Student Name: Nathan Crowley
Student Number:118429092
Date: 15/12/2020.

Question 1: *An application requires a password to log in. The password policy is:*
1) Password must be 6-8 characters in length.
*2) Passwords must use alphabetic **and** numeric characters.*
3) Passwords must have at least one alphabetic and one numeric character.
4) Letters are case sensitive.
How many different passwords exist? Describe how you got your result.

Answer 1: If we take three separate passwords at length 8 and length 7 and length 6. Each position in the password has 62 possibilities, (0-9 = 10 possibilities ,a-z = 26 possibilities, A-Z = 26 possibilities, 10+26+26 = 62. But there must be at least one position with a numeric and one with an alphabetic, so if i set the first position as the numeric(10) and the second as alphabetic(52 = a-z+A-Z):

Length=8 -> numeric(10) * alphabetic(52) * 62 * 62 * 62 * 62 * 62 * 62
+
Length=7 -> numeric(10) * alphabetic(52) * 62 * 62 * 62 * 62 * 62
+
Length=6 -> numeric(10) * alphabetic(52) * 62 * 62 * 62 * 62

To get all possibilities I added the three passwords together to get my final answer.
Final answer = 3.002019527e13 or 30,020,195,270,000 possible passwords.

Question 2: How many parts has a JSON Web Token? How are these parts encoded? Describe the function of each part. Highlight the different parts of the JST shown below..
JWT =
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiw bmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2Q T4fwpMeJf36POk6yJV_adQssw5c

Answer 2:

- JSON Web Token consists of three parts, separated by a full stop ".":
  1) **Header** - *eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9*
  2) **Payload** -*eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ*
  3) **Signature** -*SflKxwRJSMeKKF2Q T4fwpMeJf36POk6yJV_adQssw5c*
     (JWT typically looks like: xxxx.yyyy.zzzz)

- These three parts are each encoded with **Base64Url**.
- Function of each part:
  1) **Header** = typically consists of two parts: **type of token** and the **signing algorithm** being used.(eg. RSA or HMAC SHA256)
  2) **Payload** = this contains the **claims**.Claims are statements about an entity(typically the user) and additional data. There are three types of claims: *registered, public, private*.

3) **Signature** = to create the signature part you have to take the **encoded header, the encoded payload, a secret, the algorithm** in the header**,** and **sign** that**.** The signature is used to **verify the message** hasn't been changed while being transmitted, as well as **verifying the sender** of the JWT is who they say they are.

Question 3: Explain how the use of Prepared Statements prevents SQL injection attacks. Please give a commented code example, describing the difference between database access with and without the use of Prepared Statements.

Answer 3:

- Prepared Statement = parameterized and reusable SQL query which forces the developer to write the SQL command and the user-provided data separately. The SQL command is executed safely, preventing SQL injection vulnerabilities.
- **Unsafe query in PHP:**

```
$query = “SELECT * FROM people WHERE username = ‘$username’
and password = ‘$password’”;
$result = mysql_query($query);
```

  User-provided data is embedded directly into the SQl query so if the user enters “username = admin” and “password = a’ or ‘1’=’1”. They will be given access as 1=1 will always be true.
- **Prepared Statement:**
  To prevent this we must keep data separate from commands and queries.
  PHP prepared statement:

```
$stmt = $mysqli->prepare(“SELECT* FROM people WHERE
username = ? AND password = ?”);
$stmt->bind_param(“ss”,$username,$password);
$stmt->execute();
```

  The users data (username/password) not directly embedded in the SQL. Instead I have a placeholder “?”. This helps as the SQL query is *pre-compiled* with the placeholders and the users data is added later. So if the user tries to insert “username = admin” and “password = a’ or ‘1’=’1”, the initial pre-compiled SQL query logic won't be altered. The database will look for a user “admin” whose password is literally “a’ or ‘1’=’1”. Which will not return anything.

Question 4: Explain the difference between a **stored** Cross-Site Scripting attack and a **reflective** Cross-Site Scripting attack.

Answer 4:

- Cross-Site Scripting (XSS) = A common attack vector that injects malicious code into a vulnerable web application. It differs from other web attacks as it does not directly target the application but instead targets the user.
- **Stored Cross-Site Scripting:** generally occurs when the user input is stored on the target server (database,comment field). The target is then able to retrieve the stored data from the web app without that data being made safe to render in the browser. This is more damaging than *Reflective XSS*.

    - **Reflective Cross-Site Scripting:** occurs when a malicious script is reflected off a web application to the victims browser. The script is activated through a link. The vulnerability is a result of incoming requests not being sufficiently sanitized, this allows for the manipulation of web app functions to activate malicious scripts. The link is normally embedded in an email or third party website, the link is inside an **anchor text** that tries to make the user click on it.

Question 5: How are Cross-Site Requests Forgeries (CSRF) prevented? Describe a prevention using an example.
Answer 5:

- CSRF attacks = take advantage of the trust a website has for a users input and browser. The victim is tricked into performing a specific action they were not intending to do on a legitimate website. CSRF attacks will use the identity and privileges that the victim has on the website to impersonate them and perform malicious activity. Attacks will attempt to take advantage of users who have **login cookies** stored in their browsers.
- To prevent CSRF attacks you can make sure that the request you are receiving is **valid**:
    - Use a **CSRF token**, to prove that you're sending a request from a form or link generated by the server. When the server receives the request from this form, it compares the received token value with previously generated value. If they match it the request is valid.