Report for DDSS A2

Klemen Janez Poličar

# Introduction

I have written the application in PHP. I've installed three libraries that I use within the project using Composer as a package manager:

1. **illuminate/support** is used for helper functions, such as encoding special HTML characters in a string
2. **illuminate/hashing** is used for hashing passwords using the bcyrpt algorithm
3. **symfony/var-dumper** is used for cleaner debug functions

# Part 1.1

A03:2021-Injection:

Every input in the form takes the value from the URL query string ($\_GET ). The difference between the vulnerable and correct form is that the correct form escapes the value from the query string, whereas the vulnerable does not, which introduces a XSS vulnerability.

A03:2021-Injection:

The vulnerable form request handler does not sanitize the input parameters when building the SQL query, which introduces an SQL injection vulnerability. The correct form uses the built-in PHP function *pg\_query\_params* to safely build a query.

A03:2021-Injection:

The vulnerable form request handler returns an error message without having escaped the username that is interpolated in the message. This introduces an XSS vulnerability.

A04:2021-Insecure Design:

The vulnerable form error request handler (incorrect username or password) redirects the user back to the login page with both the username and password sent in the query string, to allow for quick fixes by the user. This exposes the password as cleartext and is considered bad practise. The correct form only provides the username in the query string, so the user must enter his password again before submitting.

A04:2021-Insecure Design:

The vulnerable form error request handler gives too much information about the failed login attempt – it tells the user if their exists an account with the provided email. This is considered bad practise. The correct form only replies with a failed attempt, without giving detail about the existance of an account with the provided email address.

A03:2021-Injection:

The vulnerable form request handler does not encode the special HTML characters text in the page where the username is displayed (when the form is submitted and the username does not exist, an error message is displayed which contains the submitted username), so there is a second order XSS vulnerability. The user's username may contain HTML tags, which is later interpolated as valid HTML on the webpage.

Good Practice:

Once authenticated, the user's authentication is stored in his session data – meaning on the server. This mitigates impersonation attacks.

# Part 1.2

A03:2021-Injection:

Every input in the form takes the value from the URL query string ($\_GET ). The difference between the vulnerable and correct form is that the correct form escapes the value from the query string, whereas the vulnerable does not, which introduces a XSS vulnerability.

A03:2021-Injection:

The vulnerable form request handler does not sanitize the input parameters when building the SQL query, which introduces an SQL injection vulnerability. The correct form uses the built-in PHP function *pg\_query\_params* to safely build a query.

A01:2021-Broken Access Control:

The vulnerable form request handler checks authentication using post parameters rather than through the session. The post parameters can be manipulated by the end-user and since the authentication method of the app is simply to authenticate the user with the username that is provided, he can simply provide any registered user's email to achieve impersonation. In short, he is able to create comments in the database authored by any registered user. This differs in the correct form request handler, which uses the session to verify user authentication.

A03:2021-Injection:

The vulnerable form request handler does not sanitize the inputed text, nor does it encode the special HTML characters text in the page where the text is displayed, so there is a second order XSS vulnerability. The user may send HTML tags as content of the comment's text, which is later interpolated as valid HTML on the webpage.

# Part 1.3

A03:2021-Injection:

Every input in the form takes the value from the URL query string ($\_GET ). The difference between the vulnerable and correct form is that the correct form escapes the value from the query string, whereas the vulnerable does not, which introduces a XSS vulnerability.

A03:2021-Injection:

The vulnerable form request handler does not sanitize the input parameters when building the SQL query, which introduces an SQL injection vulnerability. The correct form uses the built-in PHP function *pg\_query\_params* to safely build a query.

A04:2021-Insecure Design:

The vulnerable form request handler takes the database connection credentials from the request parameters (GET/POST). This is insecure, since it means the client must provide the database credentials, which is information he should not have.