The three proposed codes differ in terms of security level and present various vulnerabilities that can be exploited by potential attackers. Below, I highlight the differences in security level among the three codes and their respective vulnerabilities and solutions:

Code 1: Security Level - Low

Vulnerabilities:

1. SQL Injection: The code directly incorporates user input values into the SQL query without any sanitization or control, making the application vulnerable to SQL injection attacks.

2. Use of MD5 Hashing: Hashing passwords using the MD5 algorithm is considered weak and vulnerable to rainbow table hash attacks.

3. Lack of control on the number of allowed access attempts.

Solutions:

1. Use of Query Parameterization or ORM: Use parameterized SQL statements or an Object-Relational Mapping (ORM) to avoid SQL injection attacks.

2. Use of Secure Hashing Algorithms: Replace MD5 hashing with more secure hash algorithms like bcrypt or Argon2 to protect passwords from rainbow table hash attacks.

3. Implement an account locking mechanism after a certain number of failed attempts.

Code 2: Security Level - Medium

Vulnerabilities:

1. Use of Deprecated Functions: It uses the `mysql_real_escape_string` function, which is deprecated and does not provide complete protection against SQL injection attacks.

2. Use of MD5 Hashing: It still uses MD5 hashing to protect user passwords, which is vulnerable to rainbow table hash attacks.

Solutions:

1. Use of Modern Sanitization Functions: Use modern sanitization functions like `mysqli_real_escape_string` or query parameterization to prevent SQL injection attacks.

2. Use of Secure Hashing Algorithms: Replace MD5 hashing with more secure hash algorithms like bcrypt or Argon2 to protect passwords from rainbow table hash attacks.

Code 3: Security Level - Medium-High

Vulnerabilities:

1. Use of Deprecated Functions: It still uses the `mysql_real_escape_string` function, which is deprecated and does not provide complete protection against SQL injection attacks.

2. Use of MD5 Hashing: It continues to use MD5 hashing to protect user passwords, which is vulnerable to rainbow table hash attacks.

3. Sleep Time for Failed Login: It adds a 3-second delay in case of failed login, which could be exploited to execute brute force attacks.

Solutions:

1. Use of Modern Sanitization Functions: Use modern sanitization functions like `mysqli_real_escape_string` or query parameterization to prevent SQL injection attacks.

2. Use of Secure Hashing Algorithms: Replace MD5 hashing with more secure hash algorithms like bcrypt or Argon2 to protect passwords from rainbow table hash attacks.

3. Implementation of Anti-Brute Force Controls: Implement mechanisms to detect and mitigate brute force attacks, such as login attempt limits, captchas, or increasing delays.

General Note on Brute Force Attack and Vulnerabilities in the Three Scripts:

A brute force attack is a technique used by attackers to gain unauthorized access to a system or application by repeatedly trying different combinations of usernames and passwords until the correct ones are found. It is important to understand and mitigate the vulnerabilities that make this type of attack possible.

In general, it is crucial to keep systems and applications constantly updated, use secure development practices, and implement robust security measures to prevent brute force attacks and protect sensitive user data.

In addition to these technical notes, it is necessary and fundamental to adopt training and a process that educates employees about taking care of their passwords in a path that goes from choosing to regularly changing them. Employees should also be explained the importance of choosing a long and complex passphrase.

However, it is emphasized the importance of seeking a balance between security and usability.