UNIVERSIDAD SAN CARLOS DE GUATEMALA

CENTRO UNIVERSITARIO DE OCCIDENTE

DIVISIÓN CIENCIAS DE LA INGENIERÍA

CARRERA DE INGENIERÍA CIENCIAS Y SISTEMAS

LABORATORIO DE SEMINARIO DE SISTEMAS I

"**OCTAVO SEMESTRE"**

ING.: PEDRO LUIS DOMINGO VÁSQUEZ

ESTUDIANTES: LUIS ESTUARDO BOLAÑOS GONZÁLEZ - 201731766

BRYAN RENÉ GÓMEZ GÓMEZ - 201730919

TRABAJO: REPORTE DE SEGURIDAD

FECHA: 10 de octubre de 2,021

Tiempo Maya

Cadena incompleta de escape o codificación

Un transformador de cadena que no reemplaza o escapa a todas las apariciones de un metacaracter puede ser ineficaz.

La desinfección de la entrada que no es de confianza es una técnica común para prevenir ataques de inyección, como la inyección SQL o la secuencia de comandos entre sitios. Por lo general, esto se hace escapando metacaracteres, como comillas, de una manera específica del dominio para que se tratan como caracteres normales..

Sin embargo, el uso directo del método de reemplazo de cadenas para realizar el escape es notoriamente propenso a errores. Los errores comunes incluyen solo reemplazar la primera aparición de un metacaracteres, o escapar de varios metacaracteres con barra invertida, pero no la barra invertida en sí.

En el primer caso, los metadatos posteriores no se alteran y pueden usarse para subvertir la desinfección. En el último caso, anteponer un metacaracter con una barra invertida hace que la barra invertida se escape, pero el metacaracter aparece sin escape, lo que nuevamente hace que la desinfección sea ineficaz.

Incluso si la cadena de escape no se usa en un contexto crítico para la seguridad, el escape incompleto puede tener efectos no deseados, como una salida mal renderizada o confusa.

Recomendación

Utilice una biblioteca de desinfección (bien probada) si es posible. Es mucho más probable que estas bibliotecas manejen los casos de esquina correctamente que una implementación personalizada.

Una alternativa aún más segura es diseñar la aplicación para que no sea necesaria la desinfección, por ejemplo, utilizando declaraciones preparadas para consultas SQL.

De lo contrario, asegúrese de usar una expresión regular con el indicador g para asegurarse de que se reemplacen todas las apariciones y recuerde evitar las barras diagonales inversas si corresponde.

Sin embargo, tenga en cuenta que esto generalmente no es suficiente para reemplazar cadenas de varios caracteres: el método String.prototype.replace solo realiza una pasada sobre la cadena de entrada y no reemplazará más instancias de la cadena que resulten de reemplazos anteriores.

Por ejemplo, considere el fragmento de código s.replace (/\/\.\.\// g, ""), que intenta eliminar todas las apariciones de /../ de s. Esto no funcionará como se esperaba: para la cadena /./.././, por ejemplo, eliminará la única aparición de /../ en el medio, pero el resto de la cadena se convertirá en /../, que es otra instancia de la subcadena que estábamos intentando eliminar.

Ejemplo

Por ejemplo, supongamos que queremos incrustar un número de cuenta de cadena controlado por el usuario en una consulta SQL como parte de un literal de cadena. Para evitar la inyección de SQL, debemos asegurarnos de que la cadena no contenga caracteres de comillas simples sin escape. La siguiente función intenta garantizar esto duplicando las comillas simples y evitándose:

```
function escapeQuotes(s) {
  return s.replace("'", "''");
}
```

Tal como está escrito, este desinfectante es ineficaz: si el primer argumento a reemplazar es un literal de cadena (como en este caso), solo se reemplaza la primera aparición de esa cadena.

Como se mencionó anteriormente, la función escapeQuotes debe reemplazarse con una biblioteca de desinfección especialmente diseñada, como el módulo npm sqlstring. Muchas otras bibliotecas de desinfección están disponibles en npm y otras fuentes.

Si esta no es una opción, escape Quotes debe reescribirse para usar una expresión regular con la bandera g ("global") en su lugar:

```
function escapeQuotes(s) {
  return s.replace(/'/g, "'");
}
```

Tenga en cuenta que es muy importante incluir el indicador global: s.replace (/ '/, "' ") sin el indicador global es equivalente al primer ejemplo anterior y solo reemplaza la primera cita.

Expresión regular ineficiente

Una expresión regular que requiere un tiempo exponencial para coincidir con ciertas entradas puede ser un cuello de botella en el rendimiento y puede ser vulnerable a ataques de denegación de servicio.

Algunas expresiones regulares tardan mucho en hacer coincidir ciertas cadenas de entrada hasta el punto en que el tiempo que se tarda en hacer coincidir una cadena de longitud n es proporcional ank o incluso 2n.

Estas expresiones regulares pueden afectar negativamente al rendimiento, o incluso permitir que un usuario malintencionado realice un ataque de denegación de servicio ("DoS") al crear una cadena de entrada costosa para que la expresión regular coincida.

Los motores de expresión regular proporcionados por muchas plataformas populares de JavaScript utilizan autómatas finitos no deterministas de retroceso para implementar la coincidencia de expresiones regulares.

Si bien este enfoque ahorra espacio y permite admitir funciones avanzadas como grupos de captura, en general no es eficiente en el tiempo. La complejidad de tiempo en el peor de los casos de un autómata de este tipo puede ser polinomial o incluso exponencial, lo que significa que para cadenas de cierta forma, aumentar la longitud de entrada en diez caracteres puede hacer que el autómata sea aproximadamente 1000 veces más lento.

Normalmente, una expresión regular se ve afectada por este problema si contiene una repetición de la forma r * o r + donde la subexpresión r es ambigua en el sentido de que puede coincidir con alguna cadena de varias formas. Puede encontrar más información sobre las circunstancias precisas en las referencias.

Recomendación

Modifique la expresión regular para eliminar la ambigüedad o asegúrese de que las cadenas que coinciden con la expresión regular sean lo suficientemente cortas como para que la complejidad del tiempo no importe.

Ejemplo

Considere esta expresión regular:

```
/^_(__|.)+_$/
```

Su subexpresión "(__ |.) +?" puede coincidir con la cadena "__" ya sea por la primera alternativa "__" a la izquierda de "|" operador, o por dos repeticiones de la segunda alternativa "." A la derecha. Por lo tanto, una cadena que consta de un número impar de guiones bajos seguidos de algún otro carácter hará que el motor de expresiones regulares se ejecute durante una cantidad de tiempo exponencial antes de rechazar la entrada.

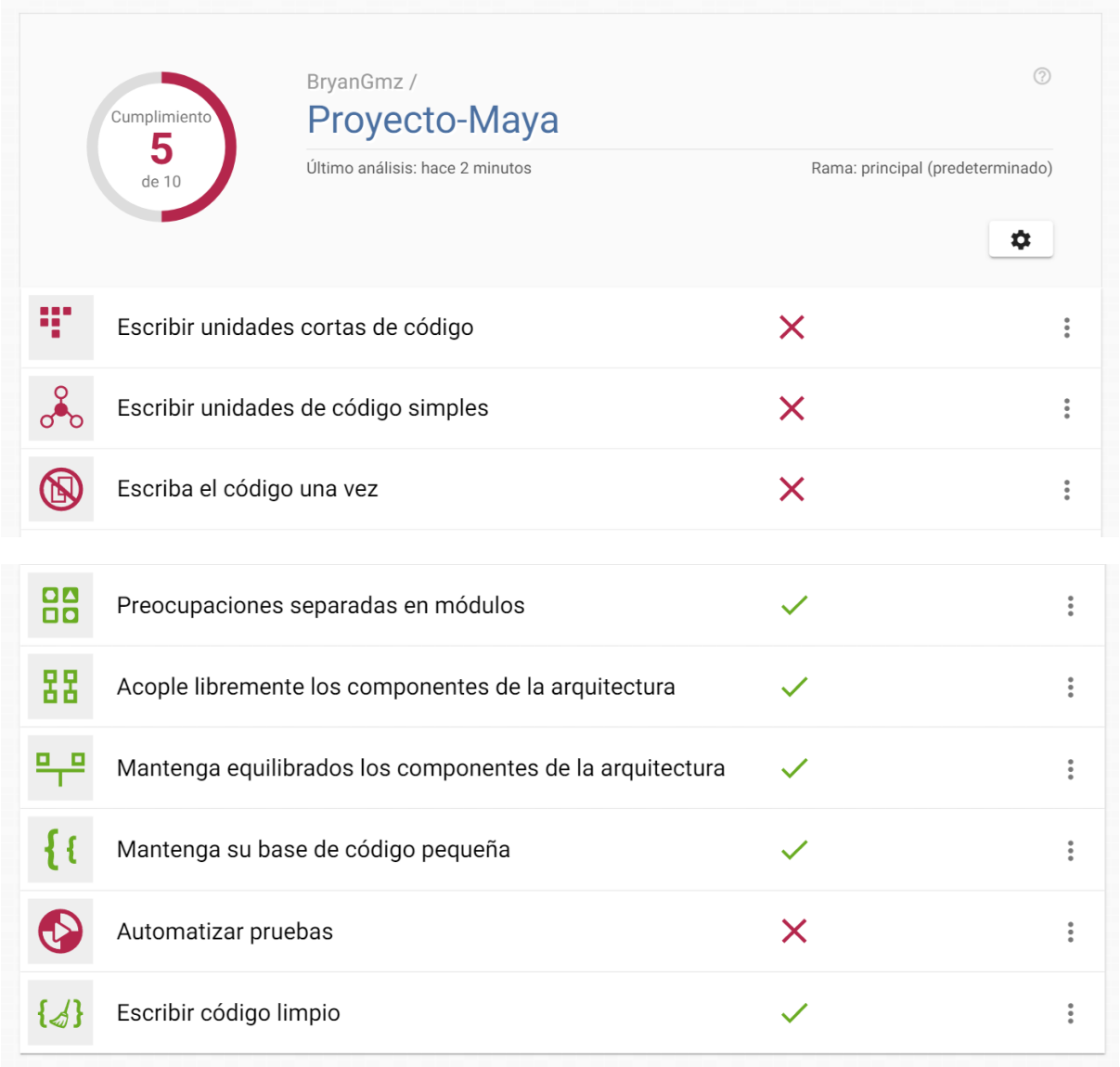
Este problema se puede evitar reescribiendo la expresión regular para eliminar la ambigüedad entre las dos ramas de la alternativa dentro de la repetición:

```
/^_(__|[^_])+_$/
```

Observaciones Generales

- La documentación interna del código es escasa e inexistente en muchos archivos, esto dificulta un poco la compresión del código.
- No existe una distinción clara entre backend y frontend debido a que en muchos archivos se escribe código que se encarga de la lógica que no forma parte de la interfaz gráfica, junto al código encargado de la interfaz gráfica de la aplicación.
- No se hace uso de funciones para el llamado y la ejecución de acciones.
- El patrón de diseño utilizado limita la escalabilidad del proyecto, se recomienda emplear un patrón MVC
- Establecer un estándar para la definición de nombres de clases, algunas clases son nombradas utilizando camelCase mientras que otras utilizan UpperCamelCase.
- La documentación interna del código es escasa y nula en la mayor parte de clases.

Resultados de la Evaluación



Según esta evaluación se puede ver de forma clara cierto tipo de aspectos importantes relacionados a la seguridad que el sistema principal no cumple en su totalidad, como puede ser el hecho de la escritura de código simple, viéndose reflejado en los archivos .php donde se hace una combinación de frontend con backend dentro de las mismas páginas, siendo esta una mala práctica dentro de la programación y que podría afectar negativamente al sistema en algún momento. Otro defecto con el cual me tope fue el hecho de la repetición de código en varios archivos, lo cual hace que la optimización no sea óptima.

Interactive Hotspots Map

67%

Red Hotspots

26%

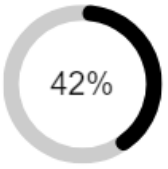
Development Effort in Red Hotspots

%

Of Estimated Bugfixes in Red Hotspots

View Hotspots

Key Personnel

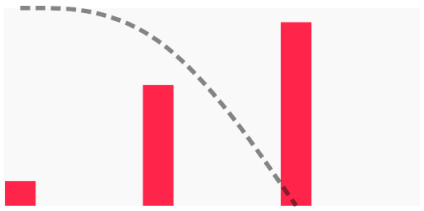


42% of the code written by 1 developers

0% of the code by former contributors

View Knowledge Metrics

Delivery Effectiveness Trends



Development activity over the last 3 months

Development Output per Author

View Delivery Effectiveness Details

Author contributions

12 total authors

Median contribution

0 months

Longest contribution

0 months

View Author contributions

Branch Delivery Risk

4 active branches

Branch duration:	1d 20h
Lead Time to Merge:	3h

None of your current branches show signs of excessive risk.

See Active Branches

Hotspot Code Health

9.88 → 0.0%

Average Code Health

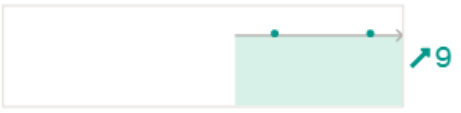
Enable full scan

Worst Performer

Enable full scan

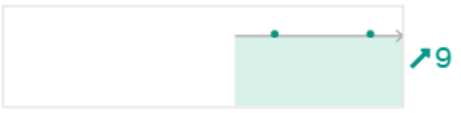
File Level Hotspots

lavBar.php
'royecto-Maya/Web/
.54 LoC | 8 commits



lavBar2.php
'royecto-Maya/Web/
.53 LoC | 7 commits

lavBarAdmin.php
'royecto-Maya/Web/
.45 LoC | 7 commits



:onexionDb.java
'royecto-Maya/Escritorio/.../database/
:1 LoC | 13 commits

:ditarInformacion.php
'royecto-Maya/Web/
.06 LoC | 8 commits

See All Hotspots