Securing e-commerce against SQL injection, cross site scripting and broken authentication

Ng Yi Xuan
School of Technology
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
TP042037@mail.apu.edu.my

Julia Juremi
School of Technology
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
julia.juremi@staffemail.apu.edu.my

Nurul Husna Mohd Saad
School of Technology
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
mohamed.shabbir@staffemail.apu.edu.my

***Abstract*— World Wide Web (WWW) has been introduced in 1980s and is widely been used until today. With WWW service, publisher able to host a website in form of hypertext using Hypertext Mark-up Language (HTML). In addition, Cascading Stylesheet (CSS) is always used with HTML to manage the layout of the webpage. Over the years, the capability of HTML and CSS is getting enhanced to create a more responsive webpage. However, all these webpages creation is more towards information sharing and does not really handle user inputs. Hence, in this project, the security measures are proposed to counter these threats will be compiled as a library to be usable in any PHP-based web application. A basic but fully functional e-commerce application is developed for the testing of the proposed security features to countermeasures the mentioned vulnerabilities.**

Keywords—SQLi, e-commerce, XSS, authentication, man-in-the-middle attack

I. INTRODUCTION

As the capability of web application in boosting the development of business, web application has become the main trend of business to provide services to customers. However, more and more sensitive data is involved during the interaction of users and web application. Bank information during payment process, living address and geographical information are the examples of data involved in some web applications. These data are transmitted between the users and servers over internet. The most critical part is internet opens for everyone. Hence, web application draws the attention of attackers to steal the data from the web application.

A familiar example of web application in business is e-commerce. E-commerce, a term to describe the business process carried out via electronic medium. This business mode has first emerged in 1960s, using network to transfer business data. Nowadays, e-commerce becomes more prevalent. As discussed, a business takes big advantages of web application in selling the products and services in a convenient way. The users and the business do not require to have physical contact to buy or sell a product or service. However, as e-commerce is using web application, it tends to face the similar web application threats as discussed in last paragraph. E-commerce needs a rapid way to build their web applications to compete with others in current trend. Templates become the solution for e-commerce to build their web applications. However, these templates used may not have enough security consideration. Therefore, the final product build from the templates may easily become the victim in a cyberattack.

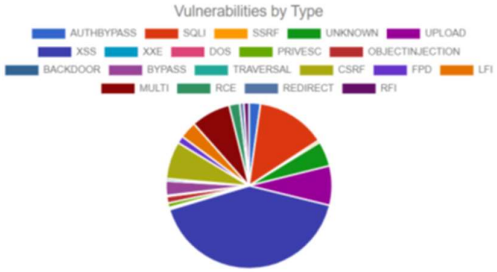

Fig. 1. Vulnerabilities in WordPress. Source: WPScan

According to a recent report by Symantec, a rise of 56% in web attacks has been observed in the year 2019. Among all the web application threats, this project will focus on SQL injection, Cross-site scripting and broken authentication. Fig.1 shows the latest detected vulnerabilities in WordPress. WordPress is a famous system used by SMEs, including e-commerce web application. It also provides template for SMEs to ease the web applications development. However, web application build with the templates is targeting by tons of web vulnerabilities. SQLi and XSS are ranked top among these vulnerabilities. This is because the template may not be reconfigured to suit the structure of developed web applications.

Hence, leading the structure of web application to vulnerable to many types of web attacks. For example, if an e-commerce web application is using template without implementing the proper security features, the product may easily suffer from the mentioned vulnerabilities. In short, weak security implementation usually happens in any other template generates web application.

II. DOMAIN RESEARCH

A. Injection

Injection is a type of attack that done by providing any malicious statement in the input fields provided in a web application. SQL (Structured Query Language) injection is one example of the injection attack. This attack as illustrated in Fig.2 is targeting the database to obtain the desire information from the database or performs some harmful actions to the database.

SQL injection was first promoted in an article titled “NT Web Technology Vulnerabilities” written by [2]. In this articles, Microsoft SQL and ASP injection are discussed as an example. However, Microsoft claims that the finding of

Rainforest Puppy is incorrect, as it is a feature of SQL. The demonstration of the exploitation in the subsequence publication of the author has verified the argument of Microsoft is false. Hence, this is also the first successful SQL injection known by public.

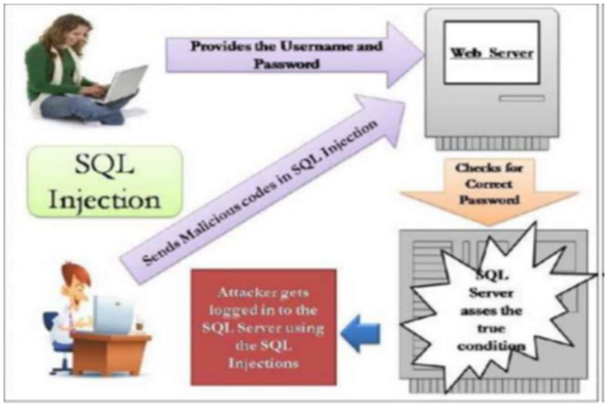

Fig. 2. SQL Injection flow

Normally, web application always provides input fields to receive request from users. Hence, these input fields become the "door" for attackers to gain access to the database. In its tenure, these input fields are expecting the correct data from users. However, attackers will input some malicious codes in these fields and send to the server to retrieve some private data in the database or perform some actions to the database. These malicious codes aim to change the condition to always true and allow the request to execute. At the end, database returns what the attackers is desire. There are 3 common types of SQL injection: tautologies, union query and blind injection [1].

B. Cross-Site Scripting (XSS)

Generally, XSS is also one type of injection attacks, but works in a different way with SQL and PHP injection. XSS occurs when a website is injected with malicious scripts to pretend it is genuine and trusty. The process of XSS starts when a web application is using by attackers to send harmful scripts to its end users. In contrast to other injection attacks, XSS is targeting the users of the web application while SQL and PHP injection is targeting the web application itself [3].

At early stage, XSS is divided into 3 categories: Stored XSS, Reflected XSS and DOM Based XSS [4].

• Stored XSS happens when the storing process occurs at the server. During the process of storing data, the malicious scripts may redirect the data to save on another location defined by attacker. If the payload is eternally stored in user's browser, the user's data can never reach the true server.

• Reflected XSS happens when the feedbacks from the server is invoked. In this kind of XSS, user data is not stored in the server, but server response to these data or requests.

• DOM Based XSS happens within the browser itself, does not interact with server at all. The flow of data is absolutely controlled by DOM (Document Object Model), from data source to the data sink.

C. Broken Authentication

Broken authentication is a term that covers all improper configurations for authentication process. In web application, broken authentication more likely to indicates the improper way in handling authentication data such as session, user credentials and sensitive data in URL. Generally, a session is created by server once the user logged in. The purpose of creating session is to handle the communication between the specific user and server. In simple word, the session holds the user data for communication purpose such as user ID.

There are few vulnerabilities to exploits broken authentication. One good example of it is the insecure way to store the user credential. As per sign up, the user data should be inserted into database. However, if the password is not hashed and salted, it can be easily leaked once the password improperly shown in any page of the web site. For instance, the developer with less knowledge in security may pass the user credential using "GET" method. "GET" method will leave history in browser and whoever manage to access the browser history may obtain this password. The worse is the password are shown in clear text without any encryption.

In addition, the session is another way for attacker to access the system without creating legit credential on his own. Therefore, this creates a chance for attacker to steal the legit user session if the session is not configured and managed in well manner. For instance, if session timeout is not set, the session of user left valid forever. Attacker may use this session to access the system as the user because the session remains valid and true for authentication purpose. Another possible vulnerability to obtain the session from a user is via URL. Some insecure websites may pass the session in the URL itself when using the web application. For example, a user logged in and sends URL contains session to his friend to share an information from the website. This leak the user's session to his friend and the URL actually allows his friend to access the web application with the user's session. Attacker may user phishing email or Man-In-The-Middle attack to obtain the URL contains session. Hence, a secure way should be practiced in creating and passing the session.

III. PROPOSED COUNTERMEASURES

A. Input Sanitization

To prevent SQL injection, [5] has proposed an algorithm to check for SQL statement inputs from users. For the first phase in the proposed algorithm, DROP keyword is detected to prevent any table is deleted by the users. Next, structure of the statement should be start with SELECT, INSERT etc. to check for validity. In 3rd steps, common SQL injection, '1' = '1' or similar syntax is scanned. On rest of the phase, any query to detect the table structure or to know the tables in the database is also the malicious target to avoid in this algorithm. The results of implementing this algorithm is excellent and able to prevent simple SQL injections.

Similar to previous SQL injection prevention method, [6]. First, all the SQL injection keywords are filtered from the inputs. These keywords are SELECT, UPDATE, DELETE, INSERT, TRUNCATE, DROP, logical operators and special symbols. Next, the type or length of input data is determined to match the data in database. Authors also implement the methodology to avoid SQL command stitching and SQL Injection with Apache Server's Rewrite Module.

[7] has proposed the pattern filtering methods to detect a cross-site attacks. All the possible malicious contents are scanned using the XSS filters in this implementation. After scanning, the malicious content is replaced with null

characters to prevent the execution of the code. All the results of filtration are stores in database to improve the accuracy of the proposed application in future scanning. Any similar pattern detected in next scanning will be blocked immediately. All the tested XSS are blocked successful except for the new pattern of XSS performed to the prototyped website.

B. Content Security Policy

According to [8], cross-site scripting (XSS) can be avoided using content security policy (CSP) (2016). CSP determines the interaction method between the content and the website to tell the browser either it can be executed and displayed. CSP prevents the XSS attacks by forcing the use of CSP's directives only. CSP's directives is a set of rules to tell compiler how the inputs should be processes. In CSP's directives, inline scripts such as JavaScripts are forbidden. The authors able to achieve a good result by mitigating all XSS attack types on a prototype website in 4 popular web browsers.

C. Secure Authentication Configuration

In the journal written by [9], session hijacking can be prevented by implementing a strong and complex session ID. To generate a strong and complex session ID, it must be always random, without any distinct logic. Despite of random generation, the session ID should be long enough. Fulfilling these two points help to increase the difficulty of brute forcing the session ID by attackers. This is because brute force tries every possibility to reach the correct answer, which may take very long time to brute force a long and complex ID.

The prevention of session hijacking can be simple by implementing logout functionality and timing out session. Logout functionality is one of the fundamental procedures in complete authentication process. Logout ends the session between the user and server, thus prevent a valid session to be stolen. Timing out session is also essential to ensure a session will automatically expired if leaving for a period of time. This is to evade reuse of session by attacker if users forget to logout themselves.

IV. IMPLEMENTATION

A. Prepared Statement

Prepared statement as shown in Fig. 3 is implemented in every part of code that required communication with database. A class is designed to manage all these communications to reduce redundancy of codes in the e-commerce web applications. A function is created in the class to conduct the full process of using prepared statements. Any other functionalities that communicate with database will rely on this function to use prepare statement.

A static function get() is created to retrieve the data from the Global variable configuration in the e-commerce web application. Below line shows example of using the function:

Config::get('myql/host')

Above command will returns the host address of the MySql database which is set in Global configuration. Once an instance of DB is created, the PDO will be created by the following command and put into private variable $_pdo:

new PDO(host; dbname, username, password)

```
class Config {
  public static function get($path = null) {
    if($path) {
      $config = $GLOBALS['config'];
      $path = explode('/', $path);


      foreach($path as $bit) {
        if(isset($config[$bit])) {
          $config = $config[$bit];
        }
      }

      return $config;
    }

    return false;
  }
}
```

```
class DB {
  private static $_instance = null;
  private $_pdo,
          $_query,
          $_error = false,
          $_results,
          $_count = 0;

  private function __construct() {
    try {
      $this->_pdo = new PDO('mysql:host=' . Config::get('mysql/host') . ';dbname=' .
                            Config::get('mysql/db'), Config::get('mysql/username'),
                            Config::get('mysql/password'));
    } catch (PDOException $e) {
      die($e->getMessage());
    }
  }

  public static function getInstance() {
    if(!isset(self::$_instance)) {
      self::$_instance = new DB();
    }
    return self::$_instance;
  }
```

Fig. 3. Config class and PDO initiation

The construction of DB coordinator can be done by the static function getInstance().A query is then created using the PDO prepare function with the input of SQL statement and stored in private variable $_query. If the preparing process produces error, the rest of the code will not be executed. If it is success, the number of parameters passed is counted. Variable $X is auto incremented for each parameter. $X is used to bind value for prepared statement. For instance, if

*$sql = 'SELECT * FROM user WHERE id = ?, username =?'*

$params = array (1, 'admin')

The below lines illustrate the output of the binding process:

bindValue ($X, $param) => bindValue (1, 1) => bindValue (2, 'admin')

Notes that $param represents each value in the $params array. $X variable will determine the position of "?" to bind the value for. After binding value, the query is then executed. If the execution success, the results retrieved from database with the function fetchObject(). This function retrieves each row of data as an object. The results are stored into private variable $_results. Function rowCount() is used to count the total number of row of data retrieved.

There are specific functions created to retrieve private variable from the DB class. The query function becomes the fundamental to conduct communication to database. Any other functions that require database communication will call this function.

B. Entities Encoding

All the inputs from user that going be shown in the browser is passed into a function to replace any HTML entities equivalent characters. The malicious input passed can be sanitized and input as HTML entities in source code. The identical input will be shown in the browser with no action trigger by the malicious input.

C. Hashing and Salting

The user password is added a random set of characters that is 32-bit length. After the salting process completed, the output is hashed with SHA256 algorithm. The hashed output is then stored in the database with the correspond salt value used. In other words, the proposed e-commerce web application strengthens the authentication process with enhanced hashing with salt and never store the plain text of user password.

```
$user = new User();
$salt = Hash::salt();

$email = escape(trim(Input::get('email')));
$username = escape(trim(Input::get('username')));
$password = escape(Input::get('password'));

try {
  $user->register(array(
    'email' => $email,
    'username' => $username,
    'password' => Hash::make($password, $salt),
    'salt' => $salt
  ));
  $user->login($username, $password, $salt);

  Redirect::to('index.php');
} catch (Exception $e) {
  die($e->getMessage());
}
```

Fig. 4. Hash Usage

Fig.4 shows the usage of hashing and salting in the e-commerce web application. The function is used when registering a new user. A random salt is first generated, then passed into make() function together with password input by user. At the end, the password is hashed before passing into database. The database will only store the hashed password and correspond salt value used.

D. User Permission Checking

For restricted page, the e-commerce web application will conduct a user permission checking to verify the user to have permission accessing the particular page. If user is valid to access the page, the user will be redirected to the home page of the e-commerce web application. After checking the unit test, the researcher noticed that any part of the secure online attendance system functions without any problems. The developer found that there were a few areas without operation, so that a consistency so reliability check needed to allow development. The developer has now implemented some improvements to indicate signs of improvement and eliminate deformities. The project has been collecting some consumer feedback about what they would like to learn later by utilizing product approval research. It will encourage the creator to change the program to make the customer feel more comfortable with the program and pleased.

IV. SECURITY TESTING

After conducting the basic functionalities testing and security testing focuses on the three proposed security features, the proposed e-commerce web application is considered well-developed. The result in Fig.5 shown that every test case is encouraging and very good. All the components in the web application is functioning and producing the expected output. The security features are also functioning well in preventing the designed payloads for each attack. In short, this chapter guarantees the functionality of developed web application and the effectiveness of implemented security features in tackling the SQLi, XSS and broken authentication.

SQLi

Test ID	Description	Expected Output	Actual Output	Result
1	Insert ' or 1=1 - -; at every input fields	System give normal response without unpredicted data shown	System give normal response without unpredicted data shown	Pass
2	Insert ' or 1=1 - -; at URL passing data for communicating database	System give normal response without unpredicted data shown	System give normal response without unpredicted data shown	Pass

XSS

Test ID	Description	Expected Output	Actual Output	Result
1	Insert <body onload=alert('test1')> at every input fields	System give normal response without unpredicted data shown	System give normal response without unpredicted data shown	Pass

Broken authentication

Test ID	Description	Expected Output	Actual Output	Result
1	Decrypt the hashed password in hashkiller.com	No result obtained	No result obtained	Pass
2	Access admin pages with normal user permission	Redirect user to home page	Redirect user to home page	Pass

Fig. 5. Unit testing on SQLi, XSS and broken authentication

V. CONCLUSION

The most critical issue in the proposed e-commerce web application is it prioritizes SQLi, XSS and broken authentication. The e-commerce web application developed able to demonstrate the business flow of e-commerce in a secure way. All the proposed countermeasures that tackle SQLi, XSS and broken authentication is critically evaluated from its effectiveness. Comparing to an e-commerce web application without any security measures, the simple steps taken to build the proposed e-commerce web application have significantly increase the security level of it. The end result of this project is securing e-commerce web application against SQLi, XSS and broken authentication with simple steps taken.

REFERENCES

[1] Z. S. Alwan and M. F. Younis, ''Detection and prevention of SQL injection attack: A survey,'' Int. J. Comput. Sci. Mobile Comput., vol. 6, no. 8, pp. 5–17, 2017. [Online]. Available: https://www.ijcsmc.com/docs/papers/August2017/V6I8201701.pdf

[2] A. A. Sarhan, S. A. Farhan, and F. M. Al-Harby, ''Understanding and discovering SQL injection vulnerabilities,'' in Proc. Int. Conf. Appl. Hum. Factors Ergonom., 2017, pp. 1063–1075.

[3] OWASP, 2018. Cross-site Scripting (XSS). [Online] Available at: https://www.owasp.org/index.php/Cross-site_Scripting_(XSS) [Accessed July 2019].

[4] OWASP, 2017. Types of Cross-Site Scripting. [Online] Available at: https://www.owasp.org/index.php/Types_of_Cross-Site_Scripting [Accessed July 2019].

[5] A.Gupta. htmlentities() vs htmlspecialchars() Function in PHP. [Online] Available at: https://www.geeksforgeeks.org/htmlentities-vs-htmlspecialchars-function-in-php/ [Accessed December 2019].

[6] H. Zhang and X. Zhang, "SQL Injection Attack Principles and Preventive Techniques for PHP Site," in Proceedings of the 2nd International Conference on Computer Science and Application Engineering, 2018.

[7] S. Bongiri, M. A. Garcia, "Using Pattern Filtering to Detect Cross-Site Attacks," Las Vegas, International Conference on Security and Management, 2017.

[8] I. Yusof and A. K. Pathan, "Mitigating cross-site scripting attacks with a content security policy," IEEE Computer, vol. 49, no. 3, pp. 56–63, 2016.

[9] A. Gupta, Dr. S. K. Yadav, "An Approach for Preventing SQL Injection Attack on Web Application", International Journal of Computer Science and Mobile Computing , vol.5, issue. 6, pp. 01-10, June 2016.