Seguridad Informática: Emulación de Tokens de Seguridad temporales como mecanismo de verificación en el registro de usuarios

Asegurarse de que el usuario que solicita el registro es humano y de que el e-mail indicado es realmente de su propiedad, es tan importante como mantener al resguardo la base de datos de usuarios ya confirmados. La emulación temporal de Tokens de Seguridad es el mecanismo más simple y eficiente para poder lograrlo, que se ha utilizado tradicionalmente.

Hace ya unas dos o tres semanas atrás, **Guillermo Montero** -uno de mis alumnos del curso de Análisis en PHP- me preguntaba sobre cómo implementar un **sistema de *Tokens* de Seguridad temporales en bases de datos** y de allí mi promesa de escribir este *paper* y dedicárselo, pues el tema, fue una idea suya, que seguramente resulte de interés a muchos programadores.

En el mundo de la Ingeniería de Software suele ser muy frecuente hablar de "*tokens*" como mecanismos de seguridad para efectuar validaciones de diversos tipos. Es importante aclarar que **un *token* de programación, es en realidad una palabra clave o identificador del lenguaje** mientras que **los *Tokens* de Seguridad, son dispositivos físicos** que electrónica o digitalmente almacenan información, que sirve para identificar al usuario que lo porta y son utilizados como medio de autenticación. **Cuando en programación, se habla de *Tokens* de Seguridad,** se lo hace en analogía a estos dispositivos y **usualmente se lo utiliza como sinónimo de "valor *hash*".**

Un valor *hash* es una cadena de longitud fija obtenida por una función H que efectúa un resumen (o compactación) de una cadena M, codificándola de manera irreversible. Se podría concluir entonces, en que el **proceso de emulación de un Token de Seguridad en programación**, es el resultado de una aplicación criptográfica, **H(M) = *valor hash.***

Generación del Token de Seguridad o valor hash

El valor Hash puede ser obtenido mediante un conjunto M a libre elección del programador. Por ejemplo, M podría ser la sumatoria del e-mail del usuario y el tiempo POSIX (*timestamp*). Mientras tanto, la función H podría ser MD5, SHA-512 o cualquier otra función *hash* segura:

```
# M = email + timestamp
$m = $email . Time();
$valor_hash = hash('sha512', $m);
```

Flujo de Procedimientos: ¿en qué consiste el mecanismo de verificación?

1. El flujo de procedimientos comienza con el usuario completando sus **datos de registro** mediante un formulario;
2. Cuando este formulario es enviado, se genera el ***Token* de Seguridad** y se lo almacena junto a los datos de registros en una **tabla de registro temporal**;
3. A continuación, se envía un **e-mail** al usuario con el *token* generado;
4. El usuario deberá ingresar en la aplicación (ya sea por GET o POST) el token recibido por e-mail, el cuál deberá ser **comparado con el *token* almacenado en la tabla de registro temporal**. En caso de coincidir, **se inserta el registro en la tabla de usuarios confirmados** y **se elimina el mismo, de la tabla temporal**.

Tabla de registro temporal

La tabla de registro temporal **debe ser un clon exacto de la tabla de usuarios confirmados**. Es importante que la tabla de usuarios confirmados cuente con un campo "token" y un *timestamp*, ya que incluso, estando confirmado un usuario, un nuevo token podría ser de utilidad para efectuar una restauración de contraseña.

Una vez creada la tabla principal (la de usuarios confirmados), puede obtenerse una copia exacta de la misma (un clon) con la siguiente sentencia SQL:

```
CREATE TABLE tmp_users LIKE users;
```

Comparación de Tokens e intercambio de registros

La comparación de Tokens e intercambio de registro, puede hacerse en un único paso, insertando el registro en tabla final, mediante una selección del mismo en la tabla temporal:

```
INSERT INTO users SELECT * FROM tmp_users WHERE token = ?
```

Si el resultado de la ejecución anterior es verdadero, entonces simplemente, se elimina el registro de la tabla temporal:

```
DELETE FROM tmp_users WHERE token = ?
```

Optimizando recursos

La optimización no solo de recursos sino también de procesos, forma parte de una buena política de seguridad:

> *cuanto más optimizada se encuentra una aplicación, más simple se hace la detección de errores y más se reducen los riesgos de fallos y de fugas de información*

Es imprescindible que la tabla destinada al registro temporal, sea verificada de forma periódica a fin de evitar la sobre-indexación de registros que pudiesen ser producto de intentos masivos y automatizados, de registros falsos. Limpiar la tabla temporal de falsos registros y optimizar sus índices y el espacio libre, previene la saturación de recursos y su consiguiente falta de disponibilidad.

Se puede crear entonces, una tarea programada con `crontab`[3] que ejecute un archivo encargado de recorrer la tabla temporal, en busca de registros donde el *timestamp* del registro + 12 horas, por ejemplo, sea inferior al *timestamp* actual y a continuación, eliminar dichos registros obsoletos y optimizar la tabla:

```
#!/bin/bash

QUERY_ELIMINAR='DELETE FROM tmp_users WHERE TIMESTAMP(timestamp, '12:00:00') < NOW();'
QUERY_OPTIMIZAR='OPTIMIZE TABLE tmp_users;'

mysql -u usuario -pclave -e "$QUERY_ELIMINAR $QUERY_OPTIMIZAR"
```

Y finalmente:

```
# Agregar permisos de ejecución al archivo
chmod +x /ruta/a/script.sh

# Y agregar la tarea programada al cron con crontab
crontab -e
@daily /ruta/a/scipt.sh
```

3 Recomiendo leer el artículo "Actualizando tus aplicaciones con Cron" que escribí para la revista Hackers & Developers Magazine Nº6: http://www.eugeniabahit.com/static/pdf/LINUXSYSADMIN%20-%20Cron.pdf