International Journal of Engineering Research in Computer Science and Engineering (IJERCSE) Vol 3, Issue 4, April 2016

"A review study on secure web application development using PHP with Laravel Framework"

[1] Ram Kumar Paliwal [2] Upasana Paliwal
[1] Assistant Professor Prestige Institute of Management, Gwalior (Madhya Pradesh)
[2] Freelancer, PHP Developer, Kota (Rajasthan)

***Abstract*: In this era of Internet everyone is free to access the data. Security is the main issue when we are developing any web application. Now a day's PHP is very well known name for web application development. There are so many open source PHP frameworks available for web application development. In this paper we are talking about security features available in Laravel framework for web application development using PHP language. Security is one of the core features which make Laravel as a first preference to the developer. In Laravel Hashcode, Eloquent ORM model, CSRF (cross-site request forgery) and CSS (Cross-Site Scripting) are the main key features which are used to secure web application. By using these key features, we can make our web application safe from SQL Injection and different cross site request to update the data by using unauthorized users. In addition to that Laravel is having very rich set of library to develop fast and secure web application in less time. This review paper will examine the various security features available in Laravel framework.**

***Keywords*: Hashcode, Eloquent ORM model, CSRF, Laravel**

I. INTRODUCTION:

PHP became one of the powerful platforms for web development in very short span of time. Popularity of PHP can be measured from the fact that it is installed on approx 2.5 million web servers worldwide and 300 million websites and web application are running on it. There are more than twenty frameworks and CMS available in the market on which we can develop application using PHP Language. Laravel is one of the most popular PHP frameworks that offers well tested set of library to develop fast and secure web application. There are some exclusive features available with Laravel.

A. Restful Routing

B. Inherent Database Version control

C. A lightweight Blade Templating Engine

D. Composer

E. Eloquent ORM

The system has to be secured by different methods, so the users can do various activities which must be secured for example online payments. Laravel aims to make a much secured system, for that Laravel use the famous and secure Confide PHP-Package to build our authentication system on top. Everything is configured for the developers; means need not to write the code for authentication.

II. FRAMEWORK:

Laravel framework is based on MVC (Model - View - Controller) pattern. In MVC pattern files related to the database (Backend) are stored in models; files related to the user's view (Front End) are stored in views and the controller in used as a traffic controller for the different requests and responses.

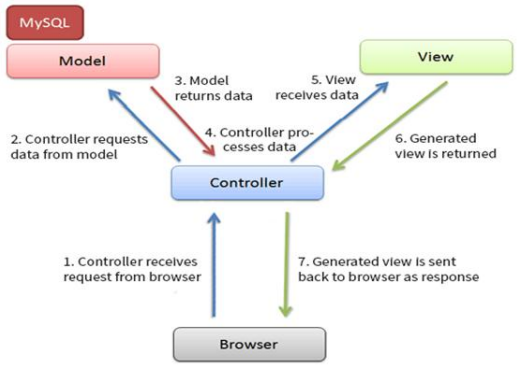

Figure 1.1

A. Application Structure:

As a Framework Laravel is having his own application structure, so that related file can be grouped on different level. Same will increase the maintainability as well security. Below is the application structure of Laravel:

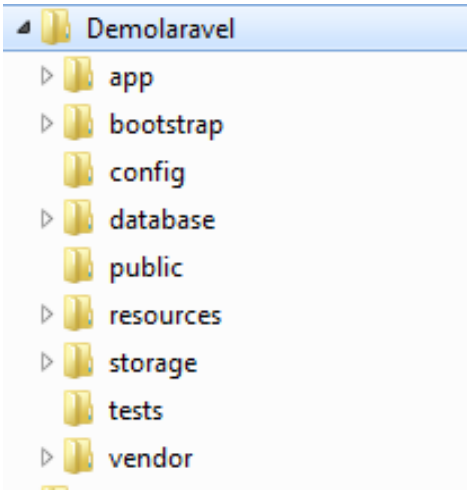

Figure 2.1

After installing the composer, by using command prompt we can create the above Laravel Application Structure, as I am using the Xampp tool which is installed on my "d:\ drive". Below command is used for creating the new project named "Demolaravel".

```
D:\xampp>cd htdocs

D:\xampp\htdocs>composer create-project laravel\la
ravel Demolaravel --prefer-dist
```

B. *Request Life Cycle:*

Request Life Cycle gives you the exact understanding that how Laravel is different from other php frameworks. "/public/index.php" file is the entry point for all request made to Laravel. The "Index.php" file is having the auto generated code which creates an object of Kernel class to perform other activity.

```
$kernel = $app->make(Illuminate\Contracts\Http\Kernel::class);

$response = $kernel->handle(
    $request = Illuminate\Http\Request::capture()
);

$response->send();

$kernel->terminate($request, $response);
```

Kernel class makes Laravel routing a bit different from other MVC framework. The following diagram explains the Laravel routing more clearly.

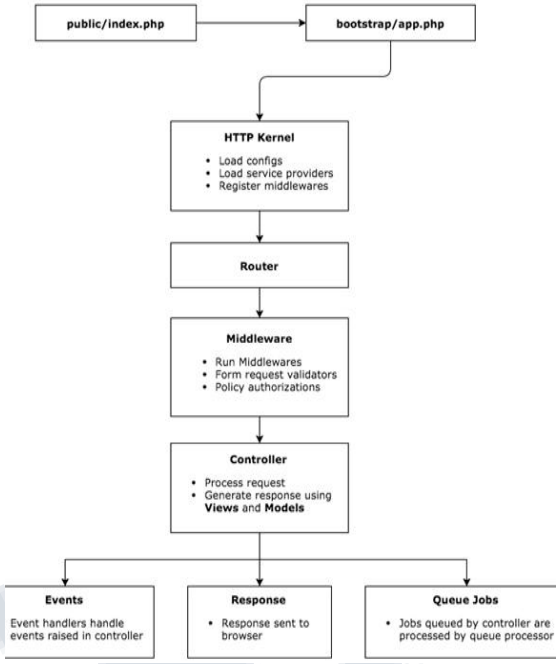

Figure 2.2

According to the figure 2.2 "public/index.php" is the entry point for all the requests made in Laravel. After that it gets an instance of Laravel application from "bootstrap/app.php", that instance load the HTTP Kernel. Kernel loads configuration, middleware and service provider and so on.

III. SECURITY IN LARAVEL:

As we have seen the Laravel life cycle section above, by which we can say that Laravel is more secured than typical MVC pattern. Besides that Laravel is using some features which make Laravel more secured.

A. *Eloquent ORM:*

Eloquent ORM uses PDO parameter binding to avoid SQL injection. Parameter binding ensures that malicious users can't pass in query data which could modify the query's intent. The Eloquent ORM provides security with an Active Record implementation to deal with database. This means that each model you create in your MVC structure corresponds to a table in the database.

A student model corresponds to students table in the database. By doing the same we can easily access the data from the database. For example to get all students from a student model, one can use student::all() function. Using this function would query the database and generate the proper SQL command.

Below is the example to create students table migration through command line.

```
D:\xampp\htdocs>cd Demolaravel

D:\xampp\htdocs\Demolaravel>php artisan make:migra
tion create_students_table --create=students
Created Migration: 2016_04_16_175307_create_studen
ts_table

D:\xampp\htdocs\Demolaravel>
```

By using above command migration get created in \database\migrations
folder

```
Schema::create('students', function (Blueprint $table) {
    $table->increments('id');
    $table->timestamps();
});
```

By default, these migrations will include an auto-incrementing "id". It will also include "timestamps" for the fields "created_at" and "updated_at". updated_at will be automatically updated whenever the record is updated. One can include other column as per the requirement after that migrate that schema on to the database by using following command.

```
D:\xampp\htdocs\Demolaravel>php artisan migrate
```

To use Eloquent ORM one must create the model by using following artisan command.

```
D:\xampp\htdocs>cd Demolaravel

D:\xampp\htdocs\Demolaravel>php artisan make:model
 Student
Model created successfully.
```

The above command create a file in \app folder with name Student.php

B. CSRF (Cross Site Request Forgery)

Cross-site Request Forgery is, is a request that the user did not intend to make. So an attacker somehow creates a forged request, and submits it on the user's behalf. Cross-site request forgery, also known as one-click attack or session riding and abbreviated as CSRF or XSRF, is a type of malicious exploit of a website where unauthorized commands are transmitted from a user that the website trusts.

For example, if you are logged in on Facebook, a hacked site could run some Javascript to make you post something under your Facebook account. Of course we don't want that to happen, so we need protection for it. And here is where the CSRF Middleware helps us.

Because of the Same-Origin security policy in browsers, you cannot read from different domain, but writes can be executed. So even though the attacker doesn't get the response, the request is still executed.

A common solution is to add a CSRF token to our form, which is generated on each request and validated on POST/PUT/DELETE requests. Because the token cannot be read, attackers can't make those requests any more.

Laravel enables the VerifyCsrfToken middleware by default to verify all requests, for the same it does the following:

- Check if the request is a reading request (HEAD, GET, OPTIONS). If so, skip the check.
- Match the token from the _token input or from the headers.
- Add a cookie with the token to each request.

This makes the CSRF check a lot more flexible. You don't have to remember where to add filter, just make sure that every form has a _token field.

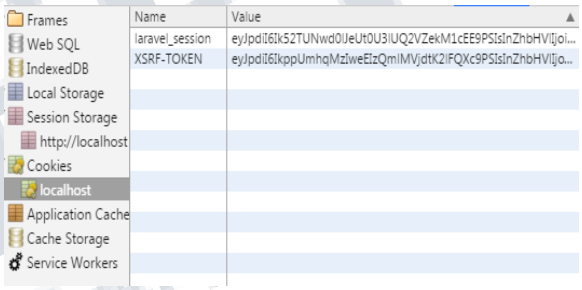

Figure 3.2

In the above figure two tokens are generated one for Laravel session and another for requested Form.

Above CSRF token generated by the VerifyCsrfToken middleware which is registered in the Kernel.php.

```
protected $middlewareGroups = [
    'web' => [
        \App\Http\Middleware\EncryptCookies::class,
        \Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::class,
        \Illuminate\Session\Middleware\StartSession::class,
        \Illuminate\View\Middleware\ShareErrorsFromSession::class,
        \App\Http\Middleware\VerifyCsrfToken::class,
    ],

    'api' => [
        'throttle:60,1',
    ],
];
```

C. Authentication

Laravel makes authentication implementation very simple. The authentication configuration implemented in auth.php file, which contains many authentication services. Laravel authentication facilities are made up of "guards" and "providers". Guards define how users are authenticated for each request. Providers define how users are retrieved from your persistent storage.

Route:

Laravel provides a quick way to scaffold all of the routes and views you need for authentication using one simple command:

```
D:\xampp\htdocs\Demolaravel>php artisan make:auth
```

The above command should be used on fresh applications and will install registration and login views, as well as routes for all authentication end-points. One of the biggest benefits of Laravel's routing is the ability to return responses directly from a route without requiring a complete controller/view structure for generating that response.

Authentication Throttling:

Authentication throttling means limiting the user login attempts. Laravel has built in AuthController class for the same. By default the user will not be able to login for one minute if they fail to provide the correct credentials after several attempts. The throttling is unique to the user's username / e-mail address and their IP address.

```
'api' => [
    'throttle:60,1',
],
```

The above code snippet from the Kernel.php file tells that after sixty failed attempts of login, user will not be able to login for one minute; the same can be customized according to the need.

Below is the example of AuthController, which is used to invoke the authentication throttling.

```
<?php

namespace App\Http\Controllers\Auth;

use App\User;
use Validator;
use App\Http\Controllers\Controller;
use Illuminate\Foundation\Auth\ThrottlesLogins;
use Illuminate\Foundation\Auth\AuthenticatesAndRegistersUsers;

class AuthController extends Controller
{
    use AuthenticatesAndRegistersUsers, ThrottlesLogins;

    // Rest of AuthController class...
}
```

Hashing:

The Laravel hashing provides secure Bcrypt hashing for storing user passwords. If you are using the AuthController controller that is included with your Laravel application, it will automatically use Bcrypt for registration and authentication.

```
$user->fill([
    'password' => Hash::make($request->newPassword)
])->save();
```

You may also use the global bcrypt helper function:

```
bcrypt('plain-text');
```

The hash should always be sixty characters long, no matter what you input as the password. You can have a five hundred character paragraph as a password, but when hashed, it will always be sixty characters long. Also, the password is not encrypted so it is impossible to decrypt the password.

IV. CONCLUSION:

Of course, there are plenty of other things you should do to further secure your Laravel application, such as ensuring browser-based error reporting is disabled so as to ensure sensitive application details aren't exposed to a potentially malicious party. However Laravel really does ensure a much more secure application. You need to do your own penetration test when your project is done to ensure everything is working and secured as planned.

REFERENCES:

1. Barry vd. Heuvel, "CSRF Protection in Laravel," September 2015. [Online]. Available: https://medium.com/@barryvdh/csrf-protection-in-laravel-explained-146d89ff1357#.4n1c4dl8m. [Accessed 2 April 2016].

2. C.Supaartagorn, "PHP Framework for database management based on MVC pattern", Department of Mathematics Statistics and Computer, Ubon Ratchathani University, Thailand, 2011.

3. Easy Laravel, "Key Security Features," July 2015. [Online]. Available: http://www.easylaravelbook.com/blog/2015/07/22/laravel-key-security-features/. [Accessed 26 March 2016].

4. Easylara, "Laravel Application Structure," September 2015. [Online]. Available: http://www.easylara.com/lesson-4-laravel-application-structure/. [Accessed 14 March 2016].

5. Laravel Book, "Architecture of Laravel Applications," 2015. [Online]. Available: http://laravelbook.com/laravel-architecture/. [Accessed 23 February 2016].

6. Laravel, "Application Structure - Laravel - The PHP Framework For Web Artisans," 2015. [Online].

Available: https://laravel.com/docs/master/structure. [Accessed 09 March 2016].

7. Laravel, "Authentication Throttling," December 2015. [Online]. Available: https://laravel.com/docs/5.2/authentication#authentication-throttling. [Accessed 12 April 2016].

8. Laravel, "Authentication," December 2015. [Online]. Available: https://laravel.com/docs/5.2/authentication. [Accessed 12 April 2016].

9. Laravel, "Hashing," December 2015. [Online]. Available: https://laravel.com/docs/5.2/hashing. [Accessed 16 April 2016]

10. Laravel, "HTTP Routing - Laravel - The PHP Framework For Web Artisans," 2015. [Online]. Available: https://laravel.com/docs/5.2/routing. [Accessed 12 February 2016].

11. Matt Stauffer, "API rate limiting in Laravel," December 2015. [Online]. Available: https://mattstauffer.co/blog/api-rate-limiting-in-laravel-5-2. [Accessed 14 April 2016].

12. P.R.Morpeth, J.Ellman, "Some Security Issues for web based frameworks", School of Computing, Engineering and Information Sciences, Northumbria University, UK, IEEE, 2010.

13. Scotch, "Eloquent ORM in Laravel," March 2014. [Online]. Available: https://scotch.io/tutorials/a-guide-to-using-eloquent-orm-in-laravel. [Accessed 20 March 2016].

14. Tinfoil Security, "Cross-Site Request Forgery," August 2014. [Online]. Available: https://www.tinfoilsecurity.com/blog/what-is-cross-site-request-forgery-csrf. [Accessed 6 April 2016].

15. TIS India, "7 Best PHP Framework For Enterprise Application," 2014. [Online]. Available: https://www.tisindia.com/blog/7-best-php-frameworks-2014/. [Accessed 19 February 2016].

16. W.Cui, L.Huang, L.J.Liang, J.Li, "The Research of PHP Development Framework Based on MVC Pattern", Conference on Computer Sciences and Convergence Information Technology, IEEE Computer Society, 2009.

17. Wikipedia, "Cross-Site Request Forgery," April 2016. [Online]. Available: https://en.wikipedia.org/wiki/Cross-site_request_forgery. [Accessed 10 April 2016].

18. (www.phpframeworks.com, diakses tanggal 19 November 2013)