Week 6 Homework 4

Matt Rieser

SDEV 325 – Section 6381

October 14, 2018

Professor Zachary Fair

# Missing Encryption of Sensitive Data:  CWE-311

## Description

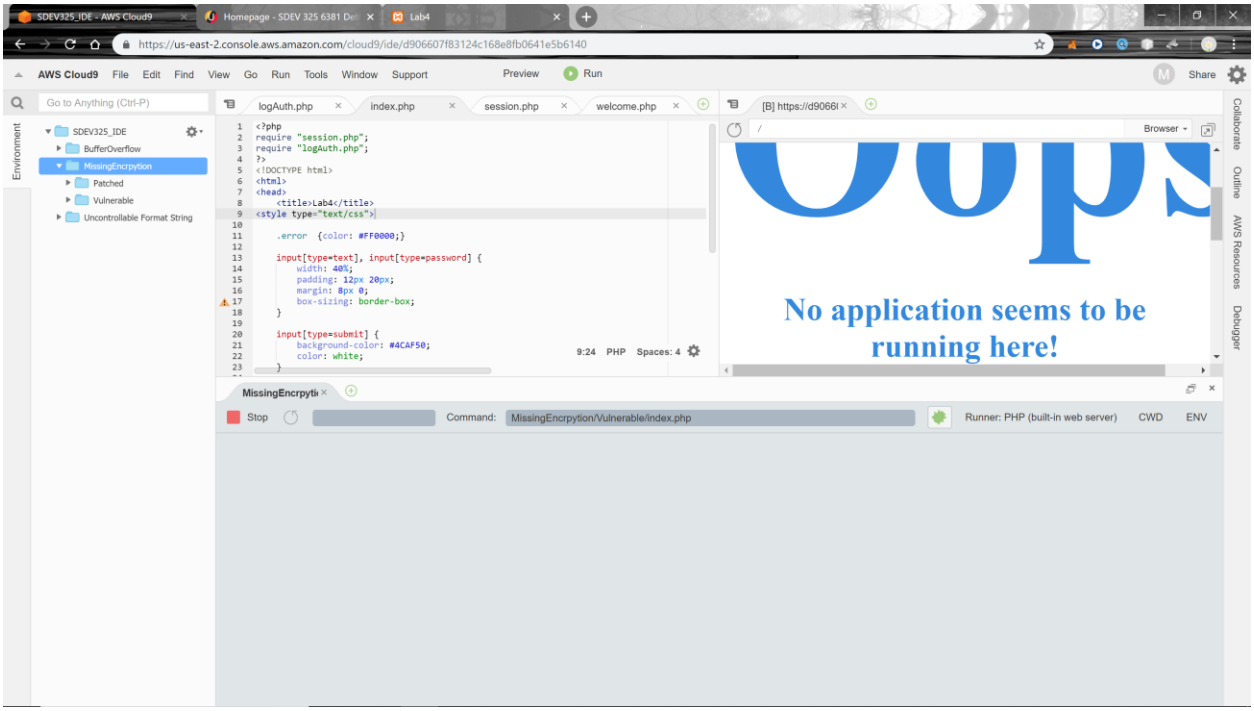
The software does not encrypt sensitive or critical information before storage or transmission.

## Extended Description

The lack of proper data encryption passes up the guarantees of confidentiality, integrity, and accountability that properly implemented encryption conveys.

Resource - https://cwe.mitre.org/data/definitions/311.html

Could not get it to work properly on aws cloud 9 for some reason so I ended up




using xampp

The application stores a client side cookie, containing the username + password, in clear text.  This is a major security issue because user login information is exposed if their computer is compromised by an attacker. This weakness combine with cross-site scripting can allow an attacker to remotely capture cookie data (i.e., user login information).

## Demonstration

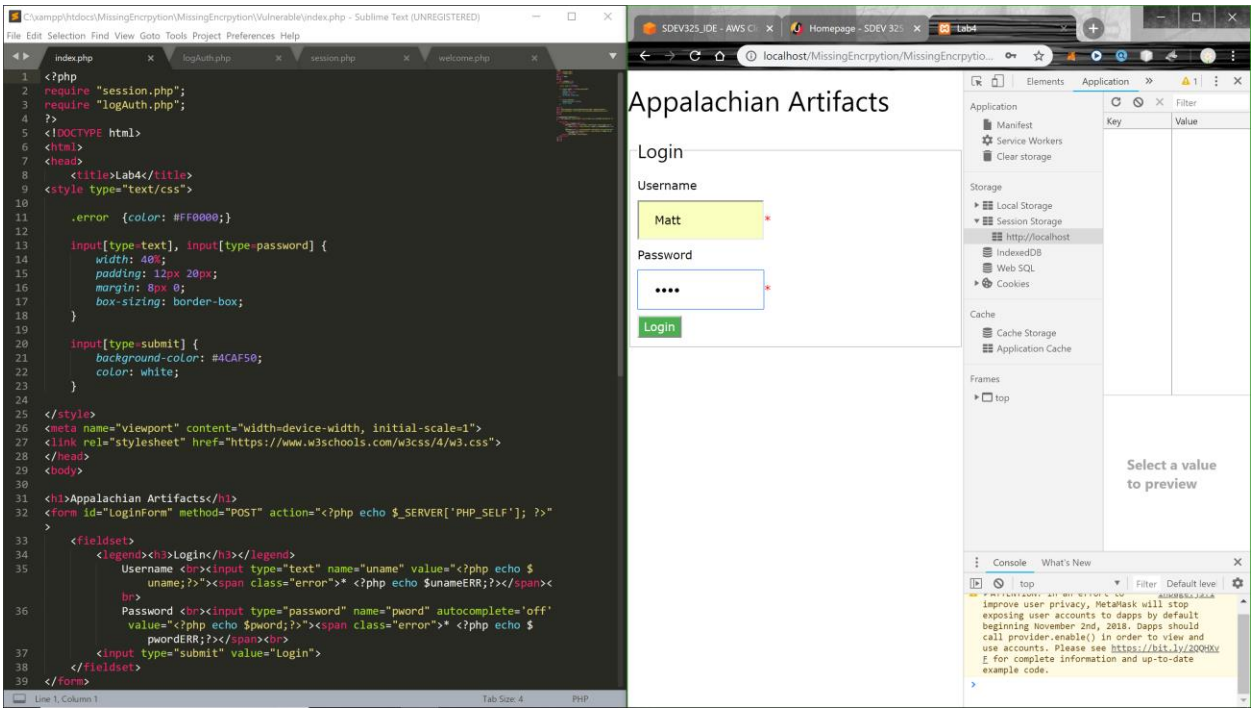



*Figure 1. Demonstrating Porous Defenses. This figure shows the default page for the web application (right) and its underlying code (left).*

In *Figure 1*, right side, the application displays a basic login form asking for username and a password. Also, notice at the bottom the current cookies for http://localhost:8888 are shown, which currently is empty. Glancing to the left, the source code indicates the form values will be submitted to the same page that is currently running and handled by the file **logAuth.php** that is imported at the top of the page, along with **session.php**.

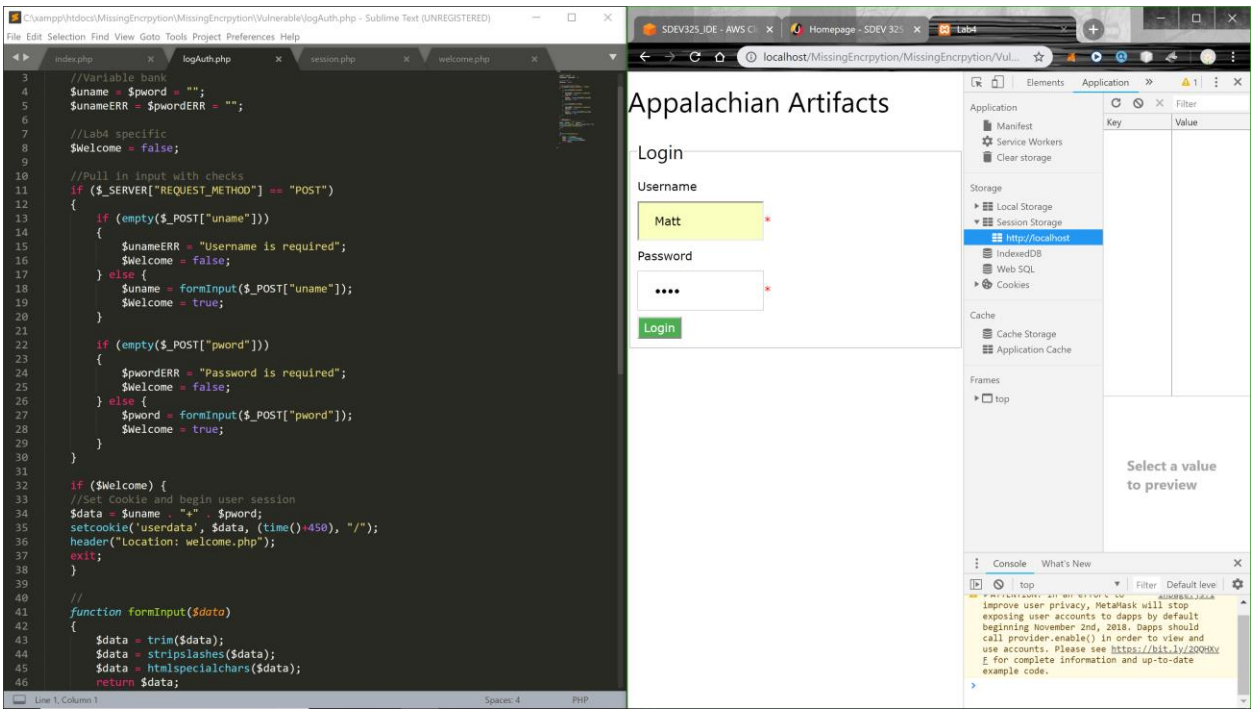



*Figure 2. Demonstrating Porous Defenses. This figure shows the default page for the web application (right) and its underlying code (left).*

*Figure 2* only differs from *Figure 1* on the left side where the code is shown. Instead of **index.php** being visible **logAuth.php** is now in focus. This php code essentially sanitizes and validates that there is a username and password (lines 11-30), doesn't matter what, and keeps track of authentication via the $Welcome variable. The function at the bottom (beginning on line 41) aids in the validation of input which is called in the block of if else statements above. If the $Welcome variable is true the username and password are concatenated into the $data variable and stored in the cookie, which is set just below that on line 35. Line 36 and 37 will send a user to the **welcome.php** and exit the currently running script.

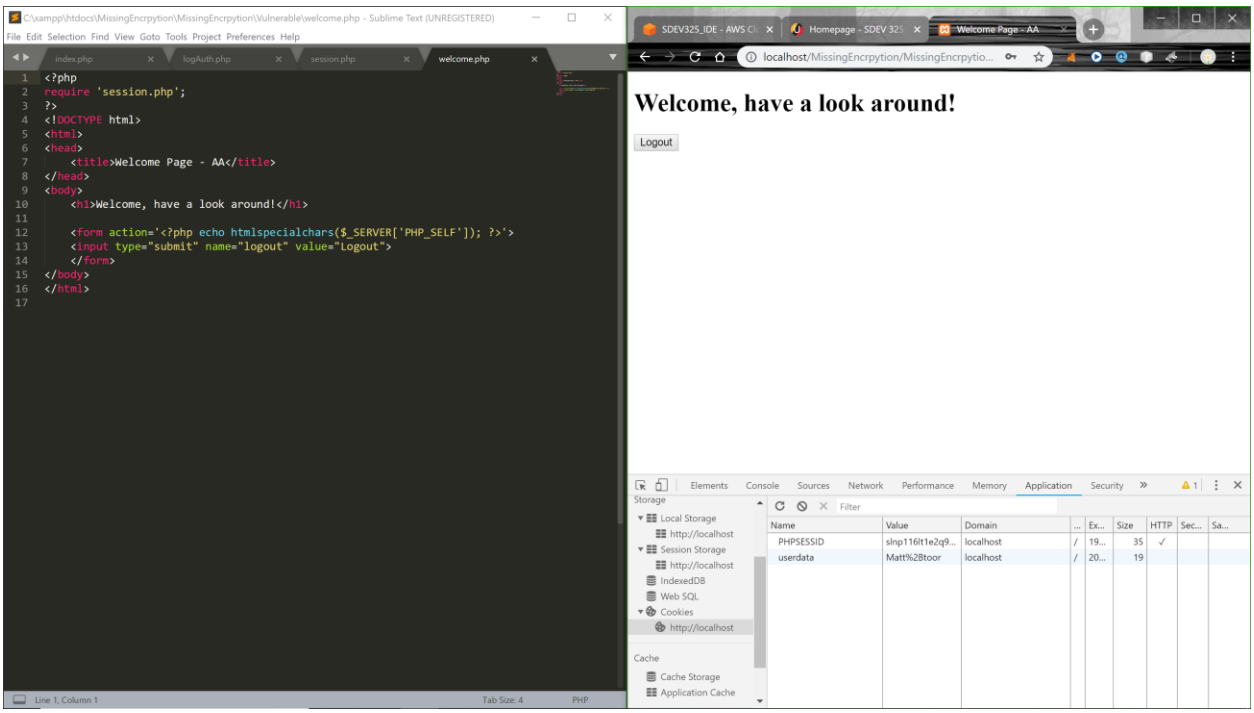



*Figure 3. Demonstrating Porous Defenses. This figure shows the landing page for the web application (right) and its underlying code (left).*

The fancy welcome page greets users after they have authenticated. Mainly, notice that there is now a cookie stored on the client side (*Figure 3*, bottom right) and the value is in plain text (e.g., Matt%2Btoor). As mentioned before this is dangerous even if the client machine has not been compromised due to cross-site scripting. It is never a good idea to store sensitive data, especially unencrypted, via cookies on the client side. Session functionality is a good alternative whenever possible, however, cookies are sometimes the only option. The code on the left indicates that the logout button will utilize the **session.php** imported on line 2.

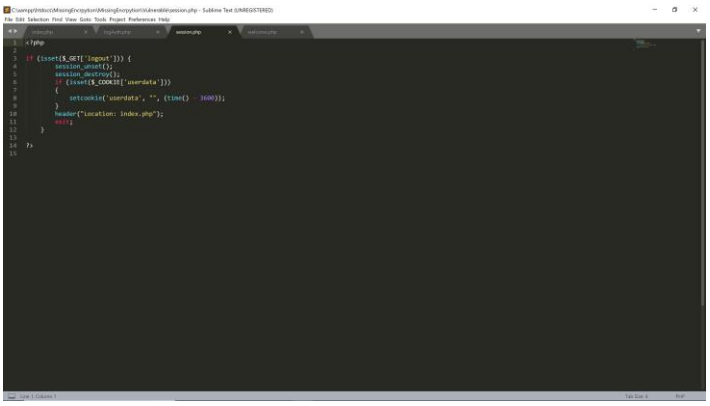


*Figure 4. Demonstrating Porous Defenses. This figure shows session.php code which supports the web application.*

In *Figure 4*, notice the code is checking for logout being set, which would indicate that the logout button had been clicked. **Note:** lines 4, 5 were later removed without impact to the program as this application is not using sessions. If the userdata cookie is

set it will be forced to expire, thus removing it. The user is then set back to **index.php** page via line 10.

## Patched Application

To fix this lack of encryption, a function will be added to **logAuth.php** and utilized to encrypt the data set in the userdata cookie.

## Demonstration

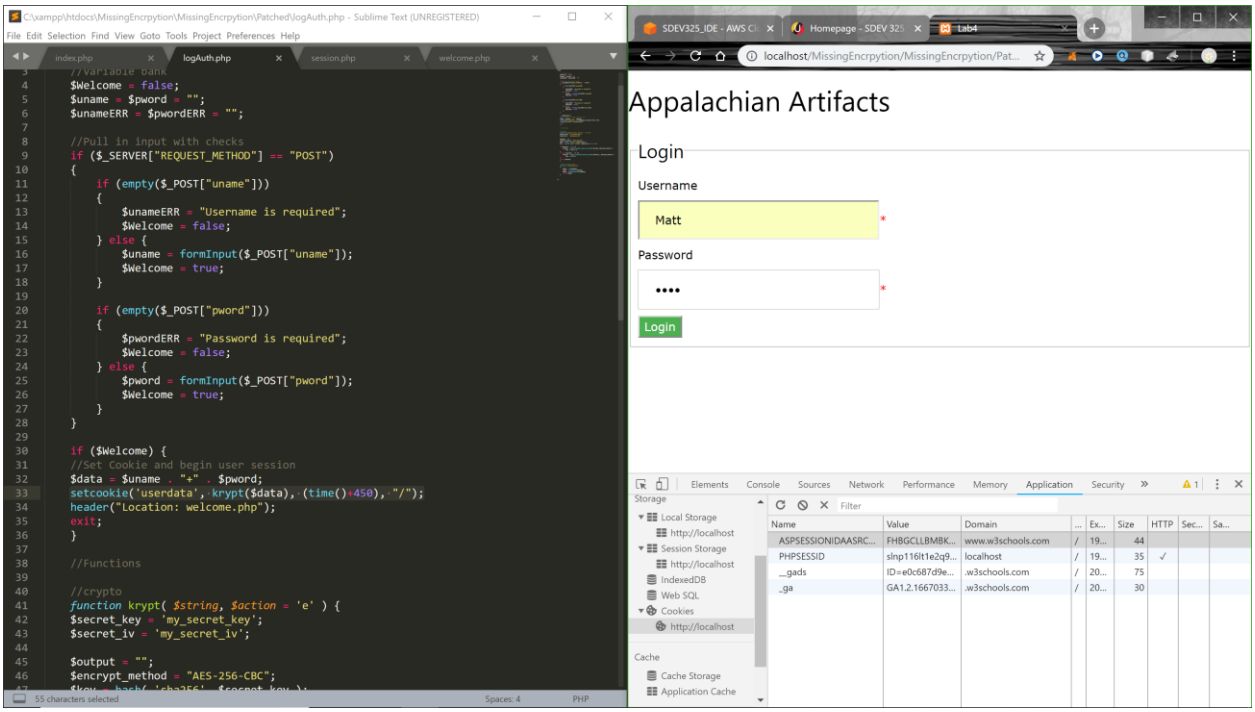



*Figure 5. Demonstrating Porous Defenses. This figure shows the default page for the web application (right) and its underlying authentication code (left).*

*Figure 5*, shows a login attempt about to be made on the right in the Google chrome browser and shows **logAuth.php** on the left, to which, the "krypt" function has been added (lines 41-57). The function is implemented on line 33 only for the cookie data.

The krypt function accepts two parameters: string to be encrypted, and 'e' or 'd' for encrypt and decrypt, respectively. If neither 'e' nor 'd' is passed the function defaults to encrypt, as seen on line 33, *figure 5*. This function utilizes the openssl_encrypt() and openssl_decrypt() PHP functions. The encryption method is AES-256-CBC which uses Advanced Encryption Standard (AES), 256bit key size, and Cipher Block Chaining (CBC) mode for

encryption. There is some debate about the different modes, CBC vs CTR (Counter), being more secure than the other. I went with CBC due to it being less susceptible to certain side channel attacks, such as frequency analysis.

Although the decrypt portion of the function is not actually used this could be a way to read the sensitive data, update it, and reset the cookie on the client side. It has been included for completeness.

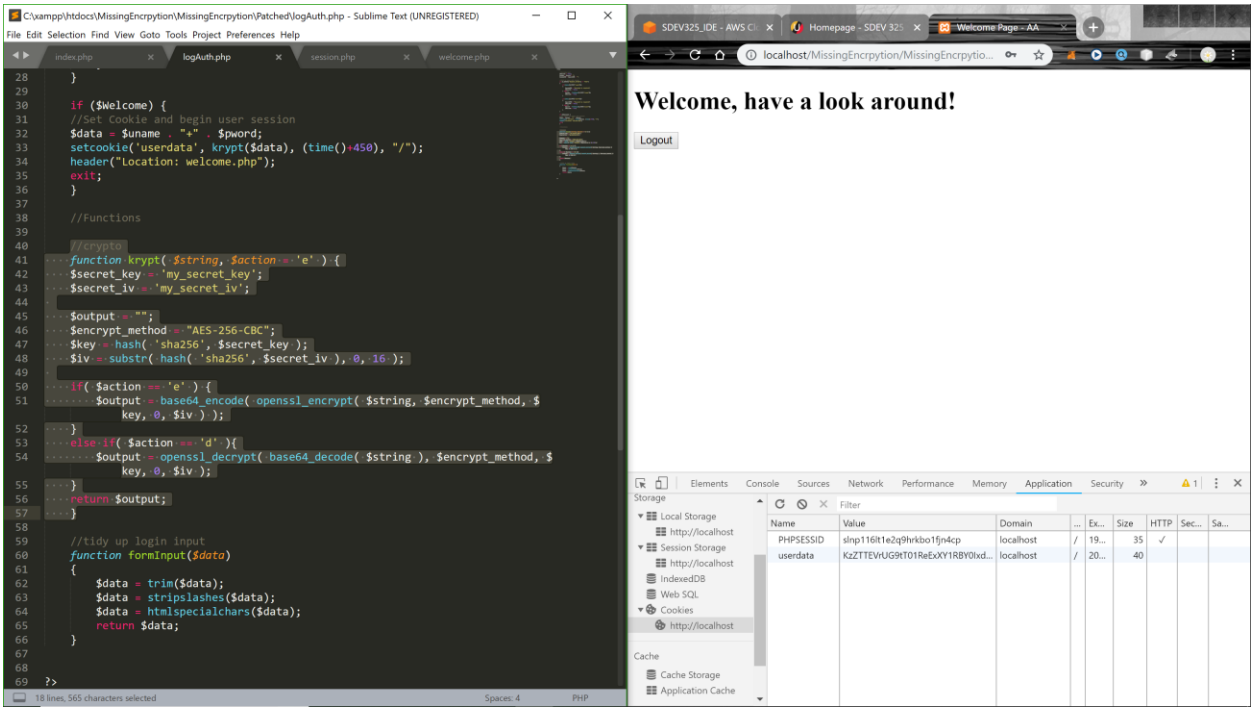



*Figure 6. Demonstrating Porous Defenses. This figure shows the default page for the web application (right) and its underlying authentication code (left).*

*Figure 6*, the code on the left is still **logAuth.php** but the notice the userdata cookie and is value after login in to the patched application. The value is still the username+password from before but now its encrypted and wrapped in a base64 representation. This should offer better protection against compromised machines and cross-site scripting. This mitigation was also one of the solutions recommended by OWASP for CWE-311.

# Use of Hard-coded Credentials: CWE-798

## Description

The software contains hard-coded credentials, such as a password or cryptographic key, which it uses for its own inbound authentication, outbound communication to external components, or encryption of internal data.

Resource - https://cwe.mitre.org/data/definitions/798.html

## Vulnerable Application

The application will accept a user supplied username and password pair which will be compared against hard-coded credentials. Anyone that has access to the code will also have access to the password, which is a huge security vulnerability. The server could mistakenly divulge credentials if it's Access Control is not properly configured.

## Demonstration

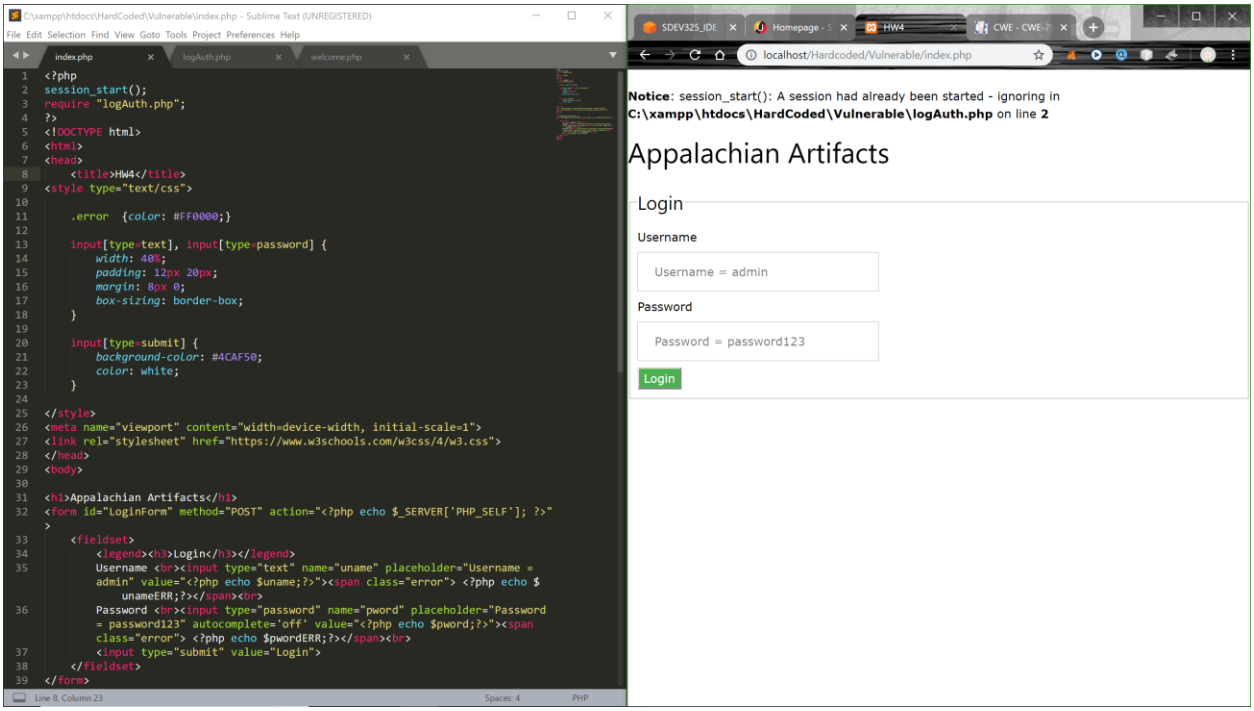



*Figure 7. Demonstrating Porous Defenses. This figure shows the default page for the web application (right) and its source code (left).*

In *Figure 7*, the code (left) shows the basic structure of a login form which includes CSS styling. The form submission is handled by **logAuth.php**.

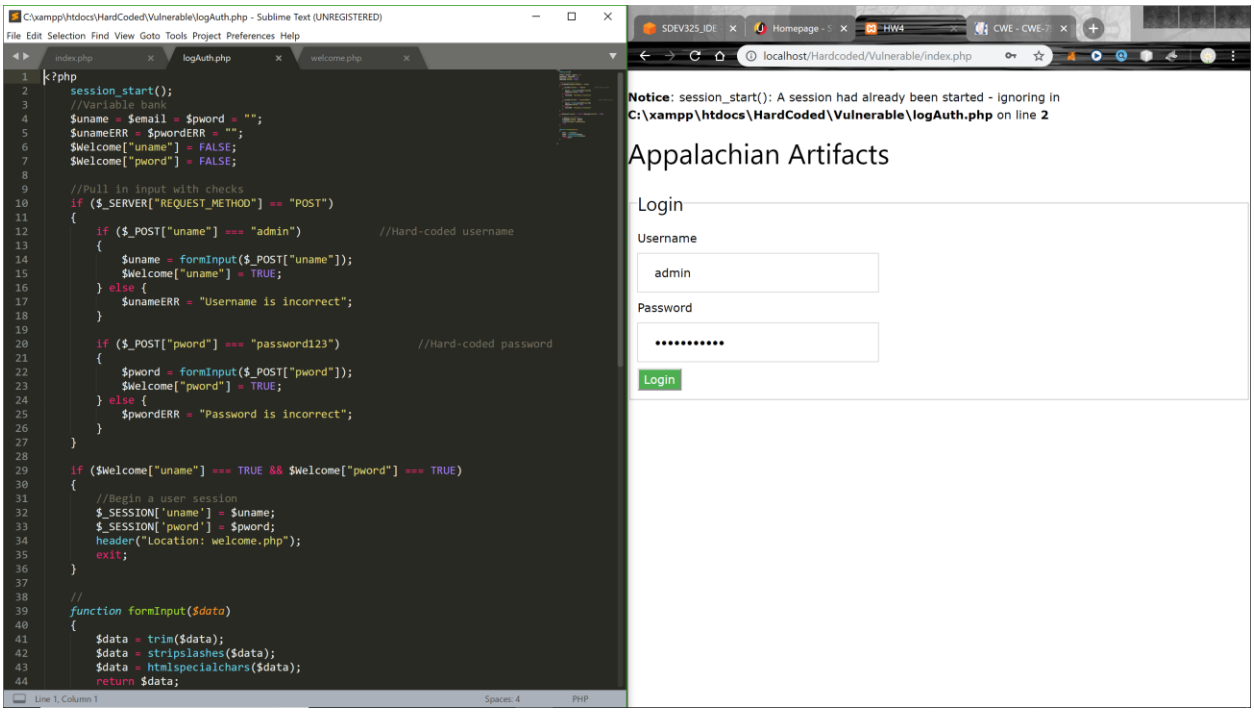



*Figure 8. Demonstrating Porous Defenses. This figure shows the default page for the web application (right) and its underlying authentication code (left).*

Lines 12 and 20 contain the hardcoded username and password respectively, which have been entered in the application on the right-side *Figure 8*. Essentially, the code checks the user supplied variables against the hard-coded strings and if they match the user is sent to the admin welcome page. If not they are asked to authenticate again – try getting it wrong once.

OWASP states "If hard-coded passwords are used it is, it is almost certain that malicious users will gain access through the account in question." The likelihood of exploit is very high. Negative implications of such practice can include: failure to authenticate under certain circumstances, default user creds are trivial to look up, it is exceedingly simple to extract strings from binary on client-side, back-end services with drop-in solutions are easily discoverable.

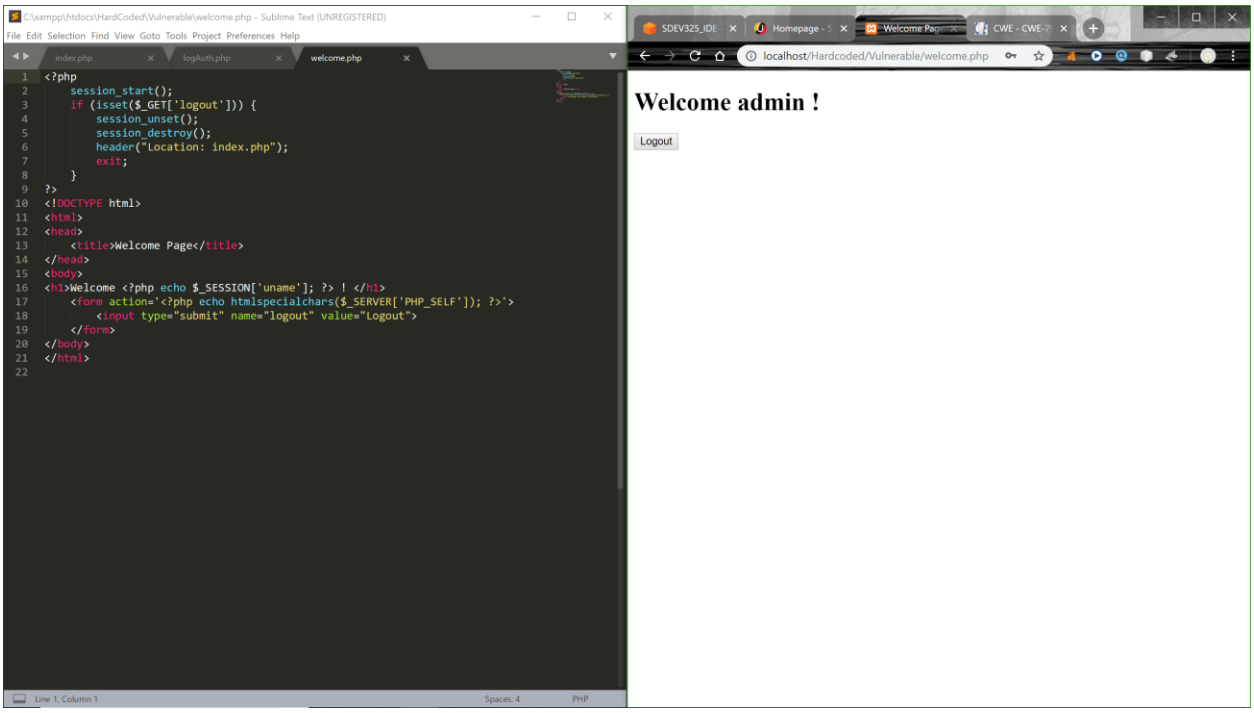



*Figure 9. Demonstrating Porous Defenses. This figure shows the landing page for the web application (right) and its source code (left).*

Authentication works and the user is greeted by the page and given the option to logout since this is just a simple application. In the real world, this page might provide administrative access to a database or other sensitive resources (see *Figure 9*).

## Patched Application

In this section, we will implement Access Control on the authentication credentials which will be isolated from the application and server. There are only some of the best practices recommended by OWASP.

## Demonstration

The first noticeable difference in the patched version is the addition of **adminCreds.php** which is located outside the web server root directory. This will isolate it from the application and allow use to restrict Access Control on the file to only the application. chmod 600 adminCreds.php will also allow only the owner of the file to read and execute it, thus providing greater security.

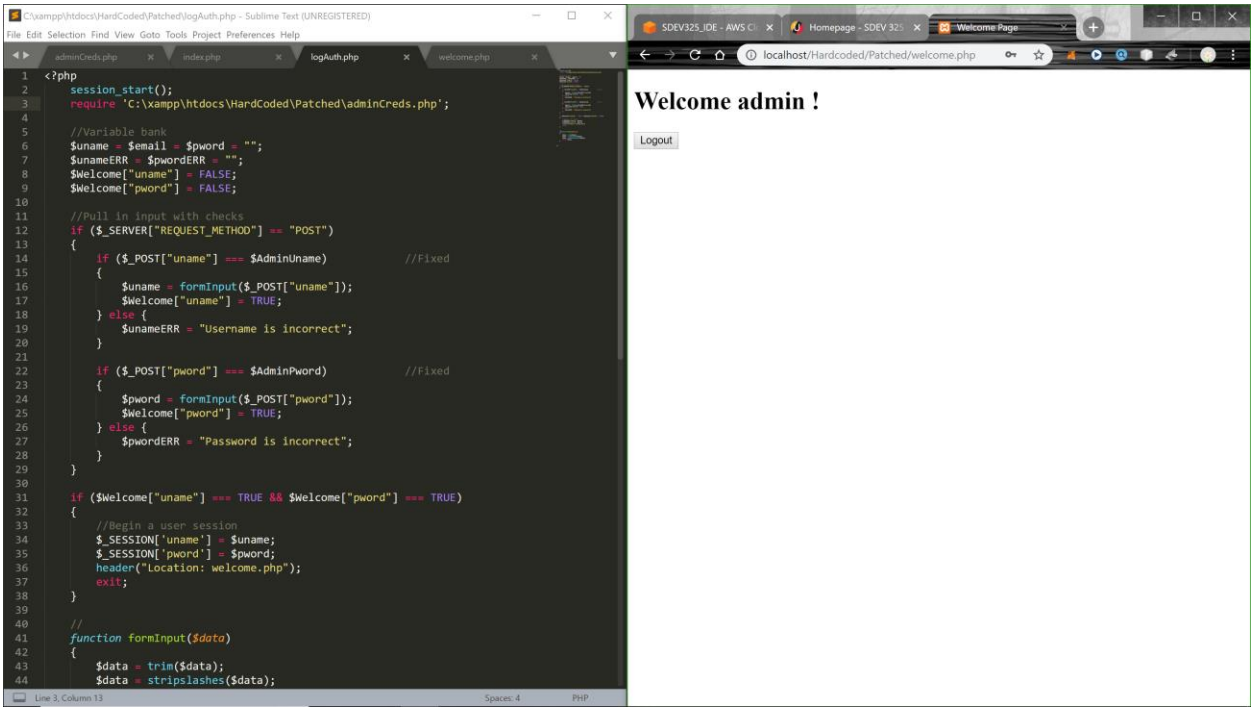



*Figure 10. Demonstrating Porous Defenses. This figure shows the landing page for the web application (right) and its authentication code (left).*

In *Figure 11*, the code shows that the previously hard-coded credentials are now appropriately names variables (lines 14 and 22). This allows for security through obfuscation due to the code not having direct access to the values, which come from the require statement on line 3. require 'C:\xampp\htdocs\HardCoded\Patched\adminCreds.php' tells the script to look for the file in the directory above the current one, which is root. By now you might be wondering what's in the **adminCreds.php** file?

The following is the contents of **adminCreds.php**:

```
 <?php
$AdminUname = "admin";
$AdminPword = "password123";
```

```
?>
```

As you can see it’s just the credentials set to their respective variables from the **logAuth.php** conditional evaluation of the user input.  The security here comes from proper server and host configuration.