

Quiz Questions

(1) Salt in Encryption

← Classroom

Password Encoding & Security Practices
Salt in Encryption

?

V

Problem Submissions Doubts

✓ Salt in Encryption

Easy • Score 20/20

Problem statement

What purpose does salt serve in encryption?

Send feedback

Options: Pick one correct answer from below

☒ It adds complexity and randomness to password hashing

☐ It reduces the size of the encrypted data.

☐ It increases the decryption speed.

☐ It simplifies the encryption process.

Solution description

Salting involves adding a random value (salt) to each password before hashing it. This adds complexity, making it significantly more difficult for attackers to use precomputed tables (rainbow tables) to crack passwords.

(2) Iterations in Hashing

← Classroom

Password Encoding & Security Practices
Iterations in Hashing

?

V

Problem Submissions Doubts

✓ Iterations in Hashing

Easy • Score 20/20

Problem statement

How does increasing the number of iterations in password hashing affect security?

Send feedback

Options: Pick one correct answer from below

☐ It decreases the time required to hash the password

☐ It does not affect security

☒ It increases security

☐ It slows down the encryption process without affecting security

Solution description

Increasing the number of iterations in password hashing strengthens security by making it more time-consuming and resource-intensive for attackers attempting to crack passwords through brute-force or dictionary attacks.

(3) Authentication Mechanism

← Classroom

Password Encoding & Security Practices
Authentication Mechanism

?

V

Problem Submissions Doubts

✓ Authentication mechanism

Easy • Score 20/20

Problem statement

Which method is commonly used for password authentication?

Send feedback

Options: Pick one correct answer from below

☐ Decoding the stored password

☐ Encoding the upcoming password

☐ Encrypting the stored password

☒ Hashing the upcoming password

Solution description

Hashing involves using a one-way function to convert the password into a fixed-length string of characters. A good hashing algorithm takes the password and produces a unique hash value. It's computationally infeasible to reverse a hash to retrieve the original password. During authentication, the hash of the entered password is compared to the stored hash, allowing access if they match.

(4) Todo Application

← Classroom

Password Encoding & Security Practices
Todo Application

Problem Submissions Doubts

Todo Application

Easy • Score 20/20

Send feedback

Problem statement

Suppose you are making a secured todo Application where the user password is encoded using the Pbkdf2PasswordEncoder. Choose the correct implementation of password encoding.

Options: One or more answers may be correct

☒

```
***
//Security Config class
public class TodoSecurityConfig {
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new Pbkdf2PasswordEncoder();
    }
}

// User Service class
public class UserService {

    @Autowired
    private PasswordEncoder passwordEncoder;

    public void createNewUser(UserRequest userRequest) {
        String encodedPassword = passwordEncoder.encode(userRequest.getPassword());
        User user = new User();
        user.setUsername(userRequest.getUsername());
        user.setPassword(encodedPassword);
        userRepository.save(user);
    }
}
```

☒

```
***
//Security Config class
public class TodoSecurityConfig {
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new Pbkdf2PasswordEncoder();
    }
}

// User Service class
public class UserService {
    PasswordEncoder passwordEncoder = new Pbkdf2PasswordEncoder();

    public void createNewUser(UserRequest userRequest) {
        String encodedPassword = passwordEncoder.encode(userRequest.getPassword());
        User user = new User();
        user.setUsername(userRequest.getUsername());
        user.setPassword(encodedPassword);
        userRepository.save(user);
    }
}
```

☐

```
***
//Security Config class
public class TodoSecurityConfig {
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new Pbkdf2PasswordEncoder();
    }
}

// User Service class
public class UserService {

    @Autowired
    private PasswordEncoder passwordEncoder;

    public void createNewUser(UserRequest userRequest) {
        User user = new User();
        user.setUsername(userRequest.getUsername());
        user.setPassword(userRequest.getPassword());
        userRepository.save(user);
    }
}
```

☐ None of the above

(5) CSRF Attack

← Classroom

Password Encoding & Security Practices
CSRF Attack

Problem Submissions Doubts

CSRF Attack

Easy • Score 20/20

Send feedback

Problem statement

Which of the following best describes a CSRF attack?

Options: Pick one correct answer from below

☐ Exploiting vulnerabilities in the browser's security settings

☐ Forging a user's identity to gain unauthorised access

☐ Manipulating data within a website's database

☒ Trickery that executes unwanted actions on a different website while the user is authenticated on that site

Solution description

Option D accurately defines CSRF: it involves maliciously inducing a user who is authenticated on a particular website to perform unintended actions on another site where the user is also authenticated without their explicit consent.

(6) XSS Attack

← Classroom

Password Encoding & Security Practices

XSS Attack

Problem Submissions Doubts

XSS Attack

Easy • Score 20/20

Problem statement

What is the primary objective of a Cross-Site Scripting (XSS) attack?

Send feedback

Options: Pick one correct answer from below

☐ Directly gaining access to the server's database

☒ Executing malicious scripts within a user's browser

☐ Intercepting communication between client and server

☐ Modifying the website's HTML structure

Solution description

XSS attack involves injecting and executing malicious scripts within a user's browser, often leading to theft of sensitive information, session hijacking, or performing unauthorised actions on behalf of the user.

(7) Todo Application II

← Classroom

Password Encoding & Security Practices

Todo Application II

Problem Submissions Doubts

Todo Application II

Easy • Score 20/20

Problem statement

Continuing the todo application, we have a TodoHistory Service, which records the past todos we have completed. Now we need to make a restTemplate call to TodoHistory Service to fetch the past Todos. The History Service runs on port 8081, has JWT authentication, and the API is: GET "http://localhost/history/getAllTodos/{userId}": It returns a list of todos, i.e. List.

Send feedback

☒

```
import java.util.List;
import org.springframework.http.ResponseEntity;
import org.springframework.web.client.RestTemplate;

public class TodoHistoryCommunicator {

    private final RestTemplate restTemplate;

    public TodoHistoryCommunicator(RestTemplateBuilder restTemplateBuilder) {
        this.restTemplate = restTemplateBuilder.build();
    }

    public List<Todo> getAllTodosFromHistory(String jwtToken, Long userId) {
        String historyUrl = "http://localhost:8081/history/getAllTodos";
        HttpHeaders headers = new HttpHeaders();
        headers.set("Authorization", "Bearer " + jwtToken);
        HttpEntity<> requestEntity = new HttpEntity<>(headers);
        ResponseEntity<List<Todo>> responseEntity = restTemplate.exchange(
            historyUrl, HttpMethod.GET,
            requestEntity,
            new ParameterizedTypeReference<>());
        if (responseEntity.getStatusCode().is2xxSuccessful()) {
            return responseEntity.getBody();
        } else {
            return Collections.emptyList();
        }
    }
}
```

☐ None of the above

Solution description

Method Signature: The method getAllTodosFromHistory takes two parameters: jwtToken (JWT token as a String) and userId (User ID as a Long). HTTP Headers and JWT: Creates an HttpHeaders object and sets the JWT token in the Authorization header as "Bearer + jwtToken". Request Entity: Builds an HttpEntity object (requestEntity) using the created headers. This entity encapsulates the headers and will be used in the request. REST Call: Uses restTemplate.exchange() to request HTTP GET to fetch Todos for a specific user. It appends the user ID to the URL to specify the user for whom the Todos are being fetched. Handling Response: Checks the response status code. If the response is successful (status code in the 2xx range), it returns the list of Todos obtained from the response body. Otherwise, it returns an empty list.

Options: Pick one correct answer from below

☒

```
import java.util.List;
import org.springframework.http.ResponseEntity;
import org.springframework.web.client.RestTemplate;

public class TodoHistoryCommunicator {

    private final RestTemplate restTemplate;

    public TodoHistoryCommunicator(RestTemplateBuilder restTemplateBuilder) {
        this.restTemplate = restTemplateBuilder.build();
    }

    public List<Todo> getAllTodosFromHistory(String jwtToken, Long userId) {
        String historyUrl = "http://localhost:8081/history/getAllTodos";
        HttpHeaders headers = new HttpHeaders();
        headers.set("Authorization", "Bearer " + jwtToken);
        HttpEntity<> requestEntity = new HttpEntity<>(headers);
        ResponseEntity<List<Todo>> responseEntity = restTemplate.exchange(
            historyUrl, HttpMethod.GET,
            requestEntity,
            new ParameterizedTypeReference<>());
        if (responseEntity.getStatusCode().is2xxSuccessful()) {
            return responseEntity.getBody();
        } else {
            return Collections.emptyList();
        }
    }
}
```

☐

```
import java.util.List;
import org.springframework.http.ResponseEntity;
import org.springframework.web.client.RestTemplate;

public class TodoHistoryCommunicator {

    private final RestTemplate restTemplate;

    public TodoHistoryCommunicator(RestTemplateBuilder restTemplateBuilder) {
        this.restTemplate = restTemplateBuilder.build();
    }

    public List<Todo> getAllTodosFromHistory(String jwtToken, Long userId) {
        String historyUrl = "http://localhost:8081/history/getAllTodos";
        HttpHeaders headers = new HttpHeaders();
        headers.set("Authorization", "Bearer " + jwtToken);
        HttpEntity<> requestEntity = new HttpEntity<>(headers);
        ResponseEntity<List<Todo>> responseEntity = restTemplate.exchange(
            historyUrl, HttpMethod.GET,
            requestEntity,
            new ParameterizedTypeReference<>());
        if (responseEntity.getStatusCode().is2xxSuccessful()) {
            return responseEntity.getBody();
        } else {
            return Collections.emptyList();
        }
    }
}
```