UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
SECRETARÍA DE DESARROLLO INSTITUCIONAL
Dirección General de Cómputo y de Tecnologías de Información y Comunicación
Dirección de Colaboración y Vinculación

Fecha elaboración: 31 de Enero de 2022

Luz María Ramírez Romero
ORCID: 0000-0002-8867-9374
Correo electrónico: luzrr@unam.mx

Aspectos de seguridad en el desarrollo de aplicaciones web con el framework Laravel

ABSTRACT

A part of the quality in software applications lies in taking care of security aspects. Neglecting security from the development of systems can compromise the security of the entire server where the system is hosted, just as it can expose other systems that coexist in the same computer equipment.

Developers, in addition to paying attention to achieving the functionality of the users' requirements, must put special emphasis on the computer security features that must be integrated into their source code.

Computer security does not only concern server administrators and network administrators, computer security must be afforded from all the roles. In this article we will deal with the tools and mechanisms that the Laravel framework offers to the application developer to shield their source code and protect it from some hacker attacks.

RESUMEN

Parte de la calidad de una aplicación de software está en el cuidado de los aspectos de la seguridad de la información que maneja. Descuidar la seguridad desde el desarrollo de sistemas, puede comprometer la seguridad de todo el servidor dónde se aloja el sistema, al igual que puede exponer a otros sistemas que convivan en el mismo equipo de cómputo.

Los desarrolladores además de poner su atención en lograr la funcionalidad de los requerimientos de los usuarios, deben poner especial acento en las características de seguridad informática que deben integrar en su código fuente.

La seguridad informática no sólo le atañe a los administradores de servidores y administradores de redes, la seguridad informática la cuidamos todos. En este artículo trataremos las herramientas y mecanismos que ofrece el framework Laravel al desarrollador de aplicaciones para blindar su código fuente y protegerlo de algunos ataques informáticos.

KEYWORDS

Security aspects in Laravel framework, SQL Injection, Cross Site Request Forgery (CSRF), Laravel's eloquent ORM (Object Relational Mapping) and the security

PALABRAS CLAVE

Aspectos de seguridad en el framework Laravel, inyección de código SQL, Cross Site Request Forgery (CSRF), el ORM (Mapeo de objetos relacional) Eloquent de Laravel y la seguridad

Introducción

En el desarrollo y operación de aplicaciones de software en web, preservar la seguridad informática es responsabilidad de todos los roles tecnológicos alrededor de la aplicación. Como ejemplos podemos mencionar que es responsabilidad de los usuarios cuidar que sus contraseñas sean fuertes y no las deben dejar al alcance de otras personas escritas en libretas, post-its u otros medios. Los administradores del servidor dónde opera la aplicación deben mantener las versiones del sistema operativo y de los servidores web actualizadas y deben instalar los parches de seguridad que vayan publicándose y deben configurar el cifrado de los canales de transmisión de información, cuando lo que viaja entre servidor y clientes sea sensible. Los administradores de red, deben monitorear y rechazar las peticiones al servidor que puedan comprometer su seguridad.

Cada rol alrededor del desarrollo y de la operación de los aplicativos informáticos debe cuidar los aspectos que le competen en torno a la seguridad informática, de tal suerte que se proteja la información y las aplicaciones desde las diferentes capas o niveles. Este documento centrará su atención en el rol de los desarrolladores de las aplicaciones, que juega también un papel muy activo y fundamental en la seguridad informática de las aplicaciones. Si la seguridad se deja de soslayo en el desarrollo del código de la aplicación, habrá muchos huecos que puedan aprovechar fácilmente los hackers para romper la seguridad de la aplicación, e incluso de todo el servidor y de otras aplicaciones que operen en el mismo equipo de cómputo, así que es un tema de la mayor relevancia y que debe tomarse en cuenta desde el inicio, en el desarrollo de software.

En el desarrollo de aplicaciones web, Laravel es uno de los frameworks más utilizados actualmente. Algunos datos de referencia son los siguientes: la rama *Laravel en GitHub* tiene más de 21,400 miembros, la comunidad *Laravel en español* en facebook cuenta con 23,200 miembros al día de hoy y la comunidad *Stack Overflow* en Web, tiene más de 6,200 preguntas con respuestas sobre desarrollos y casos específicos de aplicación con Laravel.

Laravel es un framework de código abierto creado en el 2011 para el desarrollo de aplicaciones a la medida con PHP, que pone énfasis en el desarrollo de código de forma organizada y sencilla bajo el Modelo Vista Controlador (MVC) para facilitar las tareas de mantenibilidad y escalabilidad de los desarrollos. Cuenta con un sistema de plantillas conocido como Blade para construir las Interfaces Gráficas de Usuario (GUI). Es compatible con la mayoría de servidores de bases de datos relacionales. También cuenta con características de seguridad preconstruidas. Estas características son las que trataremos en este artículo, además de algunas buenas prácticas que debe seguir un

desarrollador para proteger y blindar las aplicaciones desde el corazón de su código fuente y desde las propiedades de los archivos que conforman el software.

El uso del archivo .env

La primera buena práctica que debemos tomar en cuenta es colocar las credenciales de conexión a la base de datos y las credenciales para envío de correos electrónicos en el archivo .env de configuración de laravel, e incluir este archivo en el .gitignore (*Archivo .gitignore*, s/f). Esta acción tiene el propósito de ignorar el archivo de configuración y excluirlo del repositorio del proyecto. Esta acción evitará que los accesos a las bases de datos y al correo sean compartidas y eventualmente vistas por terceros, ajenos al proyecto. También de esta manera se evita dejar esta información sensible en los archivos de la aplicación. (*Gitignore Explicado*, 2021)

También es importante que al poner la aplicación en el ambiente de producción, se debe cambiar el valor de la variable APP_DEBUG a false, de esta manera, si se llega a presentar un bug en la aplicación, no quedarán expuestas las variables de configuración (entre ellas las variables con los accesos a la base de datos) del archivo .env en los mensajes de error de la aplicación ("¡Cuidado con tu archivo .env! No olvides hacer esto □", 2020).

Otra acción para proteger el archivo .env es definir el arreglo debug_blacklist en el archivo config/app.php de la siguiente manera (Thakur, 2020):

```
'debug_blacklist' => [
    '_COOKIE' => array_keys($_COOKIE),
    '_SERVER' => array_keys($_SERVER),
    '_ENV' => array_keys($_ENV),
    '_POST' => [
        'password',
    ],
],
```

Con esta configuración, lograremos que las variables de ambiente del archivo .env, las variables para cookies y las variables de los métodos post, sean reemplazadas visualmente por asteriscos, con lo que dejaremos oculto y protegido el valor real de cada variable.

También podemos definir en el arreglo debug_blacklist, variables específicas de la siguiente manera:

```
'debug_blacklist' => [
    '_ENV' => [
        'APP_KEY',
```

```
        'DB_PASSWORD',
        'REDIS_PASSWORD',
        'MAIL_PASSWORD',
        'PUSHER_APP_KEY',
        'PUSHER_APP_SECRET',
        'STRIPE_KEY',
        'STRIPE_SECRET',
    ],
    '_SERVER' => [
        'APP_KEY',
        'DB_PASSWORD',
        'REDIS_PASSWORD',
        'MAIL_PASSWORD',
        'PUSHER_APP_KEY',
        'PUSHER_APP_SECRET',
        'STRIPE_KEY',
        'STRIPE_SECRET',
    ],
    '_POST' => [
        'password',
    ],
],
```

También no debemos olvidar dar permisos 400 al archivo .env, en el sistema de archivos de linux/unix para otorgar permisos de lectura solamente al dueño del archivo (calculadoras.uno, s/f).

La seguridad de Laravel contra el ataque CSRF

El ataque Cross Site Request Forgery (CSRF) consiste en que los hackers obtienen los datos de una sesión loggeada y autorizada por un usuario legítimo en una aplicación y con esa sesión, se hacen pasar por el usuario real para efectuar transacciones en su nombre (compras de artículos, transferencias bancarias, entre otras) (*CSRF*, s/f).

El framework Laravel ya tiene preconstruida la protección contra el ataque CSRF, a través de la creación de un token para cada sesión activa de los usuarios. Este token es guardado en los datos de la sesión y es modificado frecuentemente y el framework constantemente verifica el token para revisar que la sesión pertenezca a un usuario legítimo de la aplicación. Un hacker no podrá obtener el token legítimo, puesto que éste es modificado constantemente por el framework.

Como desarrolladores, cada vez que definamos un formulario web, debemos incluir un campo CSRF en el formulario con la finalidad de que el middleware preconstruido de Laravel, pueda validar las solicitudes al servidor desde cada formulario.

Para incluir este campo, se debe agregar la directiva @csrf en el formulario de la siguiente manera (*CSRF Protection - Laravel - The PHP Framework For Web Artisans*, s/f):

```
<form method="POST" action="/profile">
    @csrf

    <!-- Forma equivalente... -->
    <input type="hidden" name="_token" value="{{ csrf_token() }}" />
</form>
```

XSS – Cross site scripting

Esta forma de ataque a la seguridad de una aplicación ocurre cuando un usuario intenta inyectar código malicioso, enviándolo a través de un formulario web. Cuando el sitio web imprime los valores de las variables que se introdujeron en el formulario (por ejemplo en un blog, dónde es típico y común hacer la visualización del texto, mensaje o post que introducen los usuarios en los formularios), es cuando se ejecuta el código malicioso en javascript, inyectado por el usuario (*Bloquear el XSS y subsanar vulnerabilidades*, s/f).

Este tipo de ataques buscan captar información sensible de la sesión del usuario y/o de las cookies, o bien, se busca captar información del usuario con la técnica phishing, enviando mensajes, dónde se solicita al usuario que introduzca información valiosa, en un sitio web aparentemente confiable, pero que está manipulado por la inyección de javascripts maliciosos.

Como desarrollador de laravel, cuando imprimimos valores en el equipo cliente del usuario, debemos utilizar la notación de dobles llaves que nos ofrece el framework.

Las dobles llaves no interpretan código html que introduzca el usuario en los formularios web. Todo aquello que imprimamos a través de las dobles llaves es renderizado como texto plano, lo cual nos ofrecen un primer nivel de protección contra la inyección de código.

Ejemplo:

```
<div>
    @php
        $html = "<h1> Título </h1>";
    @endphp

    {{ $html }}
</div>
```

Por lo tanto, la salida del ejemplo anterior será “<h1> Titulo </h1>”. Es decir, el código html no se interpreta, lo cual es bueno, porque también el código javascript que llegue a introducir un usuario en un formulario web, tampoco será interpretado. Será renderizado de manera textual.

También existe una función de PHP (strip_tags) que retira todas las etiquetas HTML, XML y PHP de una cadena de texto. Esta función recibe como primer parámetro la cadena de la que se desea retirar el código HTML, XML y PHP y como segundo parámetro recibe etiquetas que sí se permiten y que no se retirarán de la cadena recibida como primer parámetro:

```
<?php
echo strip_tags("<?php Hola <b><i>mundo!</i></b> ?>","<b>");
?>
```

La salida de este ejemplo será, la siguiente, puesto que estamos permitiendo la etiqueta <b> para negrillas:

Hola **mundo!**

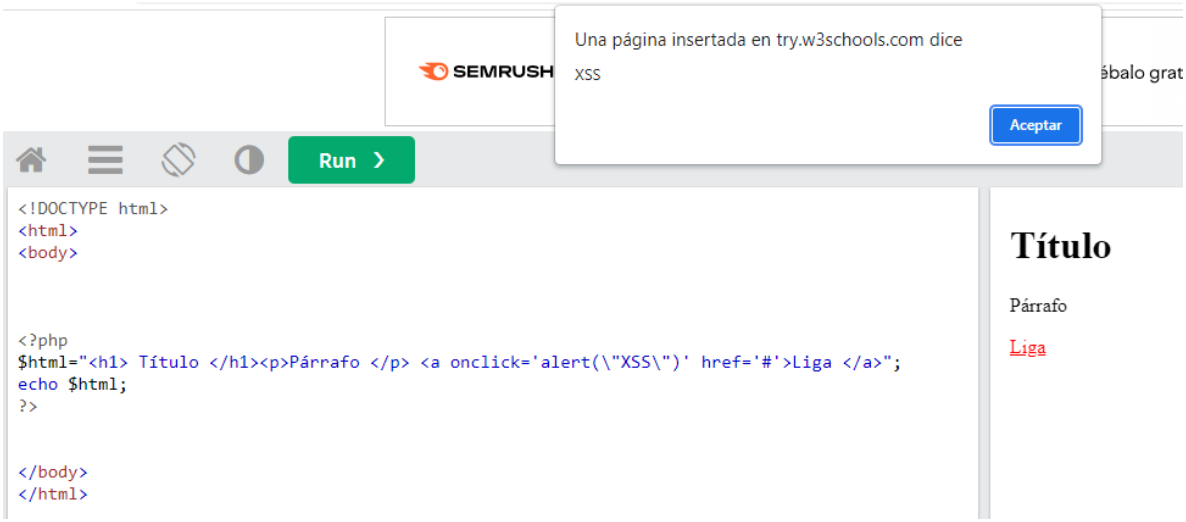
En el siguiente ejemplo, observamos cómo se deja de interpretar el código PHP de la cadena. La salida del ejemplo es una cadena vacía. No habrá renderización del contenido porque la función strip_tags elimina todo lo que está dentro de la etiqueta <?php ?>, protegiéndonos de una inyección de código.

```
<?php
echo strip_tags("<?php echo 'Hola <b><i>mundo!</i></b>'; ?>","<b>");
?>
```

Si por funcionalidad del sitio web, fuera necesario permitir la incorporación de un WYSIWYG, entonces debemos permitir la interpretación del código HTML al momento de desplegar los contenidos, incluídas las ligas HTML. Pero aquí tenemos otra vulnerabilidad, porque se puede enviar a ejecución un script de javascript a través de una liga <a href> de la siguiente manera:

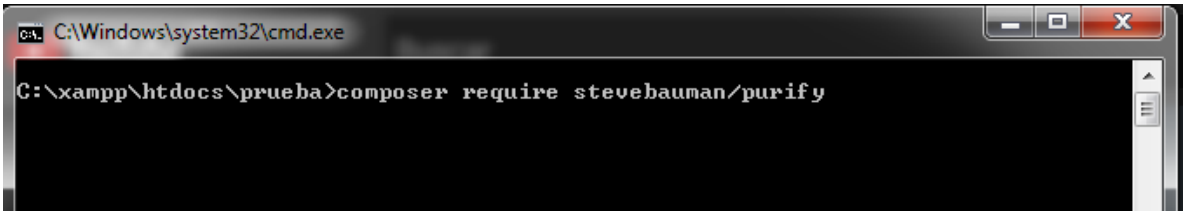
```
<?php
$html="<h1> Título </h1><p>Párrafo </p> <a onclick='alert(\"XSS\")' href='#'>Liga </a>";
echo $html;
?>
```

La salida de ese código es:

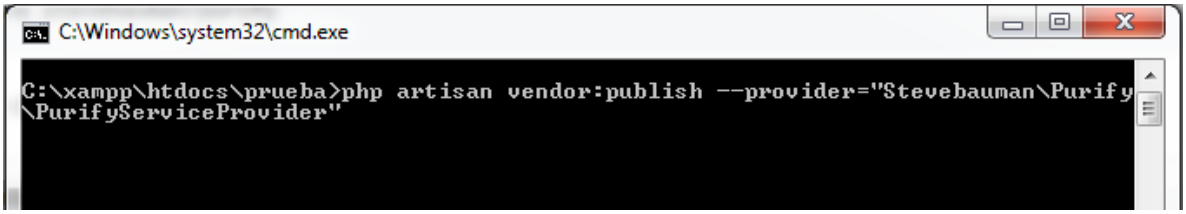


Para evitar la ejecución de un javascript a través de una liga <a href> podemos utilizar un paquete que ya ha sido creado por otros desarrolladores para limpiar el código html, el paquete se denomina *stevebauman/purify* (Bauman, 2015/2022)

La instalación de este paquete sanitizador o purificador de código se instala ejecutando desde la línea de comandos:



La documentación del paquete nos dice que una vez instalado el paquete purify, debemos publicarlo ejecutando desde la terminal el siguiente comando de php artisan:



Finalmente, para utilizar este paquete y limpiar lo que introduzca un usuario desde un formulario, llamamos a su método Purify::clean() de la siguiente manera (Aprendible, 2018):

```
<div>
    @php
        $html = "<h1>Título</h1><p>Párrafo</p><a onclick='alert(\"XSS\")' href='#'>Liga </a>
        <script>alert('ventana de javascript inyectada')</script>
        ";
    @endphp

    {!! Purify::clean($html) !!}
</div>
```

Conclusiones

Hay mecanismos de protección de la seguridad de una aplicación que son tarea exclusiva de los desarrolladores, por lo tanto, deben utilizarse y aplicarse con todo el detalle y cuidado para contribuir con esta capa de la seguridad informática.

Hay muchas tareas de seguridad en las aplicaciones que ya están resueltas, ya sea por el propio framework, o por otros desarrolladores, por lo tanto cuando nos enfrentemos a algún tema de seguridad del software, debemos recurrir a los blogs, tweets y documentación, a fin de ser eficientes y no reprogramar lo que ya está desarrollado.

Los desarrolladores debemos actualizarnos continuamente para conocer nuevos paquetes, buenas prácticas y mecanismos para blindar las aplicaciones. La seguridad informática, al igual que las TI, está en constante evolución, por lo que las buenas prácticas y recomendaciones incluídas en este texto, no son exhaustivas y deben complementarse con lecturas e indagaciones de nuevos mecanismos de protección del software, desde su código fuente.

La fecha de lanzamiento de la nueva versión de Laravel (Laravel 9) está anunciada para el 8 de febrero de 2022, por lo que debemos revisar las nuevas características de funcionalidad y seguridad

que se han contemplado en la nueva versión, a fin de actualizar nuestras aplicaciones y de incorporar nuevas técnicas y métodos de protección de la aplicación.

Referencias

Aprendible. (2018, junio 6). *Qué son los ataques XSS y cómo evitarlos en Laravel*. https://www.youtube.com/watch?v=HvhTfHOFN78

Archivo .gitignore. (s/f). Recuperado el 30 de enero de 2022, de https://desarrolloweb.com/articulos/archivo-gitignore.html

Bauman, S. (2022). *Purify* [PHP]. https://github.com/stevebauman/purify (Original work published 2015)

Bloquear el XSS y subsanar vulnerabilidades. (s/f). IONOS Digitalguide. Recuperado el 7 de febrero de 2022, de https://www.ionos.mx/digitalguide/paginas-web/desarrollo-web/que-es-el-xss-o-cross-site-scripting/

calculadoras.uno, E. /. (s/f). *CHMOD 400 r—| Calculadora CHMOD con ejemplos*. calculadoras.uno. Recuperado el 30 de enero de 2022, de https://www.calculadoras.uno/chmod/CHMOD~400~r--------

CSRF: Explicación del ataque Cross Site Request Forgery. (s/f). IONOS Digitalguide. Recuperado el 30 de enero de 2022, de https://www.ionos.mx/digitalguide/servidores/seguridad/cross-site-request-forgery/

CSRF Protection—Laravel—The PHP Framework For Web Artisans. (s/f). Recuperado el 30 de enero de 2022, de https://laravel.com/docs/8.x/csrf

¡Cuidado con tu archivo .env! No olvides hacer esto 😩. (2020, abril 16). *Laravel Tip*. https://www.laraveltip.com/cuidado-con-tu-avides-hacer-esto/

Gitignore Explicado: ¿Qué es Gitignore, y cómo agregarlo a tu repositorio? (2021, enero 19). freeCodeCamp.org. https://www.freecodecamp.org/espanol/news/gitignore-explicado-que-es-y-como-agregar-a-tu-repositorio/

Thakur, P. (2020, enero 31). Hide Environmental Variables from Laravel Debug Mode. *Tech Prastish*. https://www.tech-prastish.com/blog/hide-environmental-variables-from-laravel-debug-mode/