Student's Name

Department, Institutional Affiliation

Course Name and Number

Professor's Name

Due Date

Password Hashing Techniques for Secure Password Storage in PHP

Introduction to Password Hashing

Nowadays, the security of user data is paramount, especially when it involves sensitive information such as passwords. Passwords act as the first line of defense against unauthorized access to user accounts; therefore, ensuring their protection is crucial. Hashing is a fundamental security technique used to safeguard passwords, transforming the original password into a different string of characters. In PHP, a server-side scripting language widely used for web development, robust password hashing techniques are vital (Ribeiro 2023). As a result, user accounts will remain secure even if intruders gain access to the hashed password.

The Importance of Password Hashing

Password hashing is a one-way process of converting a password into another string, known as a hash, which is designed to be irreversible. The primary purpose of password hashing is to protect the password's integrity even if the data storage is compromised (Kinde 2023). Without hashing, stored passwords can be easily retrieved and exploited by attackers.

Hashing differs from encryption in that encryption is a two-way process that allows for data to be converted back to its original form through decryption. Hashes, however, are meant to be unique, as even a small change in the password should result in a significantly different hash

(Zelinsky 2022). This property is crucial in preventing attackers from guessing the original password even if they have access to the hash.

PHP's Built-In Functions for Password Hashing

PHP offers a comprehensive suite of functions for password hashing and verification. The most notable of these functions are password_hash() and password_verify(). PHP also includes functions to handle other aspects of password security, such as password_needs_rehash() to check if the password hash needs to be rehashed to a newer algorithm. As a result, it makes developers hash passwords with ease.

The "password_hash()" Function

The "password_hash()" function is the primary function used to create a new password hash using a strong, one-way hashing algorithm. PHP's "password_hash()" function uses the bcrypt algorithm by default, which is currently considered a robust option (Bondar 2023). The function syntax is as follows:

```
string password_hash ( string $password , int $algo [, array $options ] )
```

In the above code, $password is the user's plain text password. $algo is the password algorithm constant that you want to use. $options is an associative array of options. It can contain a cost that defines the algorithmic cost that should be used.

The screenshot below shows an example usage of the function.

```php
$options = [
    'cost' => 12,
];
$hash = password_hash("user_password", PASSWORD_DEFAULT, $options);
```

The "password_verify() "Function

To verify that a given password matches a certain hash, PHP provides the password_verify() function. It checks if the provided hash matches the hash of the given password (Nielsen 2020). It is used during the login process to validate user passwords. The screenshot below shows an example usage of the "password_verify()" function.

```php
if (password_verify('user_password', $existingHash)) {
    // Password is correct
} else {
    // Invalid password
}
```

Best Practices for Secure Password Storage

When storing passwords, several of the best practices should be adhered to in order to ensure optimal security:

Use a Strong Hashing Algorithm

Always use a strong and well-tested hashing algorithm. The PASSWORD_DEFAULT in PHP uses bcrypt, which is a good choice due to its computational intensity and resistance to brute-force attacks.

Implement Salts Correctly

Salting is a technique used to safeguard against dictionary attacks and the use of rainbow tables. PHP's password_hash() automatically generates a secure salt if one is not provided (Sreenidhi 2020). Avoid using custom salting methods, as they can be less secure than the built-in functionality.

Ensure a Sufficient Cost Factor

The cost factor is an essential aspect of hashing security. It determines how much time is needed to calculate a single hash. The higher the cost, the more secure the hash, but the more processing power is required. As hardware gets faster, the cost factor should be reviewed and increased if necessary.

Regularly Update Hashing Strategies

Security is a moving target and what is considered secure today may not be tomorrow. Regularly review and update the hashing strategy to adhere to the best current practices. This includes rehashing passwords with newer algorithms as they become available.

Protect Against Breaches

Apart from hashing, make sure to implement other security measures like HTTPS, secure cookies, rate limiting login attempts, and more, to protect against different types of attacks.

Conclusion

Hashing passwords is an essential component of secure password storage. In PHP, using the “password_hash()” and “password_verify()” functions allows for the implementation of strong and reliable security measures. Adhering to best practices like using up-to-date and robust algorithms, proper salting, adjusting cost factors, and maintaining proactive security protocols helps ensure that passwords are stored securely and efficiently.

References

Bondar, Sasha. "PHP and Password Hashing: Securely Storing and Verifying Passwords." *Software Developers As a Service | Reintech*, 28 Apr. 2023, reintech.io/blog/php-password-hashing-securely-storing-verifying-passwords.

Kinde. "Password Hashing and Why It's Important." *Kinde Guides*, 25 Oct. 2023, kinde.com/guides/authentication/passwords/password-hashing-salting/.

Nielsen, Chris. "PHP: How to Use Password_hash and Password_verify." *Chris Nielsen Code Walk*, 25 Sept. 2020, bluegalaxy.info/codewalk/2018/11/22/php-how-to-use-password_hash-and-password_verify/.

Ribeiro, Claudio. "Quick Tip: How to Hash a Password in PHP — SitePoint." *SitePoint – Learn HTML, CSS, JavaScript, PHP, UX & Responsive Design*, 14 Feb. 2023, www.sitepoint.com/quick-tip-how-to-hash-a-password-in-php/.

Sreenidhi. "PHP Salts & Password Hashing - All You Need To Know." *Astra Security Blog*, 9 Oct. 2020, www.getastra.com/blog/php-security/php-salts-and-hashes/.

Zelinsky, Alex. "Web Security: The Importance of Hashing and Salting Passwords." *Medium*, 18 July 2022, levelup.gitconnected.com/web-security-the-importance-of-hashing-and-salting-passwords-7582f36f9d0e.