

OWASP Top 10 là danh sách 10 rủi ro bảo mật web phổ biến và nguy hiểm nhất, được tổ chức OWASP (Open Web Application Security Project) tổng hợp và cập nhật định kỳ (phiên bản mới nhất hiện nay là OWASP Top 10 – 2021).

Mục tiêu của OWASP Top 10 là giúp lập trình viên, tester, và doanh nghiệp hiểu, phòng tránh, và giảm thiểu các lỗ hổng bảo mật trong ứng dụng web.



A01:2021 – Broken Access Control:

Giới thiệu: Là lỗi cho phép user truy cập chức năng hoặc dữ liệu mà họ không được phép (bypass authorization).

Broken Access Control khiến người dùng có thể:

- Truy cập tài nguyên mà họ không có quyền.
- Gọi chức năng chỉ dành cho role cao hơn.
- Thay đổi danh tính hoặc sở hữu dữ liệu người khác.

Nói cách khác: hệ thống không kiểm tra hoặc kiểm tra sai các rule về authorization.

Dựa trên phân tích các kỹ thuật tấn công phổ biến, lỗ hổng này có thể được phân loại thành 4 nhóm chính dưới đây.

Nhóm 1: Lộ/Rò rỉ Tham chiếu Đổi tượng (Leaking/Tampering Reference to Objects)

Nhóm lỗ hổng này cho phép kẻ tấn công truy cập hoặc sửa đổi dữ liệu bằng cách thao túng các tham chiếu (như ID, khóa) mà hệ thống không xác thực quyền sở hữu.

Insecure Direct Object References (IDOR): Lỗ hổng kinh điển khi ứng dụng sử dụng các tham chiếu trực tiếp đến tài nguyên (ví dụ: ?id=123) mà không có cơ chế kiểm soát truy cập phù hợp, cho phép người dùng thay đổi giá trị này để truy cập dữ liệu của người khác.

Parameter Tampering / Mass Assignment: Kẻ tấn công thay đổi các tham số trong request (từ form, API hoặc URL) để sửa đổi các trường dữ liệu mà họ không được phép, thường khai thác tính năng tự động gán dữ liệu (auto-binding) của framework.

Nhóm 2: Leo thang Đặc quyền (Elevation of Privilege)

Nhóm lỗ hổng này cho phép người dùng thực hiện các chức năng hoặc truy cập các tài nguyên vượt quá cấp độ ủy quyền hiện tại của họ.

Vertical Privilege Escalation: Người dùng thường (User) cố gắng thực hiện các chức năng dành riêng cho quản trị viên (Admin), chẳng hạn như truy cập vào các endpoint /admin.

Horizontal Privilege Escalation: Người dùng có cùng cấp quyền (ví dụ: User A) cố gắng truy cập hoặc thao tác trên tài nguyên thuộc sở hữu của một người dùng ngang hàng khác (User B). IDOR thường là phương tiện để thực hiện hành vi leo thang này.

Missing Function-Level Access Control (MFLA): Lỗ hổng nghiêm trọng khi máy chủ "quên" kiểm tra quyền của người dùng trước khi cho phép họ gọi một chức năng nhạy cảm (thường thông qua API hoặc đường dẫn URL). Đây thường là nguyên nhân cốt lõi dẫn đến leo thang đặc quyền.

Nhóm 3: Bỏ qua Luồng Kiểm soát Truy cập (Bypassing Access Control Workflow)

Nhóm lỗ hổng này xảy ra khi kẻ tấn công có thể truy cập trực tiếp vào một trang hoặc tài nguyên mà không cần tuân theo quy trình làm việc hợp lệ của ứng dụng.

Force Browsing: Kỹ thuật mà kẻ tấn công trực tiếp truy cập một URL nhạy cảm (ví dụ: /admin/payment-details.csv) mà không cần điều hướng qua các bước xác thực hoặc ủy quyền trung gian.

Unprotected API Endpoints: Một trường hợp đặc biệt của MFLA, trong đó các điểm cuối API hoàn toàn không được bảo vệ, cho phép bất kỳ ai cũng có thể gọi và sử dụng chúng mà không cần xác thực hoặc kiểm tra quyền.

Nhóm 4: Lợi dụng Cơ chế Xác thực (Compromising Authentication Mechanisms)

Nhóm lỗ hổng này nhắm mục tiêu vào chính cơ chế quản lý danh tính và phiên của người dùng để mạo danh họ.

Session Hijacking / Cookie Manipulation: Kẻ tấn công đánh cắp, dự đoán hoặc thao túng cookie phiên để chiếm đoạt quyền điều khiển phiên đăng nhập của người dùng hợp lệ.

CORS Misconfiguration: Cấu hình Chia sẻ Tài nguyên có Nguồn gốc Chéo (CORS) không chính xác, cho phép các tên miền bên ngoài không đáng tin cậy truy cập trái phép vào tài nguyên của ứng dụng, dẫn đến rò rỉ dữ liệu nhạy cảm.

1. Insecure Direct Object References (IDOR)

Mô tả:

Ứng dụng trực tiếp dùng identifier (ID, filename, UUID, path) để truy xuất tài nguyên mà không xác minh quyền sở hữu/authorization ở server-side. Kẻ tấn công thay ID trong request để truy cập dữ liệu của người khác.

Ví dụ tấn công:

Request: GET /invoices?id=1234 → attacker đổi id=1234 → id=1235 để xem hóa đơn của user khác; hoặc tải GET /files/download?name=secret.pdf → đổi name=customer_info.pdf.

Kiểm thử:

- Liệt kê các endpoint trả tài nguyên theo ID (query param, path param, body).
- Thử thay numeric IDs tăng/giảm, UUIDs sửa một đoạn, filenames, path traversal.
- Thử dùng session/token của user khác (impersonation).
- Kiểm tra responses: có trả dữ liệu người khác không (200 + nội dung), hay trả 403/404.
- Kết hợp forceful browsing (truy cập URL file/thư mục chưa link).

Phòng chống:

- Luôn kiểm tra quyền sở hữu (owner check) server-side trước khi trả tài nguyên.
- Sử dụng mapping server-side (không dùng ID public làm căn cứ quyền).
- Trả 403 thay vì 404 cho access-denied (theo chính sách), và hạn chế chi tiết lỗi.
- Áp kiểm soát truy cập theo resource + role + context (ACL).
- Log mọi attempt truy cập không hợp lệ để phát hiện quét/scan.

2. Missing Function-Level Access Control (MFLA)

Mô tả:

Chức năng hoặc endpoint chỉ dành cho nhóm/role nhất định nhưng không có kiểm tra role ở backend — thường chỉ bị ẩn ở UI/menus. Kẻ tấn công gọi trực tiếp API/route để thực thi chức năng admin.

Ví dụ tấn công:

Frontend ẩn nút “Delete user” cho non-admin, nhưng backend không kiểm tra role; attacker gửi POST /admin/deleteUser?id=5 bằng token user thường và xóa user.

Kiểm thử:

- Liệt kê các chức năng admin/privileged (create/delete user, change roles, export data).
- Gửi request đến các endpoint đó bằng token user bình thường, token expired, hoặc không token.
- Thủ đổi HTTP method (GET/POST/PUT/DELETE) và các tham số.
- Kiểm tra response codes; nếu 200 với hành động được thực hiện → lỗi nghiêm trọng.

Phòng chống:

- Enforce function-level authorization ở server (middleware/guard) cho mọi endpoint.
- Áp nguyên tắc *deny by default* — require explicit permission.
- Centralize kiểm tra quyền (không rải rác trong controller).
- Document & test permission matrix; có unit/integration tests cho các đường dẫn privileged.
- Review định kỳ routes/permissions để đóng các đường dẫn legacy.

3. Horizontal / Vertical Privilege Escalation

Mô tả:

Người dùng nâng quyền hoặc truy cập dữ liệu ngoài phạm vi:

- **Horizontal:** user A truy cập dữ liệu/đối tượng của user B (cùng cấp).
- **Vertical:** user bình thường thực hiện hành động chỉ dành cho admin (tăng quyền).

Ví dụ tấn công:

- *Horizontal:* đổi profileId=200 → profileId=201 và thấy info người khác.

- *Vertical*: gửi PATCH /users/123 với body {"role":"admin"} hoặc chỉnh cookie/session id sang id admin và được chấp nhận.

Kiểm thử:

- Thủ impersonation: sử dụng token/session của user khác.
- Tamper token/session: thay session id, sửa JWT payload (nếu không verify signature).
- Thủ sửa các trường role/isAdmin trong request body.
- Kiểm tra hành vi khi dùng session khác (logout/login user khác).
- Kiểm tra endpoint trả dữ liệu nhạy cảm có ràng buộc owner không.

Phòng chống:

- Validate quyền mọi request: owner check + role check server-side.
- Token/session phải được ký và kiểm tra (JWT verify signature) + có expiry; rotate session sau login.
- Không tin input client cho trường quyền (role, isAdmin); ignore hoặc validate server-side.
- Áp principle of least privilege cho tài khoản dịch vụ/DB.
- Ghi log và alert hành vi bất thường (nhiều attempts truy cập tài nguyên khác).
- Thực hiện automated tests để phát hiện regressions về phân quyền.

4. Bypassing Access Control Checks

Mô tả:

Người dùng tìm cách vượt qua hoặc bỏ qua các kiểm tra phân quyền để truy cập chức năng/tài nguyên không được phép — thường bằng cách tận dụng logic kiểm tra yếu hoặc điểm vào/luồng thay thế.

Ví dụ tấn công:

Gọi trực tiếp endpoint admin (/admin/create) bằng token user bình thường; dùng API version cũ (/v1/legacy) chưa có kiểm tra; thay đổi HTTP method (GET → POST) để né middleware.

Kiểm thử:

- Gửi request đến các endpoint nhạy cảm bằng token user thường, token rỗng, token bị sửa.
- Thủ truy cập endpoint dùng HTTP method khác.
- Thủ các phiên bản API (v1/v2) hoặc URL cũ.
- Kiểm tra các luồng phụ (callbacks, webhooks) có bỏ qua auth không.

Phòng chống:

- Enforce auth + permission ở server-side cho mọi route và mọi method.
- Không dựa vào frontend để ẩn/kiểm soát.
- Dùng centralized middleware/guard để kiểm tra role.
- Tắt/redirect các API/route cũ không dùng.
- Thực hiện logging và alert khi phát hiện access denied bypass.

5. Force Browsing

Mô tả:

Truy cập trực tiếp các đường dẫn/tập tin mà không qua UI (không bị link), tận dụng việc server serve file/URL mà không yêu cầu xác thực thích hợp.

Ví dụ tấn công:

Truy cập <https://site.com/admin/backup.zip> hoặc <https://site.com/.git/config> mà không cần đăng nhập; tìm file config/db trên webroot.

Kiểm thử:

- Quét thử các đường dẫn phổ biến (/admin, /backup, /old, /staging, .git/, config/).
- Dùng wordlists để brute-force đường dẫn (gobuster/feroxbuster).
- Thủ truy cập file tải về/backup trực tiếp.
- Kiểm tra file .bak/.old, folder test, staging endpoints.

Phòng chống:

- Không expose file nhạy cảm vào webroot; di chuyển file config ra khỏi document root.
- Yêu cầu authentication/authorization trước khi serve mọi resource.
- Đặt rules trên web server (deny .git, .env, *.bak).

- Sử dụng robots/htaccess/nginx rules và monitoring cho truy cập bất thường.

6. Unprotected API Endpoints

Mô tả:

API public hoặc nội bộ không được bảo vệ đúng (không yêu cầu auth, token dễ đoán, thiếu rate limit), dẫn đến lộ dữ liệu hoặc thao tác trái phép.

Ví dụ tấn công:

GET /api/v1/users trả về danh sách toàn bộ user mà không cần token; POST /api/v1/send-email cho phép gửi email spam.

Kiểm thử:

- Kiểm tra tất cả endpoint xem có yêu cầu auth không (kể cả /health, /metrics).
- Gửi request không có token, token rỗng, token expired để xem phản hồi.
- Kiểm tra quyền của token (scopes) trên endpoint nhạy cảm.
- Thủ rate-limit bypass, replay request.

Phòng chống:

- Bắt buộc authentication & authorization middleware cho mọi API (default-deny).
- Sử dụng token có scope/permission rõ ràng; validate signature/expiry.
- Áp rate limiting, throttling, IP allowlist cho API nội bộ.
- Cấu hình API gateway/ WAF để bảo vệ endpoint phổ biến.

7. Parameter Tampering / Mass Assignment

Mô tả:

Người dùng chỉnh sửa tham số request (query/form/JSON) hoặc lợi dụng binding tự động để thay đổi field không được phép (ví dụ role, isAdmin, price) dẫn tới thay đổi trạng thái hoặc nâng quyền.

Ví dụ tấn công:

Gửi JSON:

```
{  
  "username": "user1",  
  "role": "admin"
```

}

vào endpoint update profile; server chấp nhận và nâng quyền. Hoặc sửa price trong request mua hàng.

Kiểm thử:

- Thử thêm các trường ẩn (role, isAdmin, balance) vào request và quan sát.
- Thử sửa trường ID/owner để thay đổi tài nguyên của người khác.
- Kiểm tra mass assignment bằng cách gửi object lớn gồm mọi field model.
- Xem server có whitelist/blacklist fields khi binding không.

Phòng chống:

- Dùng whitelist cho fields được phép bind/update (deny by default).
- Tránh mapping trực tiếp object client → model; kiểm tra server-side owner/permission cho mỗi field.
- Sử dụng DTO/serializers rõ ràng, validate schema.
- Log các thay đổi quyền/trường nhạy cảm và alert.

8. CORS Misconfiguration (liên quan kiểm soát truy cập tài nguyên)

Mô tả:

Cấu hình CORS lỏng lẻo cho phép origin không tin cậy truy cập API (ví dụ Access-Control-Allow-Origin: * cho API private), dẫn đến rò rỉ chức năng/dữ liệu qua trình duyệt.

Ví dụ tấn công:

Một trang độc hại evil.com thực hiện request tới api.corp.com từ trình duyệt người dùng đã đăng nhập; nếu API chấp nhận origin đó hoặc *, trình duyệt cho phép truy xuất dữ liệu.

Kiểm thử:

- Kiểm tra header Access-Control-Allow-Origin cho các endpoint (đặc biệt API trả dữ liệu nhạy cảm).
- Gửi request từ origin giả (via browser fetch) và xem liệu browser có cho phép đọc response.
- Kiểm thử preflight (OPTIONS) và xem Access-Control-Allow-Credentials, Allow-Headers cấu hình ra sao.

Phòng chống:

- Không dùng * cho API cần auth; chỉ whitelist origin tin cậy.
- Nếu dùng credentials (cookies), set Access-Control-Allow-Credentials: true với origin cụ thể (không *).
- Giới hạn Access-Control-Allow-Headers và methods cần thiết.
- Thực hiện CSRF protection cho các endpoint dùng cookie-based auth.

9. Session Hijacking / Cookie Manipulation

Mô tả:

Kẻ tấn công chiếm đoạt session (cookie/token) hoặc chỉnh sửa cookie để giả mạo người dùng, dẫn đến truy cập trái phép hoặc nâng quyền.

Ví dụ tấn công:

- XSS đánh cắp cookie session rồi dùng nó để impersonate.
- Sửa cookie JSON không được ký (change isAdmin=false → true).
- Replay token cũ không hết hạn.

Kiểm thử:

- Kiểm tra cookie có HttpOnly, Secure, SameSite không.
- Thủ thay cookie/session id bằng id khác để xem server có ràng buộc owner không.
- Kiểm tra JWT signature verification, expiry, revocation.
- Thủ session fixation (gán session trước khi login).

Phòng chống:

- Thiết lập cookie flags: HttpOnly, Secure, SameSite=Strict/Lax thích hợp.
- Ràng buộc session với metadata (IP, User-Agent) nếu phù hợp; rotate session id sau login.
- Sử dụng signed & encrypted tokens; validate signature và expiry server-side.
- Sử dụng CSRF token, giảm TTL cho session nhạy cảm, hỗ trợ revocation/blacklist.
- Giảm nguy cơ XSS (input/output encoding) để tránh cookie theft.

A02:2021 – Injection:

Giới thiệu: Lỗ hổng Injection xảy ra khi ứng dụng xử lý không đúng cách các dữ liệu đầu vào không đáng tin cậy, cho phép kẻ tấn công chèn và thực thi các mã độc hại vào hệ thống xử lý phía backend hoặc frontend.

Dựa trên phân tích các kỹ thuật tấn công phổ biến, lỗ hổng Injection có thể được phân loại thành 4 nhóm chính dựa trên mục tiêu và cơ chế tấn công.

Nhóm 1: Injection Truy vấn Hệ thống Quản lý Dữ liệu

Nhóm lỗ hổng này nhắm mục tiêu vào các hệ thống lưu trữ và truy vấn dữ liệu, cho phép kẻ tấn công can thiệp vào cấu trúc truy vấn để đọc, ghi hoặc xóa dữ liệu trái phép.

SQL Injection (SQLi): Lỗ hổng kinh điển khi kẻ tấn công chèn các mã SQL độc hại vào các tham số đầu vào, cho phép can thiệp trực tiếp vào cơ sở dữ liệu quan hệ.

NoSQL Injection: Phiên bản hiện đại của SQLi, nhắm vào các cơ sở dữ liệu NoSQL (MongoDB, CouchDB) thông qua việc chèn các toán tử truy vấn đặc biệt.

ORM Injection / Query Builder Injection: Lỗ hổng xảy ra khi sử dụng không đúng cách các Object-Relational Mapping framework hoặc query builders, cho phép bỏ qua các ràng buộc truy vấn an toàn.

XPath / XML Injection: Kẻ tấn công chèn các biểu thức XPath độc hại để thao túng việc truy vấn dữ liệu trong các tài liệu XML.

Nhóm 2: Injection Mã Thực thi Hệ thống

Nhóm lỗ hổng này cho phép kẻ tấn công thực thi các mã hoặc lệnh tùy ý trên hệ thống máy chủ, dẫn đến việc chiếm quyền điều khiển toàn bộ hoặc một phần hệ thống.

Command / OS Command Injection: Lỗ hổng nghiêm trọng cho phép kẻ tấn công chèn và thực thi các lệnh hệ điều hành trái phép thông qua ứng dụng.

Server-Side Template Injection (SSTI): Kẻ tấn công chèn mã độc vào các template engine phía server (Jinja2, Twig, Freemarker), cho phép thực thi mã tùy ý trên máy chủ.

LDAP Injection: Chèn các ký tự đặc biệt vào các truy vấn LDAP để bỏ qua xác thực hoặc truy xuất thông tin nhạy cảm từ directory services.

Nhóm 3: Injection Xử lý Tài liệu & Định dạng

Nhóm lỗ hổng này khai thác các lỗ hổng trong quá trình xử lý và phân tích cú pháp các tài liệu và định dạng dữ liệu phức tạp.

XML External Entity (XXE): Lợi dụng tính năng External Entity trong bộ phân tích XML để đọc tệp tin nội bộ, quét mạng nội bộ hoặc thực thi các yêu cầu SSRF.

CSV / Formula Injection (Excel): Chèn các công thức độc hại vào các tệp CSV hoặc Excel, khiến chúng được thực thi khi người dùng mở tệp trong các ứng dụng bảng tính.

Nhóm 4: Injection Giao thức & Ứng dụng Client-side

Nhóm lỗ hổng này nhắm mục tiêu vào việc xử lý giao thức và trình duyệt của người dùng cuối, ảnh hưởng đến tính bảo mật của phiên làm việc và dữ liệu client.

Cross-Site Scripting (XSS): Chèn và thực thi các script độc hại trong trình duyệt của người dùng, cho phép đánh cắp cookie, chiếm phiên đăng nhập, hoặc thay đổi nội dung trang web.

CRLF / HTTP Response Splitting & Header Injection: Chèn các ký tự CRLF (Carriage Return Line Feed) để thao túng phản hồi HTTP, tiêm các header độc hại hoặc chia nhỏ phản hồi thành các phần khác nhau.

1) SQL Injection (SQLi)

Mô tả:

Ứng dụng chèn trực tiếp dữ liệu người dùng vào câu truy vấn SQL mà không kiểm soát, cho phép kẻ tấn công thay đổi logic của truy vấn, dẫn đến đọc, sửa hoặc xóa dữ liệu trái phép.

Ví dụ tấn công:

Kẻ tấn công nhập dữ liệu đặc biệt vào ô đăng nhập hoặc URL để khiến truy vấn trả về toàn bộ tài khoản, bỏ qua kiểm tra mật khẩu, hoặc xóa bảng dữ liệu.

Kiểm thử:

Tìm các điểm nhập dữ liệu được đưa vào câu SQL (form, URL, header, cookie).

Thử thay đổi dữ liệu đầu vào để quan sát lỗi SQL, độ trễ phản hồi hoặc kết quả bất thường.

Sử dụng công cụ tự động như Burp, sqlmap để xác nhận lỗ hổng.

Phòng chống:

Sử dụng truy vấn có tham số (prepared statement).

Không nối chuỗi trực tiếp trong truy vấn.

Kiểm tra và giới hạn định dạng đầu vào.

Tài khoản truy cập CSDL chỉ nên có quyền tối thiểu.

Theo dõi log để phát hiện các lỗi SQL bất thường.

2) NoSQL Injection

Mô tả:

Xảy ra khi ứng dụng NoSQL (như MongoDB, CouchDB) chèn dữ liệu đầu vào trực tiếp vào truy vấn mà không kiểm tra, cho phép kẻ tấn công thay đổi điều kiện tìm kiếm hoặc bỏ qua xác thực.

Ví dụ tấn công:

Kẻ tấn công gửi dữ liệu JSON giả mạo làm thay đổi điều kiện tìm kiếm người dùng, khiến hệ thống xác thực sai và cho phép truy cập bất hợp pháp.

Kiểm thử:

Quan sát các API sử dụng dữ liệu JSON hoặc object để truy vấn.

Thay đổi kiểu dữ liệu hoặc thêm khóa bất thường để xem phản ứng của server.

Phòng chống:

Áp dụng kiểm tra định dạng và kiểu dữ liệu (JSON schema validation).

Chỉ cho phép các trường hợp hợp lệ, loại bỏ khóa lạ.

Sử dụng driver chính thức hỗ trợ tham số hóa truy vấn.

Giới hạn quyền của tài khoản truy cập DB.

3) Command / OS Injection

Mô tả:

Ứng dụng gọi trực tiếp lệnh hệ điều hành bằng dữ liệu người dùng. Kẻ tấn công có thể chèn thêm lệnh để thực thi tùy ý trên server.

Ví dụ tấn công:

Ứng dụng có chức năng "ping" hoặc "convert file" và dùng dữ liệu nhập từ người dùng. Kẻ tấn công thêm nội dung đặc biệt khiến hệ thống thực thi lệnh không mong muốn như đọc file hoặc xóa dữ liệu.

Kiểm thử:

Tìm chỗ ứng dụng có thể gọi lệnh hệ thống.

Thử thay đổi input để tạo hành vi bất thường hoặc thời gian phản hồi thay đổi.

Phòng chống:

Không sử dụng lệnh hệ điều hành với dữ liệu người dùng.

Nếu buộc phải dùng, hãy giới hạn input bằng danh sách cho phép.

Dùng API hoặc thư viện thay thế thay vì gọi trực tiếp lệnh shell.

Giới hạn quyền hệ thống của tiến trình.

4) LDAP Injection

Mô tả:

Khi ứng dụng sử dụng dữ liệu người dùng để tạo truy vấn LDAP mà không kiểm soát, kẻ tấn công có thể thay đổi filter tìm kiếm, truy cập trái phép hoặc bỏ qua xác thực.

Ví dụ tấn công:

Trong màn hình đăng nhập nội bộ, kẻ tấn công gửi dữ liệu đặc biệt khiến hệ thống LDAP trả về danh sách toàn bộ tài khoản thay vì chỉ một người.

Kiểm thử:

Thử nhập các ký tự hoặc chuỗi bất thường trong các trường tên hoặc mã định danh để xem kết quả trả về có thay đổi.

Phân tích phản hồi của hệ thống để phát hiện dấu hiệu lỗi LDAP.

Phòng chống:

Escape hoặc mã hóa dữ liệu đầu vào trước khi chèn vào filter LDAP.

Xác thực dữ liệu đầu vào chỉ gồm các ký tự hợp lệ.

Sử dụng hàm hoặc API hỗ trợ truy vấn an toàn.

5) XPath / XML Injection

Mô tả:

Khi ứng dụng truy vấn dữ liệu trong file XML và chèn trực tiếp input vào câu lệnh XPath mà không kiểm tra, có thể cho phép kẻ tấn công đọc dữ liệu nhạy cảm hoặc bỏ qua xác thực.

Ví dụ tấn công:

Kẻ tấn công nhập chuỗi đặc biệt vào trường đăng nhập khiến hệ thống XML trả về tất cả người dùng mà không cần mật khẩu.

Kiểm thử:

Phân tích điểm nào của ứng dụng thao tác với file XML.

Thử thay đổi input để xem dữ liệu truy vấn có trả về bất thường.

Phòng chống:

Không chèn trực tiếp input vào biểu thức XPath.

Escape dữ liệu đầu vào.

Sử dụng thư viện hoặc API có tham số hóa.

Kiểm tra dữ liệu đầu vào theo định dạng.

6) Server-Side Template Injection (SSTI)

Mô tả:

Xảy ra khi engine template (như Jinja2, Twig, Freemarker) render dữ liệu người dùng mà

không lọc. Kẻ tấn công có thể chèn biểu thức để đọc dữ liệu nhạy cảm hoặc thực thi mã server-side.

Ví dụ tấn công:

Trang web hiển thị lời chào với tên người dùng. Kẻ tấn công chèn biểu thức đặc biệt khiến hệ thống thực thi phép toán hoặc in ra cấu hình nội bộ.

Kiểm thử:

Nhập các ký tự đặc trưng của template và quan sát kết quả hiển thị.

Nếu giá trị được tính toán hoặc trả về khác thường, có thể có SSTI.

Phòng chống:

Không render trực tiếp dữ liệu chưa tin cậy trong template.

Escape dữ liệu trước khi render.

Tắt các tính năng đánh giá biểu thức trong dữ liệu người dùng.

Dùng engine an toàn (như Mustache) nếu chỉ cần hiển thị text.

7) Cross-Site Scripting (XSS)

Mô tả:

Xảy ra khi dữ liệu người dùng được hiển thị lên trang mà không mã hóa đúng cách, cho phép kẻ tấn công chèn mã script chạy trong trình duyệt của nạn nhân, đánh cắp cookie hoặc thực hiện hành động thay mặt người dùng.

Ví dụ tấn công:

Kẻ tấn công đăng bình luận chứa nội dung đặc biệt, khi người dùng khác mở trang, trình duyệt thực thi mã và gửi thông tin người dùng về cho kẻ tấn công.

Kiểm thử:

Thử nhập ký tự hoặc đoạn văn bản đặc biệt vào các trường hiển thị trên giao diện, kiểm tra xem nó có được thực thi hay hiển thị nguyên dạng.

Kiểm tra cả các tham số URL và phản hồi DOM.

Phòng chống:

Mã hóa đầu ra phù hợp với ngữ cảnh (HTML, thuộc tính, JavaScript).

Xác thực và làm sạch dữ liệu nhập.

Bật Content Security Policy (CSP).

Sử dụng cookie có cờ HttpOnly để ngăn đọc bằng script.

8) CRLF / HTTP Header Injection

Mô tả:

Ứng dụng không lọc ký tự điều khiển khi tạo header HTTP, cho phép kẻ tấn công thêm header giả hoặc chia đôi phản hồi (HTTP response splitting).

Ví dụ tấn công:

Kẻ tấn công thay đổi giá trị của tham số dùng để tạo header phản hồi, khiến hệ thống thêm dòng header mới hoặc chuyển hướng người dùng sang trang giả mạo.

Kiểm thử:

Quan sát cách server tạo header phản hồi.

Thử thay đổi input và kiểm tra xem có header lạ hoặc redirect bất thường không.

Phòng chống:

Loại bỏ các ký tự điều khiển trong input (xuống dòng, tab...).

Sử dụng hàm có sẵn của framework để thiết lập header.

Giới hạn giá trị hợp lệ cho các tham số liên quan đến header.

9) CSV / Formula Injection (Excel Injection)

Mô tả:

Khi ứng dụng xuất dữ liệu người dùng sang file CSV hoặc Excel mà không xử lý, các chuỗi bắt đầu bằng ký tự đặc biệt có thể bị Excel coi là công thức và thực thi.

Ví dụ tấn công:

Người dùng nhập dữ liệu giả mạo khiến khi nhân viên mở file báo cáo, Excel tự động chạy công thức có thể dẫn tới rò rỉ thông tin.

Kiểm thử:

Nhập dữ liệu bắt đầu bằng ký tự đặc biệt và quan sát hành vi khi mở file xuất ra.

Kiểm tra xem có ô nào tự động tính toán hay cảnh báo không.

Phòng chống:

Thêm ký tự an toàn (như dấu nháy đơn hoặc khoảng trắng) trước chuỗi bắt đầu bằng ký tự đặc biệt.

Làm sạch dữ liệu trước khi export.

Thông báo người dùng mở file ở chế độ an toàn.

10) ORM / Query Builder Injection

Mô tả:

Dù sử dụng ORM, nếu lập trình viên nối chuỗi hoặc chèn input vào truy vấn thủ công, kẻ tấn công vẫn có thể thực hiện tiêm lệnh giống SQLi.

Ví dụ tấn công:

Ứng dụng sử dụng ORM để tìm kiếm người dùng, nhưng vì lập trình viên gộp trực tiếp input vào câu query, kẻ tấn công có thể thao túng kết quả hoặc bỏ qua điều kiện lọc.

Kiểm thử:

Tìm các chỗ dùng hàm “raw query” hoặc nối chuỗi trong ORM.

Thử thay đổi dữ liệu nhập và kiểm tra kết quả trả về.

Phòng chống:

Luôn dùng binding hoặc placeholder có sẵn của ORM.

Tránh chèn input trực tiếp vào câu truy vấn.

Kiểm tra dữ liệu đầu vào và giới hạn quyền DB.

11) XML External Entity (XXE)

Mô tả:

Ứng dụng xử lý XML nhưng cho phép khai báo external entities, khiến kẻ tấn công có thể yêu cầu server tải hoặc đọc file cục bộ, hoặc gửi yêu cầu đến máy khác.

Ví dụ tấn công:

Kẻ tấn công gửi XML đặc biệt khiến server đọc file cấu hình hệ thống hoặc gọi đến địa chỉ nội bộ (SSRF).

Kiểm thử:

Gửi XML có khai báo external entity và quan sát xem server có cố tải dữ liệu từ nguồn ngoài hay không.

Phòng chống:

Tắt tính năng external entity (DTD) trong parser XML.

Sử dụng thư viện XML an toàn.

Không parse XML từ nguồn không tin cậy hoặc xác thực cấu trúc trước khi xử lý.

3 loại XSS:

1) Reflected XSS (XSS phản chiếu)

Mô tả:

Dữ liệu độc hại được gửi trong request (URL, form, header, ...) → server phản hồi lại ngay trong HTML response mà không được escape, dẫn đến thực thi script ngay khi người dùng mở link.

Nó không lưu trong hệ thống, chỉ tồn tại tạm thời khi người dùng click đường dẫn hoặc gửi form.

Cách hoạt động:

Attacker tạo URL chứa payload (vd: <https://example.com/search?q=<script>...>).

Người dùng click vào link đó.

Server lấy q và render lại vào HTML mà không escape.

Trình duyệt nạn nhân chạy script ngay khi trang được load.

Ví dụ thực tế (tổng quát):

Khi bạn tìm kiếm, web hiển thị lại từ khóa bạn nhập — attacker gửi link giả “kết quả tìm kiếm” nhưng chứa mã JS.

Người dùng click → script chạy trong trình duyệt họ.

Kiểm thử:

Quan sát các URL params, form data phản hồi trực tiếp trên trang.

Thử payload HTML/JS đơn giản (test), xem phản hồi có bị render lại không.

Nếu thấy hiển thị “thô” mà không escape → nghi ngờ Reflected XSS.

Phòng chống:

Escape output theo context.

Không phản chiếu trực tiếp input người dùng trong HTML.

Dùng CSP, validate input.

2) Stored XSS (XSS lưu trữ)

Mô tả:

Payload XSS được lưu vào hệ thống (database, file, log, comment, user profile, ...).

Mỗi khi người khác truy cập dữ liệu đó, script sẽ chạy — nguy hiểm nhất, vì có thể lan truyền đến nhiều người (admin, khách hàng,...).

Cách hoạt động:

Attacker gửi input độc hại vào form (ví dụ comment, bio, tên user, message,...).

Server lưu input vào DB mà không sanitize.

Khi trang hiển thị dữ liệu này, script thực thi trên trình duyệt nạn nhân.

Ví dụ thực tế:

Attacker đăng comment chứa đoạn script vào bài viết.

Khi admin xem bài, script chạy, gửi cookie/session admin về attacker → chiếm tài khoản quản trị.

Kiểm thử:

Tìm chỗ người dùng nhập dữ liệu có thể hiển thị lại (comment, review, profile name,...).

Gửi input test (HTML đơn giản) rồi xem lại trang hiển thị.

Nếu payload được lưu và hiển thị “sống” → stored XSS.

Phòng chống:

Escape dữ liệu khi render (không khi lưu).

Sanitize input (nếu cho phép HTML thì phải dùng whitelist).

HttpOnly cookie, CSP để giảm thiệt hại.

Xác thực kỹ quyền nhập dữ liệu (VD: chỉ cho markdown an toàn).

3) DOM-based XSS (XSS phía client)

Mô tả:

Không qua server. Toàn bộ quá trình xảy ra trên trình duyệt, do JavaScript của client xử lý dữ liệu từ URL, DOM hoặc localStorage mà không kiểm soát — rồi chèn vào DOM bằng các API nguy hiểm (innerHTML, document.write, eval, ...).

=> Tức là không có request-response chứa script từ server, mà JS trong trang tự tạo ra XSS.

Cách hoạt động:

Trang web có JS đọc dữ liệu từ URL (location.hash, location.search, ...).

JS gán giá trị này vào DOM (vd: element.innerHTML = userInput).

Nếu dữ liệu trong URL chứa mã script, script đó chạy ngay trên trình duyệt.

Ví dụ thực tế:

URL https://example.com/page#

Trang đọc location.hash và chèn vào DOM để hiển thị.

JS trong trang không escape → DOM-based XSS.

Kiểm thử:

Kiểm tra các thao tác JS với document.write, innerHTML, outerHTML, eval, insertAdjacentHTML, location, innerText, setTimeout(string).

Sử dụng dev tools, search từ khóa innerHTML trong JS để tìm sink.

Thêm payload vào URL fragment hoặc params xem có render không.

Phòng chống:

Không dùng các API nguy hiểm với dữ liệu người dùng.

Escape dữ liệu trước khi gán vào DOM.

Dùng frameworks tự động encode (React, Vue, Angular).

CSP và sandbox iframe nếu cần.

CSRF

CSRF (Cross-Site Request Forgery) — Định nghĩa ngắn

CSRF là một lỗ hổng bảo mật web cho phép kẻ tấn công ép người dùng đang đăng nhập hợp pháp (với session/cookie) gửi các yêu cầu trái ý đến ứng dụng mục tiêu. Những yêu cầu này được trình duyệt gửi kèm cookie/credential hợp lệ, vì vậy server coi đó là request hợp lệ từ người dùng và thực hiện hành động (ví dụ chuyển tiền, đổi email, xóa tài nguyên).

Cơ chế hoạt động — mô tả chi tiết từng bước

Nạn nhân đăng nhập vào ứng dụng mục tiêu (ví dụ bank.example) và trình duyệt lưu cookie phiên (session cookie). Cookie này được trình duyệt gửi tự động cho mọi request tới domain bank.example.

Kẻ tấn công chuẩn bị một trang web độc hại (hoặc email/forum post chứa link/form) có nội dung khiến trình duyệt nạn nhân gửi một request tới ứng dụng mục tiêu khi nạn nhân truy cập trang đó. Yêu cầu có thể là một form auto-submit, một thẻ `` hay `<script>` (cho GET), hoặc script tạo POST/XHR (trong hạn chế CORS).

Khi nạn nhân (vẫn đang đăng nhập ở bank.example) truy cập trang độc hại, trình duyệt gửi request tới bank.example kèm cookie phiên tự động. Vì server chỉ dựa vào cookie để chứng thực, server coi request là được thực hiện bởi người dùng hợp lệ và thực hiện hành động.

Kết quả: hành động trái ý (chuyển tiền, đổi mật khẩu, xóa dữ liệu...) xảy ra mà nạn nhân không hay biết.

Ghi chú: CSRF lợi dụng đặc tính “trình duyệt tự quản lý và gửi cookie” cùng ưu tiên server tin tưởng cookie cho authentication.

Các biến thể tấn công CSRF (tổng quan)

Image/GET trick: dùng thẻ `` hoặc link để gửi GET tới endpoint có side-effect (lỗi thiết kế nếu GET thay đổi trạng thái).

Auto-submit form: trang attacker chứa form POST tới endpoint mục tiêu và tự submit khi trang load.

AJAX/fetch trick: nếu endpoint ít ràng buộc CORS hoặc token được truyền trong cookie, attacker có thể cố gắng sử dụng script để gửi XHR (thường bị chặn nếu CORS đúng).

Login CSRF: ép nạn nhân đăng nhập sử dụng session do attacker kiểm soát (session fixation variant) — gây tracking hoặc escalate attack chain.

Chained attacks: kết hợp social engineering hoặc khai thác khác để kéo dài cửa tấn công (ví dụ dùng CSRF để thay đổi email rồi dùng password reset).

Cách kiểm thử CSRF (practical checklist cho tester)

Liệt kê mọi endpoint thay đổi trạng thái (POST/PUT/DELETE, có hiệu ứng trên server).

Kiểm tra xem các endpoint đó có yêu cầu token CSRF (hidden field, header) hay kiểm tra header Origin/Referer không.

Sao chép request mục tiêu và chèn vào một trang HTML đơn giản trên domain khác: tạo form có các trường tương ứng rồi auto-submit; hoặc tạo thẻ `/<script>` để gọi GET.

Mở trang attacker trong trình duyệt mà đang đăng nhập trên ứng dụng mục tiêu; quan sát xem hành động có xảy ra không.

Thử gửi request thiếu token CSRF, token sai, hoặc với header Origin/Referer khác để xem server có từ chối hay không.

Kiểm tra cấu hình cookie: kiểm tra thuộc tính SameSite, Secure, HttpOnly; kiểm tra nếu cookie có SameSite gì thì ảnh hưởng đến các vector cross-site thế nào.

Đối với API JSON/SPA: kiểm tra việc authentication dùng cookie hay header (bearer token); nếu dùng cookie, thử CSRF vectors tương tự.

Các biện pháp phòng chống CSRF (thực hành, ưu tiên triển khai)

SameSite cookie: thiết lập cookie phiên với SameSite=Lax (hoặc Strict nếu không cần cross-site navigation). SameSite=Lax sẽ ngăn cookie được gửi cho hầu hết các request cross-site tự động (ví dụ form auto-submit, image), giảm mạnh vector CSRF phổ biến.

CSRF token (synchronizer token pattern): server sinh token ngẫu nhiên gắn vào form/phiên và kiểm tra token khi nhận request thay đổi trạng thái. Token phải là bí mật, không đoán được và được xác minh server-side. Đây là biện pháp truyền thống, phù hợp cho form HTML truyền thống.

Kiểm tra Origin/Referer: đối với các POST/PUT/DELETE, kiểm tra header Origin hoặc Referer để xác nhận request đến từ domain hợp lệ. Nếu header không tồn tại hoặc không đúng, từ chối. Phương pháp này đơn giản và hiệu quả với hầu hết trình duyệt hiện đại.

Double-submit cookie hoặc custom header: khi dùng cookie, có thể dùng double-submit (token trong cookie + token trong body/header) để xác minh; hoặc yêu cầu custom header cho request AJAX (trình duyệt không cho trang khác set custom header mà không preflight CORS).

Thiết kế API an toàn: sử dụng bearer token (Authorization header) hoặc JWT lưu ở bộ nhớ (không cookie) cho API; khi token không lưu trong cookie, CSRF truyền thống khó hoặc không thể khai thác.

Reauthentication/2FA cho hành động nhạy cảm: yêu cầu nhập lại mật khẩu, OTP cho các hành động quan trọng như chuyển tiền.

Fix XSS: nếu ứng dụng có XSS, attacker có thể đọc CSRF token và phá mọi biện pháp CSRF, vì vậy XSS cần được ưu tiên sửa trước.

Logging & monitoring: phát hiện request bất thường hoặc hàng loạt request cross-site, alert để phát hiện tấn công sớm.

Nguyên nhân chính khiến CSRF giảm phổ biến và không còn xuất hiện trong OWASP Top 10:2021

(1) Sự phổ biến của các framework bảo mật

Hầu hết các framework web hiện đại đã tích hợp cơ chế bảo vệ CSRF theo mặc định. Ví dụ Spring Security, Django, Ruby on Rails, Laravel đều tự động sinh và validate CSRF token cho form POST. Do đó developer thường không cần triển khai thủ công; nhiều ứng dụng mới vì

thể mặc định đã an toàn khỏi CSRF. Việc tích hợp này làm giảm tần suất lỗi CSRF trong kho mã hiện đại và trong dữ liệu phân tích của các tổ chức bảo mật.

(2) Bảo vệ ở cấp độ trình duyệt

Các trình duyệt hiện đại đã đưa ra các chính sách bảo mật ngăn chặn nhiều kịch bản CSRF. Đặc biệt, thuộc tính SameSite cho cookie (chrome 80+ triển khai từ đầu 2020) khiến cookie được đánh dấu SameSite=Lax theo mặc định trong nhiều trường hợp, ngăn cookie gửi trong đa số request cross-site tự động. Ngoài ra, các chính sách như CORS, Referrer-Policy và CSP giúp hạn chế thông tin leak và giới hạn khả năng gửi và đọc request cross-origin. Những thay đổi này làm giảm đáng kể bề mặt tấn công CSRF.

(3) Thay đổi kiến trúc ứng dụng web

Sự chuyển dịch sang SPA (Single Page Applications) dùng RESTful APIs cùng với xác thực thông qua Bearer token hoặc JWT (thường gửi trong Authorization header thay vì cookie) làm cho CSRF truyền thống ít hiệu quả. Khi token được truyền trong header hoặc lưu trữ ngoài cookie (ví dụ trong localStorage, sessionStorage), trình duyệt không tự động gửi token đó cho requests đến domain khác, vì vậy CSRF kiểu cũ không còn áp dụng. Đồng thời, việc áp dụng chính sách CORS chặt chẽ giúp kiểm soát ai được thực hiện request AJAX.

(4) Phạm vi tấn công bị thu hẹp và khó khai thác hơn

CSRF chỉ có hiệu quả với các request thay đổi trạng thái (POST/PUT/DELETE), yêu cầu nạn nhân đang đăng nhập và thường phải có interaction (ví dụ click link). CSRF không cho phép kẻ tấn công đọc dữ liệu trả về do same-origin policy — attacker chỉ có thể gây hành động, không thể lấy kết quả trực tiếp. Hơn nữa, detection và logging có khả năng phát hiện các hành vi bất thường, làm cho việc khai thác CSRF ít hấp dẫn hơn so với các lỗi như XSS hay Injection.

(5) Dữ liệu thống kê OWASP và cộng đồng cho thấy giảm tần suất

Khi OWASP cập nhật Top 10, họ dựa trên dữ liệu từ nhiều nguồn SAST/DAST và khảo sát thực tế. Các phân tích này cho thấy tần suất CSRF giảm mạnh, trong khi các rủi ro khác (Broken Access Control, Insecure Design, Software Integrity, SSRF...) nổi lên. Vì vậy CSRF bị loại khỏi Top 10:2021 không phải vì hết nguy hiểm hoàn toàn, mà vì tần suất và mức độ rủi ro tương đối đã giảm so với các mục khác.

Những hạn chế còn tồn tại: vì sao CSRF vẫn cần được chú ý

Mặc dù mức độ phổ biến đã giảm, CSRF vẫn đáng lưu tâm trong một số trường hợp: ứng dụng legacy chưa cập nhật framework, cấu hình framework sai (bật/tắt protection), hệ thống custom authentication không dùng token tiêu chuẩn, hoặc các dự án nội bộ/legacy

nơi không áp dụng SameSite hay CSRF token. Trong những môi trường này, CSRF vẫn có thể gây hậu quả nghiêm trọng, đặc biệt khi kết hợp với social engineering.

Kết luận (tóm tắt)

CSRF là một kỹ thuật tấn công dựa trên việc lợi dụng cookie/session mà trình duyệt gửi tự động. Cách phòng thanh toán hiệu quả bao gồm: thiết lập SameSite cookie, dùng CSRF token, kiểm tra Origin/Referer, dùng bearer token cho API, và yêu cầu reauthentication cho thao tác nhạy cảm. CSRF đã giảm phổ biến trong thực tế nhờ framework tích hợp bảo vệ, cơ chế trình duyệt (SameSite) và sự thay đổi sang kiến trúc API/JWT; vì vậy nó không còn xuất hiện trong OWASP Top 10:2021. Tuy nhiên, CSRF vẫn tồn tại ở các ứng dụng lỗi thời, cấu hình sai hoặc custom auth, nên không thể hoàn toàn bỏ qua trong kiểm thử bảo mật.