responde **ANTES** del **20/07/2014** a través de **Twitter** utilizando el *hashtag* **#AcertijoTOH7**
El nombre de los ganadores será publicado en la siguiente edición
La respuesta correcta será publicada en The Original Hacker Nº8 junto con los ganadores

SOLUCIÓN AL ACERTIJO DE THE ORIGINAL HACKER Nº6

Esta vez no nos ha ido tan bien con el acertijo. Por si no lo recuerdas, el mismo decía «Por 1, Juan debía pagar $5 y lo mismo que paga por 1 lo paga por 5. Por 15, el precio era de $10 pero por 100, de $15. ¿Qué estaba comprando Juan?» y la respuesta, estaba en la misma ayuda: «Ayuda: que los **números** no te engañen....». ¡Juan estaba comprando números! Los números que compraba podrían haber sido los que se colocan en las puertas de las casas para identificar un domicilio o hasta incluso velas de cumpleaños.

Esta vez no hay *ranking* de ganadores. Solo podemos mencionar la respuesta más acercada, que nuevamente, ha sido la del «futuro fiel seguidor» del **#AcertijoTOH**, **@diegodelavega** ¡Felicitaciones!

FREE AS IN FREEDOM LA VERDADERA ÉTICA HACKER

Necesitamos vivir en un mundo donde el acceso al **conocimiento** sea **LIBRE**; donde la **solidaridad** con el vecino NO sea **ILÍCITA**. Debe dejar de condenarse el compartir conocimiento con nuestros seres queridos y allegados. **El cambio está en tus manos, hay mucho que puedes hacer.**

DILE A MOZILLA QUE NO INCORPORE EL DRM DE ADOBE A FIREFOX

Sólo una semana después del Día Internacional contra el DRM, **Mozilla anunció que apoyará la Gestión Digital de Restricciones en su navegador Firefox**, el cual tendrá una utilidad incorporada que obtendrá e instalará automáticamente los DRM de Adobe.

ENVÍALE UN E-MAIL A ANDREAS GAL (CTO de Mozilla) haciéndole saber que te opones al DRM.

CANCELA TU CUENTA EN NETFLIX ¡YO LO HICE!

Hace un año atrás, enterada de que **Netflix solo permitiría la reproducción de vídeos en navegadores «*extensiones premium*» (así llamaron al DRM)** cancelé mi suscripción anual ¿por qué? Porque la libertad de toda una comunidad DEBE valer más que cualquier serie HD.

ACCEDE A TU CUENTA DE NETFLIX y **ENVÍALES UN MENSAJE CANCELANDO TU SUSCRIPCIÓN**. **Anuncia tu decisión en Twitter** y tus redes sociales.

NO USES DISPOSITIVOS QUE PROHIBEN COMPARTIR TUS PROPIOS ARCHIVOS CON LIBERTAD ¡YO NO LOS USO!

Amazon Kindle, iPad, iPod, iPhone y demás dispositivos de Apple incluyen una Gestión Digital de Restricciones que **daña la libertad de las personas**. Dispositivos con Android ¡también son peligrosos!

¡NO LOS USES!

UTILIZAR SOFTWARE PRIVATIVO O DISPOSITIVOS CON RESTRICCIONES

Y DECIR QUE APOYAS EL SOFTWARE LIBRE, ES CINISMO.

UN HACKER ÉTICO ES AQUEL QUE ES COHERENTE CON LO QUE DICE Y HACE.

DEFECTIVE BY DESIGN.org

HACKTIVISMO ÉTICO

LAS SIGUIENTES COMPAÑÍAS HACEN DAÑO A LA LIBERTAD DE LAS PERSONAS

amazon.com

Mediante sus plataformas Kindle y Prime. **Leer más »**

Apple utiliza el DRM para controlar iOS y a los usuarios de OS X. **Leer más »**

Microsoft

El DRM se incorpora en el corazón de Windows y muchos servicios como Silverlight, imponen el DRM en los usuarios. **Leer más »**

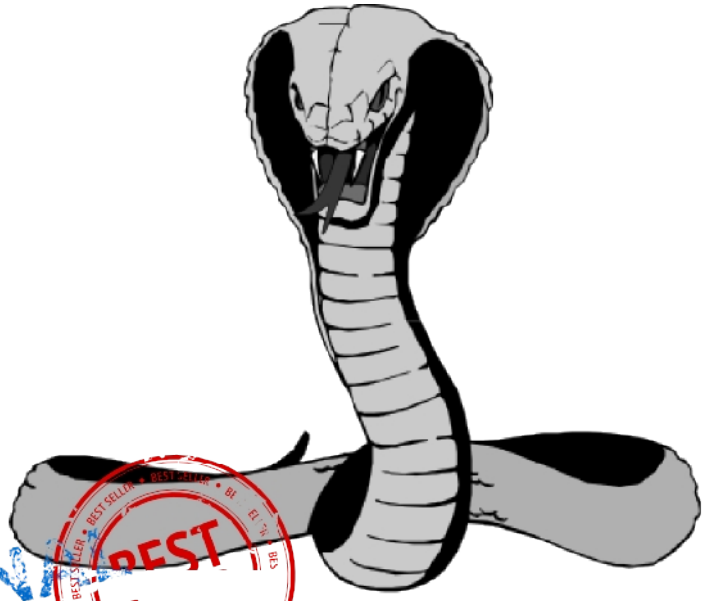
NETFLIX

No satisfecho con el uso de DRM para su servicio de *streaming*, ahora Netflix está tratando de imponer el DRM en la Web. **Leer más »**

SONY

Sony ha utilizado acciones legales para hostigar e intimidar a las personas que han modificado sus sistemas PS3. **Leer más »**

SI DE VERDAD APOYAS EL SOFTWARE LIBRE, ÚSALO, PROMUÉVELO Y ABANDONA LO PRIVATIVO SIN EXCUSAS. EL CAMBIO ESTÁ EN TUS MANOS.



The Original Hacker #_7__