

# **Web Security Application**

## **Team 1**

### **Documentation Report**

Becker, Joshua – 922465

Okereke, Uchechukwu Okpo – 930610

Shama, Muthuraman Venkatesan – 930012

Ulbrich, Damian –

June 15, 2018

**Contents**

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Architecture</b>	<b>1</b>
<b>3</b>	<b>Features</b>	<b>3</b>
<b>4</b>	<b>Tools Used</b>	<b>5</b>
<b>5</b>	<b>Team Matrix</b>	<b>5</b>
<b>6</b>	<b>Challenges and Lessons Learnt</b>	<b>6</b>
6.1	Team Challenges . . . . .	6
6.2	Lessons Learnt . . . . .	6
<b>7</b>	<b>References</b>	<b>7</b>

# 1 Introduction

Since the creation of the Internet, the world has seen it evolve with lots of practical applications. These include things like eCommerce, Instant Messaging, Video Streaming services and the likes. With this evolution and advancements, there has also been a rise in misuse and security concerns regarding identity theft, private data leaks etc.

One subset of eCommerce that is ever so popular is Online shopping, allowing consumers to buy goods or order services directly from a seller over the Internet. Hence, the element of security for user details is crucial.

In this project, we were tasked to create a simple eCommerce site called Web-shop. We look to see how a user can interact with the online store and be able to make purchases with or without registering on the site while being safeguarded against security attacks. This report, documents the required specifications of the project, alongside how the requirements were implemented. Also, the challenges encountered during the course of the project.

# 2 Architecture

Because this was a test project, a "One-Tier Architecture" approach was taken. This involves putting all of the required components for a software application or technology on a single server or platform [1]. In future, this can be scaled to an "N-Tier Architecture". As seen in figure 1, the 1-tier architecture consist of three parts: Presentation layer, Application layer and the Database layer.

- **Presentation Layer:** This is usually referred to as the application frontend. This is the layer a user or client interacts with. HTML, CSS and a bit of Javascript was used to achieve this. As an example the PageBuilder-Class was used as the presentation layer in our application which includes only presentation layer related code without almost no connection to the business logic and database.
- **Application Layer:** Usually referred to as the backend of an application. This contains all the logic needed for the application to function properly

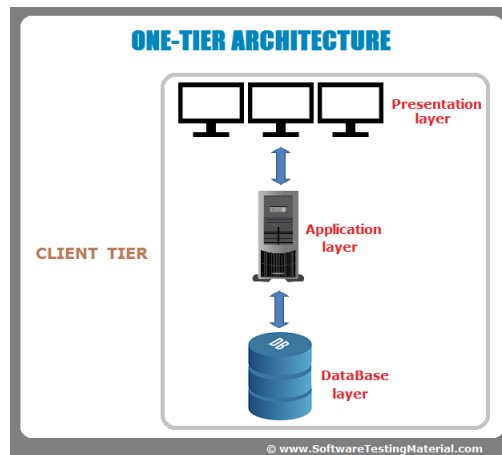


Figure 1: One-Tier Architecture

from structure to security of the application. Making this the major focus of this project and PHP (Hypertext Preprocessor) was used here. We created multiple different classes to decouple the logic from the view. The Business logic is the biggest part in our application and is represented by classes like session, data\_validation and much more.

- **Database Layer:** This is where all user and admin data were stored. MySQL which is an open-source relational database management system, was used to create, store, read and update both user and admin data. The whole Database access is encapsulated into the connect\_class which provides all the necessary method to work with the database without worrying about queries.

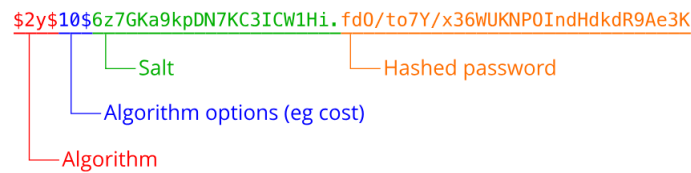
To download and install this project, one must first install **XAMPP** and clone the git repository (<https://github.com/muthuramansv/webappsecurity.git>) into the **ht-docs** folder. Start up XAMP, connect the database and run "127.0.0.1/webappsecurity/Integration/index.php" Make sure you use 127.0.0.1 instead of localhost as it is stated in the readme otherwise it will not work because of the domain-binding of the cookie.

### 3 Features

The sole criteria for this project was security based and the following features were required.

- **Secure Session Handling:** A web session is a sequence of network HTTP request and response transactions associated to the same user [2]. Web applications can keep track of both anonymous and authenticated users via a session ID, usually called a Token. A cookie is among the many forms of managing or handling a session. Since authenticated sessions contain lots of private information, they need to be protected from theft attacks which could lead to impersonation of a legitimate user. Our session is secured using several counter measurements like changing the session id every interaction and provide a timeout for every session which also automatically logs out user. Especially in our application the secure session handling is extremely important because the whole basket of every user as well as the csrf token is stored in it. For this reason we created a whole class which just provides some kind of wrapper around our session object to manage and use it easily.
- **Secure Password Storage:** To prevent theft of user credentials, passwords need to be stored securely in the database. One way of achieving this is to never store the password as plaintext, instead salt and hash the user password before storing it in the database. There are several cryptographic algorithms available for this but not all suitable to prevent brute force attacks or rainbow tables. In this project the PHP password\_hash() function was used. It is a strong one-way algorithm, which automatically salts and hashes the inputted plaintext. The output (figure 2) can then be stored in the database.
- **Cross-Site Request Forgery Handling:** This occurs when an attacker uses a malicious web site, email, blog, instant message, or program to cause a users web browser to perform an unwanted action on a trusted site

\$2y\$10\$6z7GKa9kpDN7KC3ICW1Hi.f d0/to7Y/x36wUKNP0IndHdkdR9Ae3K



The diagram shows a hashed password string: \$2y\$10\$6z7GKa9kpDN7KC3ICW1Hi.f d0/to7Y/x36wUKNP0IndHdkdR9Ae3K. Below the string, four labels with colored lines point to specific parts: 'Algorithm' (red line to '\$2y'), 'Algorithm options (eg cost)' (blue line to '\$10'), 'Salt' (green line to '\$6z7GKa9kpDN7KC3ICW1Hi.f'), and 'Hashed password' (orange line to 'd0/to7Y/x36wUKNP0IndHdkdR9Ae3K').

Figure 2: Hashed Password Sample  
Source: Safe Password Hashing, PHP Documentation

for which the user is currently authenticated [3]. In order to handle this attack a random token was generated and verified alongside. In our application we store the csrf-token as it is also called in the session of the user itself which is whenever created someone visits our site. This way our load of the database gets reduced and we are not storing unimportant user-data. To make easy use of the tokens we created several functions in the session class which provides easy access to manage the csrf-tokens in the application.

- **Cross-Site Scripting Handling:** This involves the insertion of malicious scripts into benign and trusted websites[4]. The attacker sends malicious code which is then used to steal a legitimate user cookie by redirecting the user to a different site or through pop-ups. The easiest way of preventing this attack was to escape all untrusted data before echoing it out to the user. We used method of regular expressions matching to avoid the malicious input insertion. This helped us in a way to avoid any unwanted inputs, by omitting them when the regular expression conditions are not matched and users are provided with error messages about their insertions into different tabs.
- **SQL-Injection Handling:** This consists of putting a SQL query via the input data from the client to the web application[5]. To prevent this we made use of prepared statements with parameterized queries, which en-

sures an attacker cannot change the database query even when another SQL command is inserted by the attacker.

## 4 Tools Used

Teamwork and collaboration is usually a tricky subject, but thanks to certain technologies that has helped to ease this burden. The following are a list of tools we worked with for both collaboration and application building:

- **Git:** Git is a version control system that helps track changes in project files. This really helps when coding on different platforms and machines. Source codes can be managed efficiently, generating less bugs.
- **Trello:** This is a web-based tool which helps with project management. Trello allowed us to take an Agile approach towards completion of this project.
- **WhatsApp:** WhatsApp was our default communication tool. Group meet-ups and questions were conveyed through this medium.

## 5 Team Matrix

	Becker	Okereke	Shama	Ulbrich
Project Planning	x	x	x	x
Database Design	x	x		x
XSS Handling		x	x	
CSRF Handling	x			x
Sessions Management	x		x	
Frontend Design	x		x	
SQL Injection Handling		x		x
Backend Design	x	x	x	x
Documentation	x	x	x	x

## 6 Challenges and Lessons Learnt

### 6.1 Team Challenges

We were faced with the following challenges:

- **Time:** We had to figure out time to gather so we could work together since some of us did not have prior experience in programming. We also had to struggle a lot with PHP in general, because the language was never taught in any lecture it was hard to work on it. We had to figure out how to use static values, classes and pointers in PHP just to name some issues we had. Overall, we were not sure if our application code structure is good for a PHP program. Because this is essential for writing secure code we would recommend to at least have one senior PHP developer in such a project, who can be consulted for questions.
- **Git:** Working with Git can always be tricky especially when merge conflicts arise. Lucky for us we were able to solve them. Also the huge gap between the knowledge of the the teammates where a problem. Some of us used a graphical-interface even some the CLI and others build-in IDE Plugins. To make all of them work together where challenging. Even if you are also managing documentation stuff in the repository as we did.
- **Secure Logic:** On paper it sounds simple, but figuring out how to come up with a secure logic and avoid different security task was tough. We opted for a keep-it-simple approach. We used mockups in the first to commit on an idea. After we had done this it was easier to get everybody onboard. Even a lot of challenges related to PHP arose.

### 6.2 Lessons Learnt

The following are a list of things learnt during the course of this project.

- **Coding Skills:** Every team member learnt a lot about PHP, MySQL and how various functions works. Since we developed a XAMP stack applica-



tion, we developed skills from frontend to backend. Also worth noting is it never got easier we just got better.

- **Knowledge:** There is a lot of knowledge passed across to us during the lecture time from the first lecture to the end with practical examples.
- **Team work:** This is one of the key skills that was learn in this course, we were taught how to build and work as a team which helps us in our daily lives.
- **Feedback:** We learned how to give and receive feedback in a friendly and professional way which helped to achieve the goal of delivering a successful project.

## 7 References

1. Software Architecture: One-Tier,  
<https://www.techopedia.com/definition/17374/one-tier-architecture>
2. Session Management Cheat Sheet,  
[https://www.owasp.org/index.php/Session\\_Management\\_Cheat\\_Sheet](https://www.owasp.org/index.php/Session_Management_Cheat_Sheet)
3. Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet,  
[https://www.owasp.org/index.php/Cross-Site\\_Request\\_Forgery\\_\(CSRF\)\\_Prevention\\_Cheat\\_Sheet](https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet)
4. Cross-site Scripting,  
[https://www.owasp.org/index.php/Cross-site\\_Scripting\\_\(XSS\)](https://www.owasp.org/index.php/Cross-site_Scripting_(XSS))
5. SQL Injection, [https://www.owasp.org/index.php/SQL\\_Injection](https://www.owasp.org/index.php/SQL_Injection)