

5.5 Protecting Applications

Name: _____ Class: _____ Date: _____

Total: 12 marks

KEY WORDS input sanitization • secure by design
• secure by default • sql injection SQL • directory traversal

Objective

Explain **secure by design** and choose application defenses that keep untrusted input from becoming instructions.

You must be able to:

- distinguish safe defaults from a guarantee of perfect security
- preserve legitimate data while validating it for its purpose
- match SQL values, browser output and file paths to their own protections

1 Key knowledge and worked examples

■ Design principles

Secure by design builds security into every development phase. Secure by default enables protective settings from the start; updates and careful configuration can still be needed.

■ Match the destination

Input sanitization transforms input for a particular use; it is not one universal character filter. A name such as O'Neill contains a legitimate apostrophe. For SQL, use a fixed statement such as `SELECT name FROM pupils WHERE name = ?` and bind the name separately. For plain-text browser output, use a safe text API or context-specific encoding. When HTML is allowed, use a suitable HTML sanitizer. For a file request, resolve the path and require the final location to stay inside the permitted directory.

■ Why removing punctuation is insufficient

A numeric query built by joining input to SQL can be altered by `0 OR 1=1`, which contains neither quotes nor semicolons. Bind values separately and validate the required number and range. Do not damage legitimate input while leaving the actual syntax boundary unprotected.

2 Practice

2.1 Explain what 'secure by default' means, including one limit to that claim. [2]

2.2 Give a suitable control for each destination: a value in an SQL query; a plain-text comment on a web page; a user-requested file path. [3]

2.3 A developer deletes all apostrophes from names. Explain one usability problem and why this is not a complete SQL-injection defense. [2]

3 Exam-style questions

3.1 Which control keeps an entered name as data rather than SQL syntax? [1]

- ☐ **A** deleting only semicolons
B binding the value separately in a fixed query
C encoding the name as HTML
D accepting every value shorter than 20 characters

3.2 'Secure by design' means security is [1]

- ☐ **A** added only after testing finishes
B considered throughout development
C delegated entirely to each user
D guaranteed by deleting punctuation

3.3 A form joins an age input directly into `SELECT name FROM pupils WHERE age =` followed by the entered text. An adversary enters `0 OR 1=1`.

(a) State the attack risk. [1]

(b) Explain how to keep that input from changing the SQL structure while still checking that a permitted age was entered. [2]

Solutions

2.1 Protective settings are enabled from the start. This does not guarantee no vulnerabilities or remove the need for updates.

2.2 SQL: a fixed query with separately bound values. Browser: safe text rendering or context-specific output encoding. File path: resolve the path and enforce an allowed directory boundary.

2.3 It damages a legitimate name such as O'Neill. An attack on an unsafe numeric SQL expression can need no apostrophe; binding values separately is required instead of relying on that blacklist.

3.1 B. Binding keeps the value separate from query syntax; HTML encoding addresses a different destination.

3.2 B. Security is part of design, implementation and review rather than a final addition.

3.3

(a) SQL injection: the input may alter the query condition and expose unintended rows.

(b) Use a fixed query with a bound value instead of joining input into SQL. Validate that the value is a number in the permitted age range, rejecting this expression as invalid input.