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