Secure Programming CW2

1.
Step 1: Input malicious image code into the user field, example:
<img src="url" onError="alert('executed')">
Step 2: Enter a password and register the user
Step 3: Enter a .jpg image in the image link, for example,
https://images.squarespace-cdn.com/content/v1/56acc1138a65e2a286012c54/1476623632079-BBAERA9UGQ0EODC6680U/pixabaytest6-7.jpg
Step 4: Press submit/update
Result: Whenever a user logs into the website, it causes an alert since the program tried to show the image as the user name of the candidate who had put in a malicious username, but the src is not a valid source.

Patch: Limit the username to alphanumeric characters. I have limited to alphanumeric since the sample database does not have any usernames containing any characters other than alphabets and digits, and this prevents any metacharacters being put into the field like “,’, < and >.

2.
Attacker:
Step 1: Create a user and add an image for the user
Step 2: Create another user and input malicious code into the signature field as you register (<img src="http://localhost:8080/vote.php?vote=david">) - This exploit is for David, and can be replaced by any other username. Add an image for this user

Victim:
Step 1: Victim logins in without realising they would vote for the user created in the attacker’s step 1. (The vote would go to David if the malicious code in the attacker’s step 2 is used since it is an example.)

Patch: Strips all the PHP and HTML tags, so no image can be tried to be rendered through the signature and prevents the specified attack.

3.

4.

For each vulnerability, describe its source, how it might be exploited and, by giving a *brief* correction to the code, how it may be fixed.

a) Session Token Hijacking: The source of the vulnerability is the setcookie function, which creates a unique session token by encrypting the username with a md5 encryption algorithm, According to OWASP, it is a weak encryption and is not suggested for use.

Additionally, the session token should be unpredictable but here it uses the username, which could be public information and weak encryption. Therefore the attacker could guess the token using the username and impersonate the user. [Correction: Using SHA256 -> One of the most secure hashing algorithms currently]

b) **Plaintext storage of a password** : When the user registers, the password is not being hashed and is stored as plaintext in the database. In the case that the database is stolen, all the passwords linked to the usernames would be known and therefore the users' identities can be stolen. [Correction: Hashing the password with SHA256 (One of the most secure hashing algorithms currently) so that even if the database gets stolen, the adversary does not know what the password for any of the user's is, except username which most times is public information]

c) **SQL Injection:** If any function fails, the user gets to know the response that was sent back from the database. Issue with registration: (SQLSTATE[23000]: Integrity constraint violation: 19 UNIQUE constraint failed: users.username) The adversary could get these messages for various invalid inputs and learn information about the database. Using this information, can launch an sql injection attack. [Correction: Commenting out the line which prints the error and replacing that by writing a error message which reveals basic information but not too much as is being done now according to error number. Have just commented out the line for now and put a generic message for each function, since it takes care of the vulnerability. The error message part is to provide better functionality.]