**Họ và tên : Nguyễn Thị Thùy Linh**

**MSV:22810310159**

**Lớp: D17CNPM3**

**Bài kiểm tra số 01**

**PHẦN 1:**

**Validate input & sanitize output**

1. Nguyên tắc quan trọng nhất khi kiểm tra dữ liệu đầu vào là gì?  
   A. Luôn tin tưởng dữ liệu từ client-side  
   B. Chỉ cần kiểm tra dữ liệu ở phía frontend  
   C. Luôn validate dữ liệu ở phía server-side  
   D. Không cần validate nếu người dùng đáng tin
2. Khi xử lý đầu ra để tránh XSS, cần phải:  
   A. Escape dữ liệu theo ngữ cảnh sử dụng  
   B. Escape toàn bộ dữ liệu thành plain text  
   C. Không cần escape nếu dữ liệu từ DB  
   D. Chỉ escape khi dữ liệu chứa ký tự < hoặc >
3. Cách tiếp cận nào an toàn hơn khi validate input?  
   A. Blacklist (chặn những ký tự nguy hiểm)  
   B. Whitelist (chỉ cho phép ký tự hợp lệ)  
   C. Không validate để tránh phức tạp  
   D. Tự động sửa dữ liệu sai thành hợp lệ
4. Kỹ thuật nào giúp chống Unicode trick (ví dụ: ký tự đồng hình)?  
   A. Minify dữ liệu đầu vào  
   B. Chuẩn hóa (canonicalization/normalization)  
   C. Tự động escape đầu ra  
   D. Loại bỏ tất cả ký tự Unicode
5. Khi xuất dữ liệu lên HTML attribute, cách xử lý đúng là:  
   A. Chèn trực tiếp chuỗi từ input  
   B. Escape dữ liệu theo chuẩn HTML attribute encoding  
   C. Encode base64 mọi dữ liệu  
   D. Không cần xử lý vì attribute luôn an toàn

**SQL Injection**

1. Lỗ hổng SQL Injection xảy ra khi:  
   A. Ứng dụng chỉ dùng HTTPS  
   B. Dữ liệu người dùng được nối trực tiếp vào câu SQL  
   C. DB không hỗ trợ prepared statements  
   D. Mật khẩu không được mã hóa
2. Kỹ thuật chuẩn để chống SQL Injection là gì?  
   A. Dùng parameterized queries (prepared statements)  
   B. Dùng regex lọc input  
   C. Dùng JavaScript để lọc dữ liệu trước khi gửi lên server  
   D. Giới hạn độ dài input về 20 ký tự
3. Trong SQLAlchemy ORM, cách nào dưới đây có nguy cơ SQL Injection cao?  
   A. session.query(User).filter(User.id == user\_id).all()  
   B. session.execute("SELECT \* FROM users WHERE id=" + user\_id)  
   C. session.query(User).get(user\_id)  
   D. session.query(User).filter(User.email == email).first()
4. Biện pháp bổ sung giúp giảm rủi ro SQL Injection (ngoài prepared statements) là:  
   A. Sử dụng user DB với quyền hạn tối thiểu  
   B. Chạy DB ở chế độ debug  
   C. Mã hóa toàn bộ dữ liệu input  
   D. Cho phép người dùng nhập SQL trực tiếp
5. Tình huống nào dưới đây KHÔNG phải là SQL Injection?  
   A. Người dùng nhập ' OR '1'='1 và truy vấn trả về toàn bộ bảng  
   B. Người dùng nhập DROP TABLE users; và bảng bị xóa  
   C. Người dùng nhập chuỗi dài vượt max length → server trả lỗi 400  
   D. Người dùng nhập " OR ""=" để bypass xác thực

**XSS**

1. Loại XSS nào xảy ra khi dữ liệu độc hại được lưu vào DB và hiển thị lại nhiều lần?  
   A. Reflected XSS  
   B. Stored XSS  
   C. DOM-based XSS  
   D. Blind XSS
2. Biện pháp nào sau đây KHÔNG phải để chống XSS?  
   A. Escape dữ liệu theo ngữ cảnh (HTML, JS, URL)  
   B. Dùng CSP (Content Security Policy)  
   C. Dùng cookie với HttpOnly và Secure  
   D. Lưu mật khẩu bằng bcrypt
3. Trong ứng dụng Flask/Jinja2, cách nào là an toàn khi hiển thị dữ liệu người dùng?  
   A. {{ user\_input }}  
   B. {{ user\_input | safe }}  
   C. {{ user\_input | raw }}  
   D. {{! user\_input }}
4. Khi cấu hình CSP, directive nào giúp chặn việc nhúng Flash/Java applet?  
   A. default-src 'self'  
   B. object-src 'none'  
   C. script-src 'self'  
   D. frame-ancestors 'none'
5. DOM-based XSS thường xuất hiện khi:  
   A. Dữ liệu từ URL hash/query được chèn trực tiếp vào DOM bằng .innerHTML  
   B. Dữ liệu từ DB được render ở server  
   C. Dữ liệu được lưu trong cookie  
   D. Dữ liệu được escape bằng html.escape()

**File upload attack**

1. Mối nguy hiểm chính khi cho phép upload file trực tiếp vào webroot là gì?  
   A. Người dùng có thể tải file lên chậm  
   B. File độc hại có thể được thực thi như script  
   C. Server tốn nhiều dung lượng  
   D. Khó quản lý tên file trùng nhau
2. Cách kiểm tra loại file an toàn hơn là:  
   A. Dựa vào phần mở rộng file (extension)  
   B. Dựa vào MIME type trong HTTP header  
   C. Kiểm tra magic bytes / signature thật của file  
   D. Tin tưởng tên file người dùng gửi lên
3. Biện pháp nào sau đây giúp giảm rủi ro từ file upload?  
   A. Lưu file ngoài thư mục webroot  
   B. Giữ nguyên tên file gốc để dễ quản lý  
   C. Cho phép upload bất kỳ loại file nào  
   D. Tự động cấp quyền thực thi cho file
4. Tấn công “zip bomb” liên quan đến:  
   A. Upload file nén chứa dữ liệu giải nén cực lớn  
   B. Upload file nén bị lỗi CRC  
   C. Upload file nén chỉ chứa một file trống  
   D. Upload file nén có tên file rất dài
5. Để tránh lỗ hổng khi upload ảnh SVG, cần:  
   A. Cho phép mọi thẻ <script> trong SVG  
   B. Chỉ kiểm tra phần mở rộng .svg  
   C. Sanitize nội dung SVG trước khi lưu hoặc hiển thị  
   D. Lưu SVG ở chế độ text/plain

**PHẦN 2: 5 TÌNH HUỐNG THỰC TẾ**

1. Bạn được giao kiểm tra form đăng ký tài khoản. Người dùng có thể nhập bất kỳ chuỗi nào vào trường "username". Bạn sẽ áp dụng những bước validate và sanitize nào để đảm bảo an toàn?

* **Validate (Xác thực dữ liệu)**

Mục tiêu: đảm bảo dữ liệu nhập vào hợp lệ theo quy tắc nghiệp vụ và không gây lỗi hệ thống.

**Các bước cụ thể:**

**Bước 1: Kiểm tra độ dài**

* Ví dụ: username từ 3 đến 30 ký tự.
* Tránh username quá ngắn (dễ đoán) hoặc quá dài (gây lỗi cơ sở dữ liệu).

**Bước 2: Kiểm tra ký tự hợp lệ**

* Chỉ cho phép: chữ cái (a-z, A-Z), số (0-9), dấu gạch dưới \_ hoặc dấu chấm . (tuỳ chính sách).

Regex mẫu:  
  
 ^[a-zA-Z0-9\_.]{3,30}$

**Bước 3: Tránh từ cấm**

* Không cho phép username chứa từ khóa nhạy cảm như admin, root, system.

**Bước 4: Kiểm tra trùng lặp**

* Kiểm tra cơ sở dữ liệu để tránh username đã tồn tại.

**Bước 5: Xác thực định dạng nâng cao (tuỳ chọn)**

* Không bắt đầu hoặc kết thúc bằng dấu chấm, dấu gạch dưới.
* Không chứa khoảng trắng hoặc ký tự đặc biệt khác.
* **Sanitize (Làm sạch dữ liệu)**

Mục tiêu: ngăn các cuộc tấn công XSS, SQL injection, và bảo vệ hệ thống.

**Các bước cụ thể:**

**Bước 1: Escape ký tự đặc biệt** trước khi hiển thị trên HTML

* Chuyển & → &amp;, < → &lt;, > → &gt;, " → &quot;, ' → &#x27;.

**Bước 2: Sử dụng prepared statement khi lưu vào database**

* Không dùng trực tiếp chuỗi nhập từ người dùng trong query SQL để ngăn SQL injection.

**Bước 3: Loại bỏ khoảng trắng thừa**

* username = username.trim()

**Bước 4: Chuẩn hóa Unicode (tuỳ chọn)**

* Nếu cho phép ký tự quốc tế, chuẩn hóa Unicode để tránh bypass validation.

**- Ví dụ tổng hợp trong code (JavaScript/Node.js)**

function sanitizeUsername(username) {

    // Loại bỏ khoảng trắng 2 đầu

    username = username.trim();

    // Kiểm tra regex

    const regex = /^[a-zA-Z0-9\_.]{3,30}$/;

    if (!regex.test(username)) {

        throw new Error("Tên đăng nhập không hợp lệ");

    }

    return username;

}

// Ví dụ dùng với SQL (MySQL)

const username = sanitizeUsername(req.body.username);

const stmt = 'INSERT INTO users (username) VALUES (?)';

db.execute(stmt, [username]); // Dùng prepared statement

1. Một lập trình viên trong nhóm viết truy vấn như sau:

query = "SELECT \* FROM users WHERE email = '" + email + "' AND pass = '" + password + "'"

Hãy phân tích rủi ro và đề xuất cách sửa an toàn.

**1. Phân tích rủi ro**

query = "SELECT \* FROM users WHERE email = '" + email + "' AND pass = '" + password + "'"

* **Vấn đề**: Chuỗi email và password được nối trực tiếp vào câu SQL mà không kiểm tra hay escape.
* **Hậu quả**: Kẻ tấn công có thể nhập các ký tự SQL đặc biệt để **chèn lệnh** hoặc **bypass đăng nhập**.

**Ví dụ tấn công:**

Nếu kẻ tấn công nhập vào email:

' OR '1'='1

và password:

' OR '1'='1

Thì câu SQL sẽ trở thành:

SELECT \* FROM users WHERE email = '' OR '1'='1' AND pass = '' OR '1'='1'

* Điều này luôn đúng (1='1') → kẻ tấn công có thể đăng nhập mà không cần mật khẩu.

**Các rủi ro khác:**

* Rò rỉ dữ liệu toàn bộ bảng users.  
  Xoá hoặc sửa dữ liệu nếu dùng INSERT/UPDATE/DELETE.
* Khai thác sâu hơn để chiếm quyền server.

**2. Cách sửa an toàn**

**A. Dùng Prepared Statement / Parameterized Query**

Hầu hết các ngôn ngữ lập trình đều hỗ trợ “parameterized query” để tách **dữ liệu người dùng** ra khỏi **câu lệnh SQL**.

**Ví dụ Python + MySQL:**

cursor.execute(

    "SELECT \* FROM users WHERE email = %s AND pass = %s",

    (email, password)

)

* email và password được **escape tự động**, ngăn SQL injection.

**Ví dụ PHP + PDO:**

$stmt = $pdo->prepare("SELECT \* FROM users WHERE email = :email AND pass = :password");

$stmt->execute(['email' => $email, 'password' => $password]);

**B. Hash mật khẩu**

* Không lưu mật khẩu dưới dạng plain text.
* Dùng bcrypt hoặc argon2 để hash và verify:

# Lưu password

hashed = bcrypt.hashpw(password.encode(), bcrypt.gensalt())

# Kiểm tra password khi login

bcrypt.checkpw(password.encode(), hashed)

* Điều này giúp giảm thiểu rủi ro nếu database bị lộ.

**C. Validate & Sanitize input**

* Kiểm tra định dạng email hợp lệ.
* Loại bỏ khoảng trắng dư thừa.
* Không cần escape SQL nếu đã dùng prepared statement, nhưng nên validate input để nâng cao an toàn.

**Tóm tắt:**

1. **Nguy cơ:** SQL Injection → đánh cắp dữ liệu, bypass login, sửa/xoá dữ liệu.  
   **Giải pháp:**
   * **Dùng prepared statement** thay vì nối chuỗi SQL.
   * **Hash mật khẩu** thay vì lưu plain text.
   * **Validate input** để dữ liệu hợp lệ.
2. Hệ thống blog cho phép người dùng đăng bình luận, hiện tại hiển thị lại bình luận trực tiếp bằng HTML thô. Bạn sẽ kiểm tra nguy cơ gì, và giải pháp nào phù hợp để ngăn XSS?

**1. Nguy cơ**

**A. XSS Stored (Persistent)**

* Bình luận được lưu vào database và hiển thị lại cho tất cả người dùng.
* Kẻ tấn công có thể chèn <script> hoặc HTML độc hại:

<script>alert('Tài khoản của bạn bị hack!');</script>

* Khi người dùng khác mở trang, script tự chạy → đánh cắp **cookie, session**, hoặc redirect tới trang độc hại.

**B. XSS Reflected (Non-persistent)**

* Nếu hệ thống phản hồi trực tiếp input (ví dụ trả về “Bạn vừa bình luận: ...”) mà không escape, XSS cũng xảy ra.

**C. Nguy cơ khác**

* Keylogger, redirect phishing, chiếm quyền session.
* Ảnh hưởng tới **uy tín và bảo mật hệ thống**.

**2. Giải pháp ngăn XSS**

**A. Escape HTML khi hiển thị**

* Chuyển các ký tự đặc biệt thành entity HTML:  
    
  + < → &lt;
  + > → &gt;
  + " → &quot;
  + ' → &#x27;
  + & → &amp;

**Ví dụ (JavaScript / Node.js):**

function escapeHTML(str) {

    return str.replace(/&/g, "&amp;")

              .replace(/</g, "&lt;")

              .replace(/>/g, "&gt;")

              .replace(/"/g, "&quot;")

              .replace(/'/g, "&#x27;");

}

// Hiển thị bình luận

commentDiv.innerHTML = escapeHTML(userComment);.

**B. Sử dụng thư viện sanitize**

* Nếu muốn cho phép một số HTML cơ bản (ví dụ <b>, <i>), dùng **whitelist sanitizer**:
  + JavaScript: DOMPurify
  + PHP: HTML Purifier
* Loại bỏ tất cả các thẻ, attribute nguy hiểm (<script>, onload, onclick...)

**C. Content Security Policy (CSP)**

* Thiết lập CSP để ngăn inline script hoặc domain không tin cậy.
* Ví dụ:

Content-Security-Policy: default-src 'self'; script-src 'self';

**D. Validate input**

* Giới hạn độ dài bình luận.
* Cấm những ký tự hoặc pattern nguy hiểm nếu cần.
* Tuy nhiên, **escape HTML vẫn quan trọng**, vì validate không đủ ngăn XSS.

**Tóm tắt:**

1. **Nguy cơ:** XSS Stored/Reflected → đánh cắp cookie, session, tấn công người dùng khác.
2. **Giải pháp:**
   * Escape HTML khi hiển thị dữ liệu người dùng.
   * Hoặc dùng sanitizer whitelist nếu muốn cho phép HTML an toàn.
   * Kết hợp CSP để giảm rủi ro.
   * Validate input về độ dài, ký tự hợp lệ.
3. Trong một ứng dụng thương mại điện tử, người dùng có thể upload ảnh sản phẩm. Bạn sẽ xây dựng cơ chế kiểm tra, xử lý và lưu trữ file upload như thế nào để an toàn?

**1. Validate file khi upload**

**A. Kiểm tra loại file (MIME type + extension)**

* Chỉ cho phép định dạng ảnh: jpg, jpeg, png, gif, webp.
* Không chỉ dựa vào extension, kiểm tra MIME type thực sự:

from imghdr import what

if what(file) not in ['jpeg', 'png', 'gif']:

    raise ValueError("Không phải file ảnh hợp lệ")

**B. Giới hạn kích thước file**

* Tránh DoS hoặc chiếm dụng storage: ví dụ max 5MB.

**C. Kiểm tra nội dung file**

* Tránh file SVG hoặc ảnh chứa script (SVG có thể chứa <script> → XSS).
* Tùy chọn: chuyển SVG sang PNG nếu muốn hiển thị an toàn.

**2. Sanitize và xử lý file**

**A. Đổi tên file**

* Không dùng tên gốc của người dùng để tránh ghi đè.
* Sử dụng **UUID** hoặc timestamp + hash:

1629654321\_f8d7a9e2.png

**B. Resize / tối ưu hình**

* Giảm kích thước, tạo thumbnail, tránh lưu file quá nặng.
* Có thể dùng thư viện Pillow (Python) hoặc Sharp (Node.js).

**C. Quét virus/malware (tuỳ chọn)**

* Sử dụng ClamAV hoặc dịch vụ scan cloud.

**3. Lưu trữ file**

**A. Thư mục riêng, không đặt trong web root**

* Ví dụ: /var/www/uploads/ hoặc cloud storage như S3.
* Tránh truy cập trực tiếp theo đường dẫn do user cung cấp.

**B. Cấu hình server**

* Nếu lưu trong web root, vô hiệu hóa thực thi script:

# Apache

<Directory "/uploads">

    Options -ExecCGI

    AllowOverride None

    Require all granted

</Directory>

**C. Sử dụng đường dẫn ảo khi hiển thị**

* Tránh dùng đường dẫn thật → giảm rủi ro lộ file hệ thống.
* Ví dụ: /images/1629654321\_f8d7a9e2.png

**4. Hiển thị file an toàn**

* Khi render ảnh lên HTML:
  + Dùng tag <img src="..." />
  + Không chèn trực tiếp SVG chứa nội dung người dùng nếu chưa sanitize.
  + Escape URL nếu cần.

1. Khi kiểm thử bảo mật một ứng dụng, bạn phát hiện trang profile có thể hiển thị lại giá trị query string ?name=... mà không escape. Hãy mô tả cách kiểm tra nguy cơ XSS và biện pháp khắc phục.

**1. Cách kiểm tra nguy cơ XSS**

**A. Kiểm thử thủ công**

1. Truy cập URL có query string:

https://example.com/profile?name=Nguyen

* Kiểm tra xem giá trị Nguyen có hiển thị trên trang.

1. Chèn payload thử nghiệm:

<script>alert('XSS')</script>

URL:

https://example.com/profile?name=<script>alert('XSS')</script>

* Nếu trang hiển thị popup hoặc chạy script → XSS tồn tại.

1. Thử các payload an toàn hơn để bypass filter (nếu có):

"><img src=x onerror=alert('XSS')>

* Kiểm tra khả năng escape/encode của ứng dụng.

**B. Sử dụng công cụ tự động**

* Burp Suite, OWASP ZAP để gửi các payload XSS phổ biến vào query string và kiểm tra phản hồi.

**2. Biện pháp khắc phục**

**A. Escape giá trị trước khi hiển thị**

* Khi hiển thị trên HTML, **escape các ký tự đặc biệt**:  
    
  + < → &lt;
  + > → &gt;
  + " → &quot;
  + ' → &#x27;
  + & → &amp;

**Ví dụ (JavaScript / Node.js + template engine)**:

function escapeHTML(str) {

    return str.replace(/&/g, "&amp;")

              .replace(/</g, "&lt;")

              .replace(/>/g, "&gt;")

              .replace(/"/g, "&quot;")

              .replace(/'/g, "&#x27;");

}

const safeName = escapeHTML(req.query.name || '');

res.send(`<h1>Hello, ${safeName}</h1>`);

**B. Sử dụng template engine an toàn**

* Hầu hết template engine hiện đại (Handlebars, Twig, EJS, Django templates) tự escape biến khi render.
* Tránh dùng innerHTML hoặc render thô từ dữ liệu người dùng.

**C. Validate input**

* Kiểm tra định dạng hợp lệ (ví dụ: tên chỉ chứa chữ cái và khoảng trắng, độ dài giới hạn).
* **Lưu ý:** validate không thay thế escape HTML.

**D. Content Security Policy (CSP)**

* Giảm rủi ro nếu escape bị bỏ sót:

Content-Security-Policy: default-src 'self'; script-src 'self'

**3. Tóm tắt**

* **Nguy cơ:** Reflected XSS → kẻ tấn công có thể chèn script, đánh cắp session hoặc redirect.
* **Kiểm thử:** chèn payload <script>alert(1)</script> vào query string, xem có thực thi hay không.
* **Khắc phục:**
  1. Escape HTML trước khi hiển thị giá trị người dùng.
  2. Sử dụng template engine an toàn.
  3. Validate input để giới hạn ký tự hợp lệ.
  4. Áp dụng CSP để giảm tác hại nếu escape thất bại.