BỘ CÔNG THƯƠNG

TRƯỜNG ĐẠI HỌC ĐIỆN LỰC

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO CHUYÊN ĐỀ HỌC PHẦN

PHÁT TRIỂN PHẦN MỀM WEB AN TOÀN

ĐỀ TÀI: PHÂN TÍCH LỖ HỔNG MASS ASSIGNMENT TRONG API USER ROLE VÀ KHẮC PHỤC BẰNG WHITELIST FIELDS

Giảng viên hướng dẫn	**: TS. CẤN ĐỨC ĐIỆP**
Sinh viên thực hiện	**: ĐÀO DUY QUYỀN**
	: TRẦN GIA THÀNH
	: NGUYỄN HOÀNG THANH TÙNG
Ngành	**: CÔNG NGHỆ THÔNG TIN**
Chuyên ngành	**: CÔNG NGHỆ PHẦN MỀM**
Lớp	**: D17CNPM6**
Khóa	**: 2022 – 2027**

Hà Nội, 09 tháng 12 năm 2025

PHIẾU CHẤM ĐIỂM

STT	Họ và tên sinh viên	Nội dung thực hiện	Điểm	Chữ ký
1	**Đào Duy Quyền** 22810310231			
2	**Trần Gia Thành** 22810310238			
3	**Nguyễn Hoàng Thanh Tùng** 22810310248			

Họ và tên giảng viên	Chữ ký	Ghi chú
Giảng viên chấm 1:		
Giảng viên chấm 2:		

MỤC LỤC

LỜI MỞ ĐẦU 1
CHƯƠNG 1: TÌM HIỂU VỀ AN TOÀN VÀ BẢO MẬT WEB 2
1.1. Tổng quan về an toàn và bảo mật web 2
1.1.1. Khái niệm 2
1.1.2. Tầm quan trọng của việc bảo mật web 3
1.2. Nguyên tắc cơ bản của bảo mật web 3
1.2.1. Mô hình CIA (Confidentiality – Integrity - Availability) 4
1.2.2. Nguyên tắc Least Privilege (Đặc quyền tối thiểu) 5
1.2.3. Nguyên tắc Defense in Depth (Phòng thủ nhiều lớp) 6
1.3. Các loại mối đe dọa chính trên web 7
1.3.1. Cross-Site Scripting (XSS) 7
1.3.2. Cross-Site Request Forgery (CSRF) 8
1.3.3. SQL Injection / NoSQL Injection 9
1.3.4. Mass Assignment 10
1.3.5. Tấn công mạng (Network Attack) 10
1.3.6. Ddos và Bot Traffic 11
1.4. Các chuẩn và công cụ bảo mật web 12
1.4.1. OWASP Top 10 12
1.4.2. HTTPS/TLS 12
1.4.3. Một số công cụ kiểm thử bảo mật phổ biến 13
1.5. Thách thức và xu hướng bảo mật web hiện nay 13
CHƯƠNG 2: TỔNG QUAN VỀ ĐỀ TÀI 15
2.1 Bối cảnh 15
2.2. Lí do chọn đề tài 16
2.3. Mục tiêu của đề tài 17

2.4. Phạm vi và đối tượng nghiên cứu 18
2.5. Ý nghĩa thực tiễn của đề tài 19
CHƯƠNG 3: DEMO LỖ HỔNG MASS ASSIGNMENT TRONG HỆ THỐNG THUÊ PHÒNG TRỌ 20
3.1. Tấn công vào API đăng ký. 20
3.2. Tấn công nâng quyền người dùng (Privilege Escalation) 21
3.3. Tấn công nâng số dư tài khoản (Balance Escalation) 22
CHƯƠNG 4: GIẢI PHÁP PHÒNG TRÁNH LỖ HỔNG MASS ASSIGNMENT TRONG HỆ THỐNG THUÊ PHÒNG TRỌ 24
4.1. Giới thiệu 24
4.2. Nguyên tắc phòng tránh Mass Assignment 25
4.2.1. Không tin tưởng bất kỳ dữ liệu nào từ phía client 25
4.2.2. Áp dụng cơ chế Whitelist Field (lọc danh sách trường cho phép) 25
4.2.3. Dùng thư viện validate (Joi, class-validator, …) 26
4.2.4. Kết hợp phân quyền role-based (requireRoles) 27
4.2.5 Chống XSS lấy Refresh Token 27
KẾT LUẬN 29
TÀI LIỆU THAM KHẢO 30

DANH MỤC HÌNH ẢNH

Hình 1.1. Bảo mật web 2
Hình 1.2. Mô hình CIA 4
Hình 1.3. Nguyên tắc đặc quyền tối thiểu 5
Hình 1.4.: Phòng thủ sâu 6
Hình 1.5. Tấn công XSS 7
Hình 1.6. Tấn công CSRF 8
Hình 1.7. Tấn công SQL Injection 9
Hình 1.8. Tấn công mạng 10
Hình 2.1. Lỗ hổng Mass Assignment 16
Hình 3.1. Minh họa API đăng ký User 20
Hình 3.2. Minh họa API đăng ký Hacker tấn công 20
Hình 3.3. Hàm đăng ký người dùng không an toàn (Mass Assignment) 21
Hình 3.4. Cấu trúc User trên server với các trường nhạy cảm 21
Hình 3.5. Hacker sửa trường nhạy cảm 22
Hình 3.6. Hàm UpdateUser không an toàn 22
Hình 3.7. Hàm cập nhật người dùng không an toàn 23
Hình 3.8. Hacker sửa số dư tài khoản 23
Hình 4.1. Cập nhật thông tin người dùng không kiểm soát trường 25
Hình 4.2. Nhận dữ liệu người dùng giới hạn trường 25
Hình 4.3. Hàm cập nhật người dùng đã xử lý bảo mật 25
Hình 4.4. Hàm đăng ký được xử lý bảo mật 26
Hình 4.5. Dùng thư viện validate 26
Hình 4.6. Phân quyền cho web 27

LỜI CẢM ƠN

Trên thực tế, không có sự thành công nào mà không gắn liền với những sự hỗ trợ, sự giúp đỡ dù ít hay nhiều, dù là trực tiếp hay gián tiếp của người khác. Trong suốt thời gian từ khi bắt đầu học tập ở giảng đường Đại học đã đến nay, em đã nhận được rất nhiều sự quan tâm, giúp đỡ của thầy cô, gia đình và bạn bè. Với lòng biết ơn sâu sắc nhất, em xin gửi đến thầy cô ở Khoa Công nghệ thông tin - trường Đại Học Điện Lực đã cùng với tri thức và tâm huyết của mình để truyền đạt vốn kiến thức quý báu cho chúng em trong suốt thời gian học tập tại trường.

Nhóm em xin chân thành cảm ơn thầy **Cấn Đức Điệp** đã hướng dẫn chúng em qua những buổi học trên lớp, thảo luận về môn học. Trong thời gian được học tập và thực hành dưới sự hướng dẫn của thầy, chúng em không những thu được rất nhiều kiến thức bổ ích, mà còn được truyền sự say mê và thích thú đối với bộ môn "***Phát triển phần mềm web an toàn***". Nếu không có những lời hướng dẫn, dạy bảo của thầy thì chúng em nghĩ báo cáo này rất khó có thể hoàn thành được.

Mặc dù đã rất cố gắng hoàn thiện báo cáo với tất cả sự nỗ lực, tuy nhiên, do bước đầu đi vào thực tế, tìm hiểu và xây dựng báo cáo và dự án trong thời gian có hạn, và kiến thức còn hạn chế, nhiều bỡ ngỡ chắc chắn sẽ không thể tránh khỏi những thiếu sót. Chúng em rất mong nhận được sự quan tâm, thông cảm và những đóng góp quý báu của các thầy cô và các bạn để báo cáo này được hoàn thiện hơn.

Một lần nữa, chúng em xin chân thành cảm ơn và luôn mong nhận được sự đóng góp của mọi người.

Trân trọng!

LỜI MỞ ĐẦU

Trong bối cảnh các ứng dụng web hiện đại liên tục mở rộng về quy mô và tính năng, vấn đề bảo mật trở thành yếu tố then chốt để đảm bảo an toàn thông tin và tính toàn vẹn hệ thống. Một trong những lỗ hổng thường bị xem nhẹ nhưng gây hậu quả nghiêm trọng là Mass Assignment Vulnerability – lỗ hổng xuất phát từ việc hệ thống tự động gán (bind) toàn bộ dữ liệu đầu vào của người dùng vào các thuộc tính của mô hình (model) mà không có cơ chế kiểm soát. Khi API thiếu bộ lọc trường hợp lệ, kẻ tấn công có thể chèn thêm các trường nhạy cảm vào JSON request, điển hình như “role”: “ADMIN”, từ đó chiếm quyền truy cập, leo thang đặc quyền và gây mất kiểm soát toàn bộ hệ thống.

Đề tài này tập trung trình bày cách khai thác lỗ hổng Mass Assignment thông qua một mô phỏng đơn giản: người dùng gửi JSON chứa trường role = "ADMIN" trong quá trình đăng ký hoặc cập nhật thông tin. Tiếp đó, đề tài phân tích nguyên nhân, điều kiện phát sinh, và quan trọng hơn là đưa ra giải pháp khắc phục thông qua Whitelist Fields, tức chỉ cho phép những thuộc tính an toàn được ghi vào cơ sở dữ liệu, kết hợp với các lớp kiểm tra và xác thực phía server.

Thông qua việc thực hành khai thác và khắc phục, sinh viên có thể hiểu sâu hơn về cách hoạt động của các cơ chế ORM/ODM, middleware binding trong backend và các rủi ro xuất phát từ việc xử lý dữ liệu đầu vào không an toàn. Đề tài không chỉ mang tính minh họa mà còn hướng đến mục tiêu giúp người phát triển xây dựng hệ thống web an toàn, hạn chế tối đa khả năng kẻ tấn công lợi dụng để leo thang đặc quyền trên môi trường thực tế.

CHƯƠNG 1: TÌM HIỂU VỀ AN TOÀN VÀ BẢO MẬT WEB

1.1. Tổng quan về an toàn và bảo mật web

Bảo mật web (Web Security) là tập hợp các phương pháp, kỹ thuật và biện pháp nhằm bảo vệ các ứng dụng web, dữ liệu và hệ thống liên quan khỏi các mối đe dọa tấn công từ bên ngoài cũng như sai sót từ người dùng nội bộ.

Hình 1.1. Bảo mật web

Trong bối cảnh các dịch vụ trực tuyến phát triển mạnh mẽ—từ thương mại điện tử, thanh toán trực tuyến cho đến hệ thống đặt phòng, truy xuất nguồn gốc hay các API công cộng—việc đảm bảo an toàn cho ứng dụng web trở thành yêu cầu bắt buộc để duy trì sự tin cậy và tính ổn định của toàn hệ thống.

1.1.1. Khái niệm

Bảo mật web là quá trình thiết kế, triển khai và vận hành các cơ chế bảo vệ nhằm: Ngăn chặn truy cập trái phép, bảo vệ dữ liệu nhạy cảm, đảm bảo hệ thống hoạt động đúng chức năng, giảm thiểu và phát hiện kịp thời các hành vi tấn công.

Một hệ thống web an toàn không chỉ dựa vào phần mềm, mà còn bao gồm cấu hình máy chủ, chính sách truy cập, mã nguồn backend, cơ chế xác thực người dùng, giao thức truyền dữ liệu và các lớp bảo vệ theo nhiều tầng.

1.1.2. Tầm quan trọng của việc bảo mật web

Sự quan trọng của bảo mật web thể hiện qua các yếu tố sau:

- Bảo vệ dữ liệu người dùng: Các ứng dụng hiện nay lưu trữ thông tin cá nhân (PII), dữ liệu tài chính, thông tin đặt phòng, thanh toán. Việc lộ lọt có thể gây thiệt hại lớn cho cả người dùng và doanh nghiệp.
- Duy trì uy tín và thương hiệu: Một hệ thống bị hack hoặc rò rỉ dữ liệu có thể đánh mất sự tin tưởng của khách hàng, gây ảnh hưởng nặng nề tới uy tín doanh nghiệp.
- Đảm bảo tính liên tục của dịch vụ: Các cuộc tấn công từ chối dịch vụ (DDoS) hoặc phá hoại backend có thể khiến hệ thống ngừng hoạt động, gây gián đoạn dịch vụ.
- Đảm bảo tuân thủ pháp lý: Nhiều quốc gia yêu cầu hệ thống xử lý dữ liệu cá nhân phải đáp ứng quy định về bảo mật (GDPR, ISO 27001, …). Hệ thống không đáp ứng có thể bị phạt.
- Ngăn chặn gian lận và thao túng dữ liệu: Ví dụ trong hệ thống thuê phòng trọ, hacker có thể thay đổi giá phòng, sửa ngày đặt, xóa tài khoản, sửa bài viết, … nếu backend không bảo mật tốt.

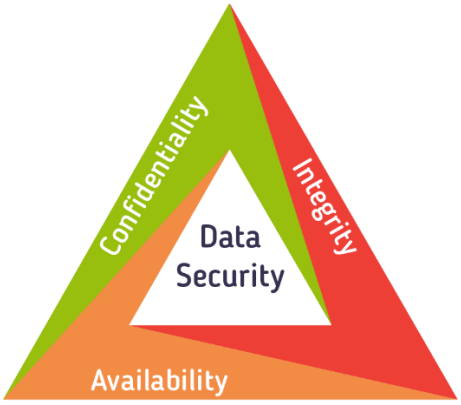
1.2. Nguyên tắc cơ bản của bảo mật web

Để xây dựng và vận hành một hệ thống web an toàn, việc nắm vững các nguyên tắc cốt lõi của bảo mật thông tin là điều bắt buộc.

Các nguyên tắc này đóng vai trò nền tảng cho mọi giải pháp, từ thiết kế kiến trúc đến triển khai mã nguồn, cấu hình hệ thống và đánh giá rủi ro. Trong đó, mô hình CIA và hai nguyên tắc Least Privilege và Defense in Depth luôn được xem là trọng tâm trong bảo mật web.

Nguyên tắc “Least Privilege” và “Defense in Depth”.

1.2.1. Mô hình CIA (Confidentiality – Integrity - Availability)



Hình 1.2. Mô hình CIA

Mô hình CIA là tiêu chuẩn kinh điển dùng để đánh giá và xây dựng các hệ thống bảo mật. Ba thành phần của CIA gồm:

1.2.1.1. Confidentiality (Tính bảo mật – Giữ bí mật dữ liệu)

Confidentiality đảm bảo rằng dữ liệu chỉ được truy cập bởi những người hoặc dịch vụ có quyền.

Các biện pháp thường dùng:

- Sử dụng HTTPS/TLS để mã hóa dữ liệu khi truyền.
- Hash và salt mật khẩu (BCrypt, Argon2).
- Áp dụng xác thực mạnh (JWT, OAuth2, MFA).

Phân quyền truy cập tài nguyên (RBAC – Role-Based Access Control).

Ví dụ trong hệ thống đặt phòng: Dữ liệu thẻ thanh toán, email, số điện thoại khách hàng phải được lưu trữ và truyền tải an toàn, không để lộ trong log hay response.

1.2.1.2. Integrity (Tính toàn vẹn – Không bị sửa đổi trái phép)

Integrity đảm bảo dữ liệu không bị thay đổi, xóa hoặc chèn sửa trái phép bởi hacker hoặc người dùng không được cấp quyền.

Các cơ chế hỗ trợ:

- Kiểm tra chữ ký số hoặc hash (SHA-256).

- Tách quyền đọc/ghi trong database.
- Ghi nhật ký (logging) để có thể truy vết thay đổi.
- Ràng buộc dữ liệu phía backend (validation, schema).

Ví dụ trong hệ thống: Booking của khách không thể bị thay đổi giá tiền hoặc ngày lưu trú trừ khi nhân viên được phân quyền thực hiện.

1.2.1.3. Availability (Tính sẵn sàng – Hệ thống luôn hoạt động)

Availability đảm bảo hệ thống luôn hoạt động ổn định, không bị gián đoạn dù chịu tấn công hoặc lỗi phần cứng.

Các biện pháp:

- Chống tấn công DDoS.
- Sử dụng rate limit cho API nhạy cảm (login, booking).
- Backup dữ liệu định kỳ.
- Hệ thống server có dự phòng (redundancy).

Ví dụ: API thanh toán hoặc API tạo booking không được phép "nghẽn" hoặc sập khi nhiều người dùng truy cập cùng lúc.

1.2.2. Nguyên tắc Least Privilege (Đặc quyền tối thiểu)

Hình 1.3. Nguyên tắc đặc quyền tối thiểu

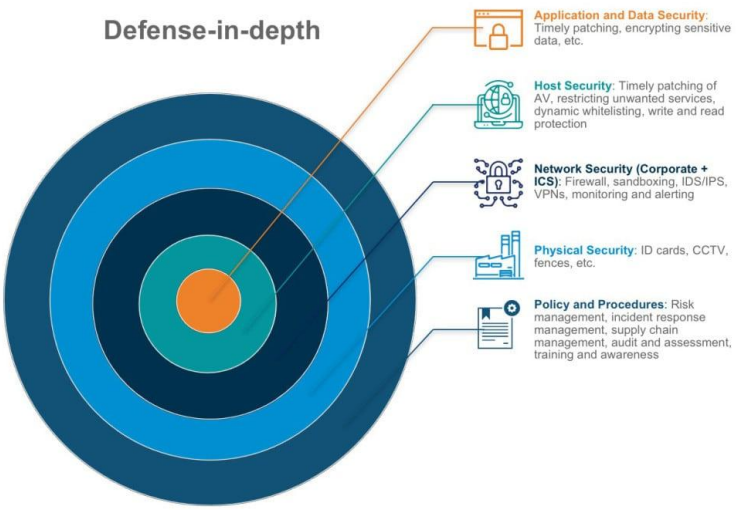
Nguyên tắc đặc quyền tối thiểu yêu cầu mỗi người dùng, dịch vụ hoặc tiến trình chỉ được cấp quyền đủ để thực hiện nhiệm vụ, không hơn.

Cách áp dụng trong phát triển web:

- Tài khoản database chỉ có quyền CRUD đúng bảng cần thiết.
- Chủ trọ chỉ được xem và quản lý các bài đăng của chính họ, không chỉnh sửa được bài đăng hoặc xem giao dịch của chủ trọ khác.
- Người thuê chỉ được chỉnh sửa thông tin cá nhân, không có quyền tác động đến bài đăng của người khác hoặc dữ liệu quản trị.
- API public không thể ghi, chỉ có thể đọc.

Refresh token chỉ được lưu ở cookie HttpOnly, không chia sẻ với frontend. Lợi ích là giảm thiểu thiệt hại khi tài khoản hoặc API key bị lộ.

1.2.3. Nguyên tắc Defense in Depth (Phòng thủ nhiều lớp)



Hình 1.4.: Phòng thủ sâu

Defense in Depth nghĩa là không dựa vào một lớp bảo mật duy nhất, mà xây dựng **nhiều tầng bảo vệ** để giảm rủi ro khi một lớp bị xâm nhập. Các lớp phòng thủ thường gặp:

- Lớp 1: Bảo mật giao thức — dùng HTTPS, chặn HTTP downgrade

- Lớp 2: Bảo mật ứng dụng — validation input, chống XSS, chống CSRF
- Lớp 3: Bảo mật API — rate limit, phân quyền, token
- Lớp 4: Bảo mật server — firewall, CORS, IP whitelist
- Lớp 5: Bảo mật dữ liệu — mã hóa, backup, phân quyền bảng
- Lớp 6: Giám sát và cảnh báo — logging, cảnh báo hành vi bất thường

1.3. Các loại mối đe dọa chính trên web

Hệ thống web luôn là mục tiêu của nhiều loại tấn công khác nhau, từ những cuộc tấn công kỹ thuật cao cho đến các phương thức lợi dụng yếu tố con người.

Trong thực tế, các mối đe dọa này có thể gây ra hậu quả nghiêm trọng như lộ lọt dữ liệu cá nhân, chiếm quyền tài khoản, truy cập trái phép hệ thống hoặc làm gián đoạn hoạt động kinh doanh. Dưới đây là các nhóm mối đe dọa phổ biến và nguy hiểm nhất đối với các ứng dụng web hiện nay.

1.3.1. Cross-Site Scripting (XSS)

Hình 1.5. Tấn công XSS

XSS là kiểu tấn công cho phép hacker chèn mã JavaScript độc hại vào trang web hợp pháp. Khi người dùng truy cập, đoạn script đó sẽ được chạy trong trình duyệt và có thể:

- Đánh cắp cookie hoặc token đăng nhập
- Chuyển hướng sang trang giả mạo

- Gửi yêu cầu thay người dùng
- Thay đổi giao diện hoặc chèn nội dung độc hại.

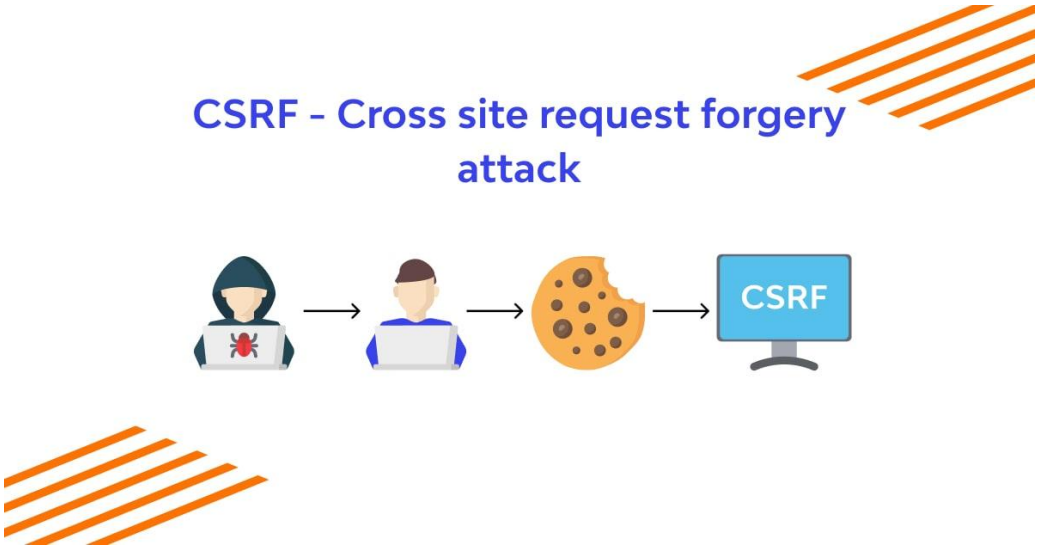
Nguyên nhân

- Không lọc hoặc escape dữ liệu đầu vào/đầu ra.
- Cho phép render HTML từ dữ liệu người dùng (ví dụ: đánh giá, comment).

Hậu quả

- Người dùng bị chiếm tài khoản.
- Hệ thống mất uy tín.
- Có thể trở thành nền tảng phát tán mã độc.

1.3.2. Cross-Site Request Forgery (CSRF)



Hình 1.6. Tấn công CSRF

CSRF là kiểu tấn công trong đó hacker lừa người dùng đã đăng nhập gửi yêu cầu ngoài ý muốn lên ứng dụng web. Tấn công xảy ra khi:

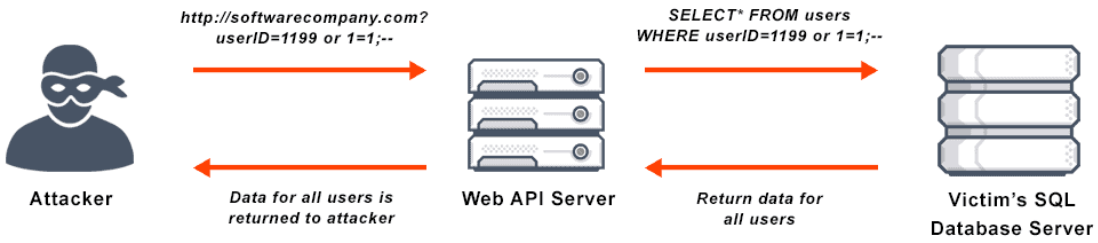
- Người dùng đang login
- Hệ thống dùng cookie để xác thực
- Hacker gửi một request giả mạo từ website khác.

Hậu quả

- Thay đổi mật khẩu người dùng
- Xóa hoặc tạo booking mới
- Thực hiện thanh toán không mong muốn.

Nguyên nhân: API nhạy cảm không yêu cầu token CSRF hoặc không dùng xác thực dựa header (Bearer Token).

1.3.3. SQL Injection / NoSQL Injection



Hình 1.7. Tấn công SQL Injection

SQL Injection (với MySQL/PostgreSQL) và NoSQL Injection (với MongoDB) cho phép hacker chèn câu lệnh truy vấn độc hại để:

- Đọc dữ liệu nhạy cảm
- Sửa hoặc xóa dữ liệu
- Bypass xác thực.

Nguyên nhân

- Ghép chuỗi để tạo query (string concatenation),
- API không kiểm tra loại dữ liệu đầu vào,
- Sử dụng $where, $regex không kiểm soát (đối với MongoDB).

Hậu quả

- Lộ toàn bộ thông tin khách hàng, booking, thanh toán,
- Hacker chiếm quyền admin,

- Xóa toàn bộ database.

1.3.4. Mass Assignment

Mass Assignment là lỗ hổng xảy ra khi backend cho phép người dùng gửi bất kỳ trường nào lên API, và các trường này được gán trực tiếp vào model/database.

Nguy hiểm

- Người dùng tự nâng quyền (role: "ADMIN")
- Tự chỉnh sửa trạng thái booking (status: "PAID")
- Bypass logic kinh doanh.

Hậu quả

- Nâng quyền trái phép: Hacker tự gán role: "ADMIN" hoặc quyền cao hơn.
- Thay đổi dữ liệu quan trọng: Tự chỉnh trạng thái booking (PAID, CANCELLED) hoặc giá tiền.
- Ghi đè trường nhạy cảm: Như isDeleted, verified, createdBy.
- Phá vỡ logic hệ thống: Gây sai lệch dữ liệu, thất thoát doanh thu.

1.3.5. Tấn công mạng (Network Attack)

Hình 1.8. Tấn công mạng

Bao gồm:

- Man-in-the-Middle (MITM): hacker chặn và sửa dữ liệu giữa client và server.
- Sniffing: nghe lén dữ liệu không mã hóa.
- DNS Spoofing: chuyển hướng người dùng đến trang giả mạo.

Nguyên nhân:

- Không dùng HTTPS,
- Không kiểm soát certificate,
- Cấu hình domain sai.

Hậu quả:

- Đánh cắp thông tin: Cookie, token đăng nhập, dữ liệu cá nhân.
- Giả mạo phiên làm việc: Chiếm quyền người dùng.
- Sửa đổi dữ liệu khi truyền: Thay đổi số tiền thanh toán, ID booking.
- Chuyển hướng đến trang giả mạo: Người dùng bị lừa nhập thông tin.
- Phát tán mã độc: Chèn script độc vào response.

1.3.6. Ddos và Bot Traffic

Tấn công từ chối dịch vụ bằng cách gửi lượng request khổng lồ vào hệ thống.

- Hậu quả:
    - API không phản hồi
    - Không ai có thể tạo booking hoặc thanh toán
    - Tổn thất doanh thu.
- Biện pháp:
    - Rate limit
    - WAF (Web Application Firewall)
    - CDN như Cloudflare.

1.4. Các chuẩn và công cụ bảo mật web

Trong quá trình phát triển và vận hành hệ thống web, việc tuân thủ các chuẩn bảo mật và sử dụng các công cụ kiểm thử chuyên dụng giúp phát hiện sớm lỗ hổng, giảm thiểu rủi ro và nâng cao độ tin cậy của ứng dụng. Các chuẩn này đóng vai trò định hướng, còn công cụ hỗ trợ đánh giá và mô phỏng các phương thức tấn công phổ biến.

1.4.1. OWASP Top 10

OWASP Top 10 là bộ tiêu chuẩn quan trọng nhất trong lĩnh vực bảo mật web, được cập nhật định kỳ bởi Open Web Application Security Project. Danh sách này tổng hợp 10 nhóm rủi ro bảo mật phổ biến và nguy hiểm nhất:

- A01 – Broken Access Control: Lỗi phân quyền.
- A02 – Cryptographic Failures: Lỗi mã hóa.
- A03 – Injection: SQL/NoSQL/XSS Injection.
- A04 – Insecure Design: Thiết kế bảo mật kém.
- A05 – Security Misconfiguration: Cấu hình sai (CORS, server, DB).
- A06 – Vulnerable and Outdated Components: Dùng thư viện lỗi thời.
- A07 – Identification and Authentication Failures: Lỗi xác thực.
- A08 – Software and Data Integrity Failures: Dữ liệu không kiểm soát.
- A09 – Security Logging and Monitoring Failures: Thiếu log/giám sát.
- A10 – Server-Side Request Forgery (SSRF).

OWASP đóng vai trò khung tham chiếu để phát triển web an toàn, giúp đội ngũ dev, QA và security đánh giá toàn diện theo chuẩn quốc tế.

1.4.2. HTTPS/TLS

HTTPS/TLS là chuẩn mã hóa bắt buộc giúp bảo vệ dữ liệu giữa client và server khỏi bị đánh cắp hoặc sửa đổi. Đây là lớp phòng thủ quan trọng đối với bất kỳ hệ thống nào có đăng nhập, thanh toán hoặc API public.

Lợi ích chính: Bảo vệ khỏi tấn công Man-in-the-Middle (MITM), mã hóa dữ liệu nhạy cảm: token, cookie, thông tin người dùng, tăng uy tín website và khả năng SEO, bắt buộc khi tích hợp cổng thanh toán (như VNPay).

1.4.3. Một số công cụ kiểm thử bảo mật phổ biến

Để đánh giá mức độ an toàn của ứng dụng, nhiều công cụ pentest và scanning được sử dụng rộng rãi:

- Burp Suite: Công cụ mạnh nhất cho pentest web, hỗ trợ intercept request, khám phá lỗ hổng, fuzzing API, dùng để kiểm thử XSS, CSRF, injection, SSRF…
- OWASP ZAP: Công cụ miễn phí mã nguồn mở, tự động quét lỗ hổng (active/passive scanning), thích hợp cho kiểm thử CI/CD hoặc local,
- Postman / Thunder Client: Không chuyên về bảo mật nhưng quan trọng để, test API, kiểm tra phản hồi khi nhập input bất thường, mô phỏng tấn công Mass Assignment hoặc NoSQL Injection.
- Nikto: Scanner kiểm tra cấu hình server, phát hiện lỗi header bảo mật, file nguy hiểm.
- Nmap: Quét cổng, quét dịch vụ đang mở, dùng để phát hiện điểm yếu dịch vụ mạng.
- Snyk / npm audit: Kiểm tra thư viện node modules lỗi thời hoặc có lỗ hổng.
- Security Headers Analyzez: Kiểm tra các header bảo mật: CSP, X-Frame-Options, HSTS, …

1.5. Thách thức và xu hướng bảo mật web hiện nay

Các ứng dụng hiện đại thường bao gồm nhiều thành phần phân tán, dẫn đến việc quản lý cấu hình, phân quyền, bảo mật API và kiểm soát truy cập trở nên phức tạp hơn. Bên cạnh đó, khối lượng dữ liệu người dùng ngày càng tăng khiến yêu cầu về mã hóa, tuân thủ pháp lý và quản trị rủi ro trở nên nghiêm ngặt hơn, đặc biệt với các hệ thống phức tạp như thương mại điện tử, đặt phòng hay truy xuất nguồn gốc.

Xu hướng tấn công cũng ngày càng tinh vi và tự động hóa hơn. Hacker sử dụng bot, công cụ quét tự động, thậm chí trí tuệ nhân tạo để dò tìm lỗ hổng, khai thác API yếu hoặc thực hiện tấn công brute-force với tốc độ lớn. Các kỹ thuật như XSS, CSRF, SSRF,

và Mass Assignment vẫn phổ biến, trong khi các hình thức mới như API Abuse, tấn công vào cloud configuration, chiếm quyền session hoặc khai thác chuỗi cung ứng (supply-chain attack) trở nên đáng lo ngại. Điều này khiến việc bảo vệ ứng dụng không chỉ dừng lại ở tầng mã nguồn mà còn cần giám sát toàn diện từ mạng, hạ tầng đến thư viện bên thứ ba.

Trước sự phát triển nhanh của công nghệ, xu hướng bảo mật hiện nay tập trung vào Zero Trust Architecture, Defense in Depth, và sử dụng AI/ML để phát hiện hành vi bất thường. Các doanh nghiệp ưu tiên triển khai HTTPS/TLS toàn diện, áp dụng xác thực mạnh (MFA, OAuth2), giám sát liên tục (SIEM, logging), và tuân thủ các tiêu chuẩn như OWASP Top 10.

CHƯƠNG 2: TỔNG QUAN VỀ ĐỀ TÀI

2.1 Bối cảnh

Trong bối cảnh công nghệ số phát triển nhanh chóng, các ứng dụng web ngày càng trở thành thành phần cốt lõi trong nhiều lĩnh vực như thương mại điện tử, du lịch, nền tảng thuê phòng trọ, hệ thống quản lý tài khoản hay các giải pháp quản trị doanh nghiệp. Đặc biệt, kiến trúc RESTful API được ưa chuộng nhờ tính linh hoạt, khả năng mở rộng tốt và sự tách biệt rõ ràng giữa frontend và backend.

Tuy nhiên, chính sự phổ biến của RESTful API khiến chúng dễ trở thành mục tiêu tấn công của hacker. Việc truyền tải dữ liệu qua JSON cùng cơ chế ánh xạ dữ liệu giữa client và server khiến hệ thống dễ bị ảnh hưởng nếu các lớp xác thực hoặc kiểm soát đầu vào tồn tại lỗ hổng. Chỉ một sai sót nhỏ cũng có thể khiến toàn bộ API rơi vào tình trạng mất an toàn.

Cùng với sự phát triển của các framework như Node.js/Express, Mongoose, Laravel, Rails hay Django, cơ chế “tự động bind dữ liệu” từ request vào model đã giúp việc lập trình trở nên nhanh và thuận tiện hơn. Tuy nhiên, sự tiện lợi này lại tiềm ẩn nguy cơ lớn về bảo mật. Khi backend không giới hạn rõ ràng những trường được phép cập nhật, kẻ tấn công có thể dễ dàng thêm các thuộc tính độc hại vào JSON request.

Ví dụ điển hình là trường hợp hacker tự chèn "role": "ADMIN" vào dữ liệu đăng ký tài khoản. Nếu server không kiểm soát kỹ, hệ thống sẽ vô tình cấp quyền quản trị cho người dùng không hợp lệ – dẫn tới hậu quả nghiêm trọng.

Trong bối cảnh các hình thức tấn công web ngày càng tinh vi, lỗ hổng Mass Assignment (gán dữ liệu hàng loạt mà không kiểm soát) trở thành nguyên nhân phổ biến dẫn tới leo thang đặc quyền, chỉnh sửa trái phép thông tin nhạy cảm hoặc chiếm quyền toàn bộ hệ thống. Dù không nằm trong danh sách OWASP Top 10 hiện nay, lỗi này vẫn xuất hiện với tần suất rất cao trong các dự án thực tế, nhất là các hệ thống do sinh viên hoặc lập trình viên mới xây dựng khi bước kiểm tra dữ liệu thường bị bỏ qua.

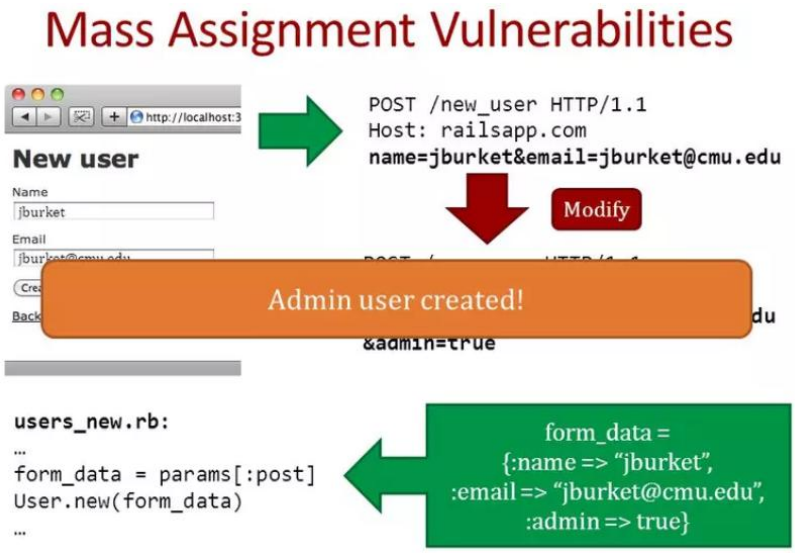
Do đó, đề tài đặt trong bối cảnh nhằm làm rõ mức độ nguy hiểm của Mass Assignment trong API hiện đại, mô phỏng cách hacker khai thác lỗ hổng, đồng thời đề xuất các phương pháp phòng tránh hiệu quả. Thực tế khảo sát cho thấy lỗ hổng này phổ

biến hơn nhiều người tưởng, và việc nghiên cứu sâu sẽ mang lại giá trị lớn cho lập trình viên backend khi xây dựng các hệ thống web an toàn.

2.2. Lí do chọn đề tài

Lý do đầu tiên để lựa chọn đề tài xuất phát từ một thực tế phổ biến trong cộng đồng lập trình: đa số nhà phát triển API thường chú trọng vào việc xây dựng tính năng và đảm bảo API hoạt động đúng yêu cầu, nhưng lại xem nhẹ vấn đề bảo mật.

Trong quá trình triển khai ứng dụng, đặc biệt khi chịu áp lực về thời gian, nhiều developer vô tình để server tiếp nhận toàn bộ dữ liệu trong JSON request mà không giới hạn hay kiểm soát từng trường một — tạo ra lỗ hổng để hacker lợi dụng.



Hình 2.1. Lỗ hổng Mass Assignment

Mặc dù Mass Assignment là một lỗ hổng tương đối dễ hiểu, nhưng lại rất khó phát hiện trong thực tế, bởi người dùng bình thường không bao giờ gửi những trường nhạy cảm như "role", "permissions" hay "isAdmin".

Tuy nhiên, chỉ cần một hacker có hiểu biết kỹ thuật cơ bản, họ có thể sử dụng Postman, Curl hoặc BurpSuite để tự thêm các trường này vào request. Nhiều hệ thống, chỉ với một dòng JSON độc hại, đã vô tình tạo tài khoản có đặc quyền quản trị mà không trải qua bất kỳ bước xác thực nâng cao nào.

Việc chọn đề tài còn nhằm giúp người học nhìn lại cách viết code của bản thân. Không ít sinh viên khi làm bài tập lớn, đồ án môn học hay đồ án tốt nghiệp đã vô tình tạo ra hệ thống chứa lỗ hổng nghiêm trọng mà không hề nhận ra.

Khi tiến hành demo tấn công Mass Assignment, sinh viên có thể trực tiếp quan sát mức độ nguy hiểm khi backend thiếu cơ chế bảo vệ thích hợp. Điều này không chỉ mang ý nghĩa lý thuyết mà còn có giá trị thực tiễn đối với con đường phát triển nghề nghiệp sau này.

Bên cạnh đó, chủ đề này được lựa chọn vì tính ứng dụng cao trong các hệ thống web hiện đại. Những chức năng quen thuộc như đăng ký tài khoản, cập nhật thông tin cá nhân, chỉnh sửa sản phẩm hay thay đổi cấu hình đều có nguy cơ bị khai thác nếu không kiểm soát dữ liệu đầu vào cẩn thận.

Thông qua đề tài, sinh viên có cơ hội rèn luyện tư duy lập trình an toàn, hiểu rõ tầm quan trọng của việc giới hạn trường dữ liệu, áp dụng whitelist, kiểm tra quyền truy cập và xác thực dữ liệu trước khi ghi vào cơ sở dữ liệu — từ đó góp phần xây dựng các hệ thống web an toàn và đáng tin cậy hơn.

2.3. Mục tiêu của đề tài

Mục tiêu đầu tiên của đề tài là xây dựng một nền tảng lý thuyết vững chắc liên quan đến lỗ hổng Mass Assignment. Nội dung này bao gồm việc làm rõ khái niệm, nguyên nhân hình thành, cũng như cơ chế hoạt động của lỗi dựa trên quá trình binding dữ liệu trong các framework backend. Đề tài phân tích vì sao việc tự động ánh xạ dữ liệu JSON vào object lại trở thành điểm yếu, đặc biệt khi các trường nhạy cảm không được kiểm soát chặt chẽ.

Mục tiêu thứ hai là phát triển một hệ thống mô phỏng tấn công thực tế. Thông qua các ví dụ minh họa, báo cáo trình bày cách hacker gửi JSON độc hại chứa những trường như "role": "ADMIN" hoặc "isActive": true để chiếm quyền điều khiển hệ thống. Những mô phỏng này giúp người học nhận thấy đây không chỉ là vấn đề lý thuyết, mà là kiểu tấn công hoàn toàn có thể xảy ra đối với bất kỳ API nào thiếu lớp bảo vệ phù hợp.

Mục tiêu tiếp theo của đề tài là xây dựng các biện pháp phòng tránh, trong đó trọng tâm là cơ chế Whitelist Fields — cho phép backend chỉ cập nhật những trường đã được

định nghĩa an toàn từ trước. Đồng thời, đề tài xem xét việc ứng dụng các thành phần kiến trúc an toàn như tầng DTO, middleware kiểm tra dữ liệu, cùng các thư viện validate như Joi, nhằm đảm bảo mọi thông tin truyền vào hệ thống đều được kiểm soát trước khi ghi xuống cơ sở dữ liệu.

Mục tiêu cuối cùng là tiến hành đánh giá và rút ra kết luận sau quá trình mô phỏng và thử nghiệm các giải pháp. Báo cáo sẽ phân tích mức độ hiệu quả của từng biện pháp, từ đó khẳng định tính cần thiết của whitelist và validation trong thực tiễn phát triển API. Đây cũng là căn cứ để nhấn mạnh giá trị ứng dụng của đề tài trong việc xây dựng các hệ thống web an toàn và đáng tin cậy.

2.4. Phạm vi và đối tượng nghiên cứu

Phạm vi của đề tài tập trung vào việc nghiên cứu lỗ hổng Mass Assignment trong môi trường backend API. Đề tài chỉ tập trung vào quy trình dữ liệu đi từ client đến server thông qua JSON request, cách dữ liệu được ánh xạ vào model trong database, và các kỹ thuật phòng chống phổ biến trong Node.js/Express kết hợp với MongoDB/Mongoose. Việc giới hạn phạm vi như vậy giúp đề tài tập trung vào các nội dung then chốt, tránh lan man sang những khía cạnh bảo mật khác.

Đối tượng nghiên cứu chính bao gồm:

- Dữ liệu JSON được gửi từ client
- Request body parser (Express middleware)
- Cơ chế ánh xạ thuộc tính trong Mongoose schemas
- Các trường nhạy cảm trong user model như roles, permissions, isVerified
- Quy trình cập nhật thông tin người dùng trong API
- Cách hacker lợi dụng API để gửi dữ liệu bất hợp pháp

Ngoài backend Node.js, đề tài cũng đề cập đến một số framework khác như Laravel hay Rails để so sánh mức độ nguy hiểm của Mass Assignment trong từng môi trường. Tuy nhiên, các framework này chỉ được dùng để minh họa chứ không phải trọng tâm nghiên cứu.

Đề tài không phân tích các loại tấn công khác như XSS, SQL Injection, CSRF hay SSRF, cũng không nghiên cứu các phương pháp xác thực nâng cao như OAuth2, JWT refresh rotation, hay bảo mật đa lớp. Việc giới hạn rõ ràng phạm vi giúp đảm bảo nội dung đi sâu đúng trọng tâm, phục vụ tốt cho mục tiêu bài học.

2.5. Ý nghĩa thực tiễn của đề tài

Ý nghĩa lớn nhất của đề tài nằm ở việc nâng cao nhận thức của người lập trình backend về tầm quan trọng của việc kiểm soát dữ liệu đầu vào. Mass Assignment tuy không "ồn ào" như SQL Injection hay XSS, nhưng mức độ nguy hiểm không hề kém cạnh, thậm chí có thể nghiêm trọng hơn trong nhiều tình huống thực tế. Thông qua việc mô phỏng tấn công và phân tích cách khắc phục, người lập trình hiểu rõ rằng bất kỳ API nào thiếu whitelist đều có thể trở thành điểm yếu dẫn đến chiếm quyền, sửa dữ liệu hoặc phá vỡ toàn bộ hệ thống.

Đề tài còn có ý nghĩa trong việc trang bị cho sinh viên tư duy lập trình an toàn (secure coding mindset). Thay vì chỉ viết code để "chạy được", người phát triển cần viết code an toàn, tuân thủ các chuẩn OWASP và đảm bảo dữ liệu nhạy cảm không thể bị người dùng can thiệp. Đây là yếu tố cực kỳ quan trọng khi bước vào môi trường làm việc chuyên nghiệp, nơi an toàn thông tin luôn là ưu tiên hàng đầu.

Về mặt ứng dụng thực tế, kiến thức từ đề tài có thể áp dụng trực tiếp vào các hệ thống phổ biến như nền tảng đăng tin thuê phòng trọ, hệ thống quản lý người dùng, cổng thanh toán, các website thương mại điện tử hoặc bất kỳ ứng dụng nào có phân quyền giữa người dùng và quản trị viên. Chỉ một sai sót nhỏ trong việc kiểm soát trường nhạy cảm cũng có thể dẫn đến việc người dùng tự nâng quyền admin, can thiệp bài đăng, thay đổi giá hoặc giả mạo giao dịch.

Cuối cùng, đề tài giúp người học xây dựng nền tảng vững chắc để nghiên cứu các chủ đề bảo mật nâng cao trong tương lai. Các kỹ thuật phòng tránh Mass Assignment cũng là bước đệm quan trọng để tìm hiểu sâu hơn về kiến trúc RBAC, Access Control List, API Hardening và các phương pháp bảo mật backend hiện đại, phục vụ tốt cho việc phát triển các hệ thống quy mô lớn và an toàn.

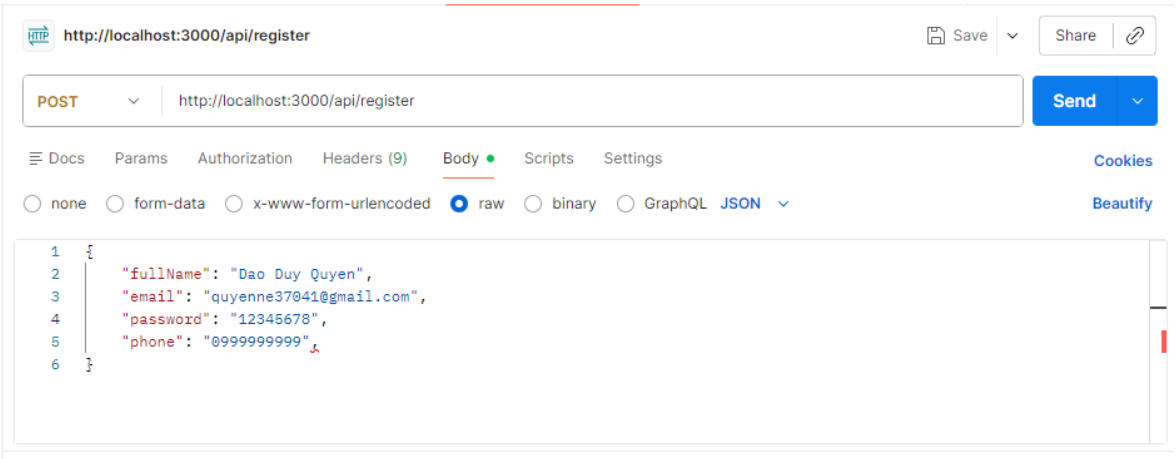
CHƯƠNG 3: DEMO LỖ HỔNG MASS ASSIGNMENT TRONG HỆ THỐNG THUÊ PHÒNG TRỌ

3.1. Tấn công vào API đăng ký.

Giả sử API đăng ký của bạn:

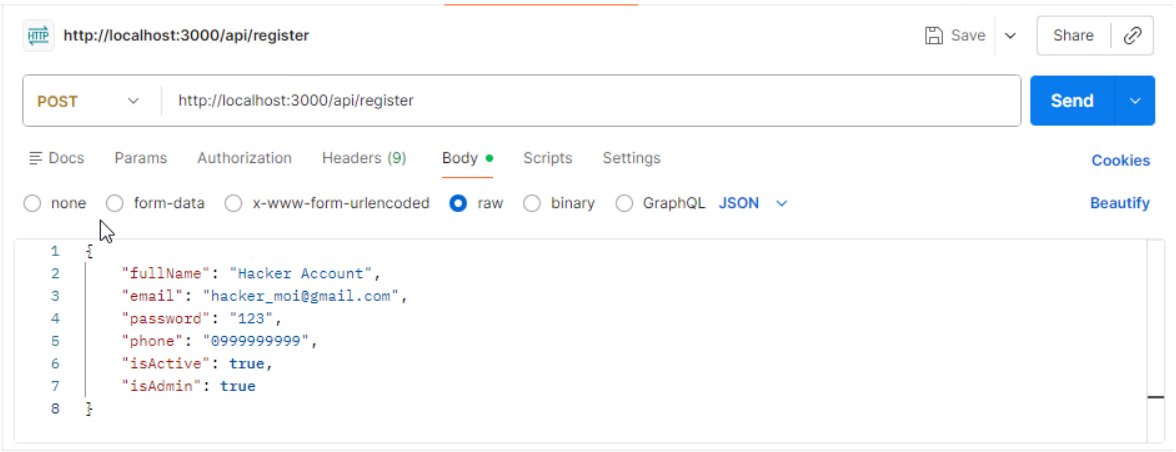
- POST /api/register
- ContentType:application/json

Frontend chỉ gửi:

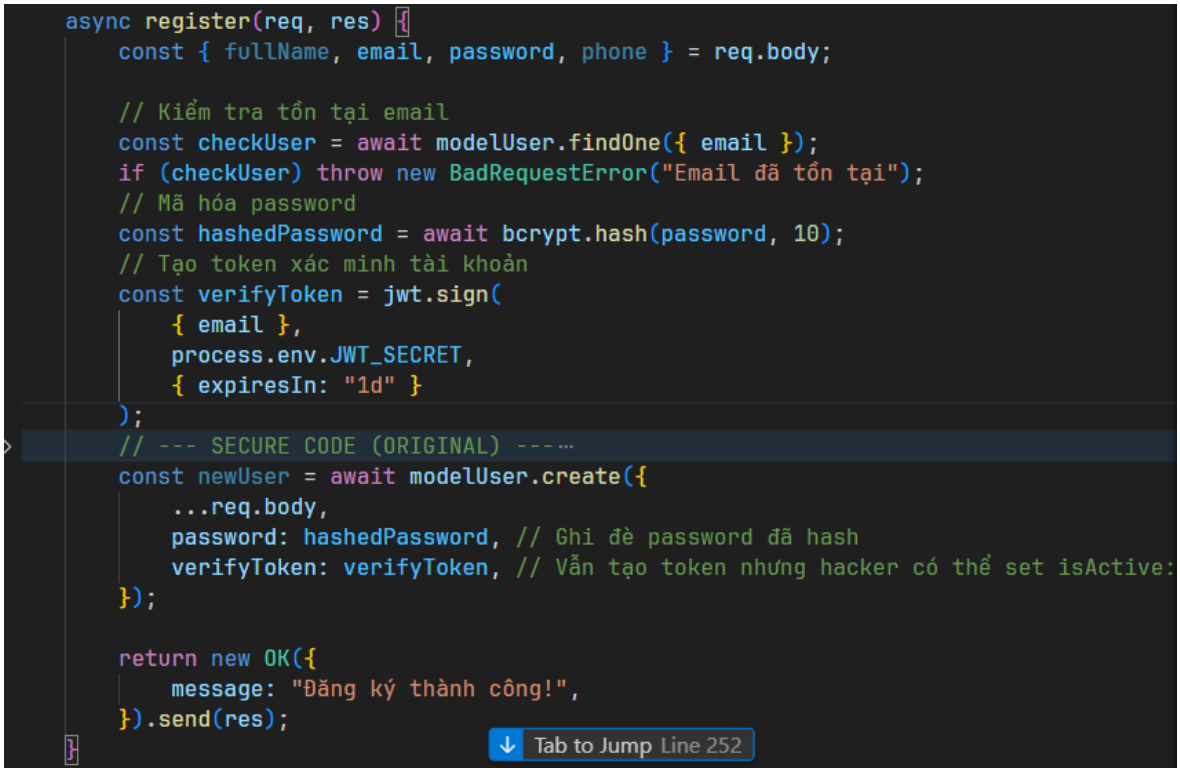


Hình 3.1. Minh họa API đăng ký User

Nhưng hacker dùng Postman gửi:



Hình 3.2. Minh họa API đăng ký Hacker tấn công



Hình 3.3. Hàm đăng ký người dùng không an toàn (Mass Assignment)

=> User mới sẽ có roles: ["ADMIN"] → hacker trở thành **ADMIN hệ thống**.

3.2. Tấn công nâng quyền người dùng (Privilege Escalation)

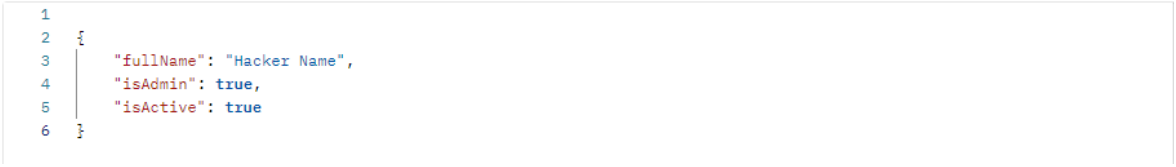
Mô tả

Trong hệ thống, mỗi User có roles xác định quyền hạn như: ADMIN, USER.

Giao diện frontend đăng ký chỉ có các trường:

Hình 3.4. Cấu trúc User trên server với các trường nhạy cảm

Nhưng hacker có thể sử dụng Postman và sửa trường isAdmin và isActive:



Hình 3.5. Hacker sửa trường nhạy cảm

Nếu backend đang viết code nguy hiểm như sau:

```
async updateUser(req, res) {
    const { id } = req.user;
    // const { fullName, phone, email, address, avatar } = req.body;
    // const user = await modelUser.findByIdAndUpdate(id, { fullName, phone, email,

    // VULNERABLE CODE FOR DEMO: Mass Assignment
    // Cho phép update tất cả các trường trong req.body mà không lọc
    const user = await modelUser.findByIdAndUpdate(id, req.body, { new: true });

    new OK({ message: 'Cập nhật thông tin thành công', metadata: user }).send(res);
}
```

Hình 3.6. Hàm UpdateUser không an toàn

Người dùng tạo mới sẽ trở thành **ADMIN**, có quyền truy cập toàn bộ hệ thống, quản lý phòng trọ, tạo sửa xóa bài đăng, xem toàn bộ giao dịch, người dùng.

Hậu quả:

- Hacker có full quyền hệ thống
- Kiểm soát toàn bộ phòng trọ và người dùng
- Thay đổi dữ liệu không giới hạn

3.3. Tấn công nâng số dư tài khoản (Balance Escalation)

Trong hệ thống, mỗi người dùng có trường balance để lưu số dư tài khoản. Bình thường số dư chỉ được thay đổi thông qua các quy trình hợp lệ như:

- Nạp tiền (qua ngân hàng / ví điện tử)
- Hoàn tiền đơn hàng
- Thanh toán dịch vụ

Tuy nhiên, nếu API cập nhật thông tin người dùng được viết không an toàn, hệ thống có thể cho phép người dùng tự ý sửa số dư của chính mình bằng cách lợi dụng lỗi Mass Assignment.

Frontend bình thường chỉ gửi các trường cho phép

```
async updateUser(req, res) {
    const { id } = req.user;
    // const { fullName, phone, email, address, avatar } = req.body;
    // const user = await modelUser.findByIdAndUpdate(id, { fullName, phone, email, a

    // VULNERABLE CODE FOR DEMO: Mass Assignment
    // Cho phép update tất cả các trường trong req.body mà không lọc
    const user = await modelUser.findByIdAndUpdate(id, req.body, { new: true });

    new OK({ message: 'Cập nhật thông tin thành công', metadata: user }).send(res);
}
```

Hình 3.7. Hàm cập nhật người dùng không an toàn

Nhưng hacker dùng Postman gửi thêm trường balance

```
{
    "fullName": "Hacker Name",
    "balance": 9999999999,
    "isActive": true
}
```

Hình 3.8. Hacker sửa số dư tài khoản

Hậu quả:

- Trường balance sẽ được cập nhật theo đúng giá trị hacker gửi
- Người dùng tự ý tăng số dư, không cần nạp tiền
- Phá vỡ toàn bộ logic tài chính của hệ thống

CHƯƠNG 4: GIẢI PHÁP PHÒNG TRÁNH LỖ HỔNG MASS ASSIGNMENT TRONG HỆ THỐNG THUÊ PHÒNG TRỌ

4.1. Giới thiệu

Trong các ứng dụng web hiện đại, đặc biệt là những hệ thống có nhiều nhóm người dùng như người thuê, chủ trọ, admin, cùng với các mức phân quyền khác nhau, lỗ hổng Mass Assignment được xem là một trong những rủi ro bảo mật phổ biến và nguy hiểm nhất. Lỗ hổng này xảy ra khi backend tự động ánh xạ (bind) toàn bộ dữ liệu từ phía client vào model trong cơ sở dữ liệu, mà không kiểm soát chặt chẽ các trường nhạy cảm mà người dùng không được phép thay đổi.

Đối với hệ thống thuê phòng trọ, việc bảo vệ dữ liệu là vô cùng quan trọng vì hệ thống chứa nhiều thông tin có giá trị cao như:

- Thông tin cá nhân của người dùng
- Lịch sử giao dịch và nạp tiền
- Dữ liệu bài đăng cho thuê phòng
- Giá thuê, trạng thái tin đăng, trạng thái duyệt tin
- Quyền quản trị tin đăng và tài khoản

Nếu không có biện pháp xử lý đúng cách, hacker hoặc người dùng có chủ đích xấu có thể lợi dụng lỗ hổng Mass Assignment để:

- Tự nâng cấp tài khoản thành admin hoặc chủ trọ
- Tự ý chỉnh sửa trạng thái bài đăng
- Thay đổi giá phòng, làm giả mức giá rất thấp để thu hút người thuê hoặc tạo gian lận
- Tự đánh dấu giao dịch là “đã thanh toán” dù không hề nạp tiền
- Chiếm quyền sở hữu bài đăng bằng cách sửa trường ownerId trong request
- Mở khóa hoặc xóa tài khoản người khác nếu backend không có whitelist

Các nguy cơ trên có thể dẫn đến sai lệch dữ liệu, mất doanh thu, thất thoát tài chính và nghiêm trọng hơn là mất kiểm soát toàn bộ hệ thống.

4.2. Nguyên tắc phòng tránh Mass Assignment

Để phòng tránh triệt để lỗ hổng Mass Assignment, cần tuân thủ các nguyên tắc bảo mật lõi sau:

4.2.1. Không tin tưởng bất kỳ dữ liệu nào từ phía client

Mọi dữ liệu mà frontend hoặc Postman gửi lên đều có thể bị thay đổi hoặc giả mạo. Backend phải luôn xác thực, kiểm soát và lựa chọn các trường được phép xử lý thay vì tin toàn bộ dữ liệu.

4.2.2. Áp dụng cơ chế Whitelist Field (lọc danh sách trường cho phép)

Thay vì sử dụng các cách nguy hiểm như:

```
const user = await modelUser.findByIdAndUpdate(id, req.body, { new: true });

new OK({ message: 'Cập nhật thông tin thành công', metadata: user }).send(res);
```

Hình 4.1. Cập nhật thông tin người dùng không kiểm soát trường

Chỉ nên xây dựng Object từ các trường an toàn:

```
const { id } = req.user;
const { fullName, phone, email, address, avatar } = req.body;
```

Hình 4.2. Nhận dữ liệu người dùng giới hạn trường

```
async updateUser(req, res) {
    const { id } = req.user;
    const { fullName, phone, email, address, avatar } = req.body;

    const user = await modelUser.findByIdAndUpdate(id, { fullName, phone, email, address, avatar }, { new: true })

    new OK({ message: 'Cập nhật thông tin thành công', metadata: user }).send(res);
}
```

Hình 4.3. Hàm cập nhật người dùng đã xử lý bảo mật

```
async register(req, res) {
    const { fullName, email, password, phone } = req.body;
    // Kiểm tra tồn tại email
    const checkUser = await modelUser.findOne({ email });
    if (checkUser) throw new BadRequestError("Email đã tồn tại");
    // Mã hóa password
    const hashedPassword = await bcrypt.hash(password, 10);
    // Tạo token xác minh tài khoản
    const verifyToken = jwt.sign(
        { email },
        process.env.JWT_SECRET,
        { expiresIn: "1d" }
    );
    // --- SECURE CODE (ORIGINAL) ---
    const newUser = await modelUser.create({
        fullName,
        email,
        password: hashedPassword,
        phone,
        isActive: false,
        verifyToken: verifyToken,
    });

    // --- VULNERABLE CODE (DEMO MASS ASSIGNMENT) ----…
    return new OK({
        message: "Đăng ký thành công!",
    }).send(res);
}
```

Hình 4.4. Hàm đăng ký được xử lý bảo mật

Điều này giúp loại bỏ mọi field nguy hiểm mà hacker có thể gửi lên như:

- isActive
- isAdmin
- Balance

“Kẻ tấn công có thể gửi thêm isAdmin: true, nhưng server sẽ bỏ qua vì code chỉ lấy các field fullName, phone, email, address, avatar để cập nhật.”

4.2.3. Dùng thư viện validate (Joi, class-validator, …)

Với dự án lớn hơn, có thể dùng thư viện validate/DTO để vừa kiểm tra dữ liệu, vừa tự động loại bỏ các field không mong muốn.

```
const Joi = require('joi');

const userValidation = {
    // Schema cho chức năng update user
    // CHỈ cho phép các trường cụ thể được update (Whitelist)
    updateUser: Joi.object({
        fullName: Joi.string().min(2).max(50),
        phone: Joi.string().pattern(/^[0-9]{10,11}$/),
        email: Joi.string().email(),
        address: Joi.string().max(255),
        avatar: Joi.string().uri(),

        // Tuyệt đối KHÔNG khai báo isAdmin, balance, isActive ở đây
        // Nếu client truyền lên, Joi sẽ báo lỗi hoặc lọc bỏ (tuỳ config)
    }),

    // Schema cho chức năng register
    register: Joi.object({
        fullName: Joi.string().required().min(2).max(50),
        email: Joi.string().email().required(),
        password: Joi.string().required().min(6),
        phone: Joi.string().pattern(/^[0-9]{10,11}$/).required(),
        // Không có isActive, isAdmin
    })
};

module.exports = userValidation;
```

Hình 4.5. Dùng thư viện validate

4.2.4. Kết hợp phân quyền role-based (requireRoles)

Chỉ có admin mới đc phép cập nhật các trường như isAdmin, isActive

```
const { BadUserRequestError } = require('../core/error.response');
const modelUser = require('../models/users.model');

// Middleware kiểm tra quyền hạn dựa trên các trường được gửi lên
// role: 'admin' | 'user' - Role bắt buộc để được update các restrictedFields
// restrictedFields: array - Các trường nhạy cảm cần bảo vệ
const grantAccess = (role, restrictedFields = []) => {
    return async (req, res, next) => {
        try {
            // Lấy danh sách các trường user đang muốn update
            const updateFields = Object.keys(req.body);

            // Tìm các trường nhạy cảm có trong request
            const sensitiveFields = updateFields.filter(field => restrictedFields.includes(field));

            // Nếu không có trường nhạy cảm nào -> Cho qua
            if (sensitiveFields.length === 0) {
                return next();
            }
            // Nếu có trường nhạy cảm -> Kiểm tra Role
            const { id } = req.user; // req.user có từ middleware authUser
            const user = await modelUser.findById(id);

            if (!user) {
                throw new BadUserRequestError("Không tìm thấy người dùng");
            }
            // Logic kiểm tra quyền
            const isAdmin = user.isAdmin;

            if (role === 'admin' && !isAdmin) {
                throw new BadUserRequestError(`Bạn không có quyền cập nhật các trường: ${sensitiveFields.join(', ')}`);
            }

            next();
        } catch (error) {
            next(error);
        }
    };
};

module.exports = grantAccess;
```

Hình 4.6. Phân quyền cho web

4.2.5 Chống XSS lấy Refresh Token

XSS (Cross-Site Scripting) là tấn công chèn mã JS độc vào website để đánh cắp dữ liệu trình duyệt của nạn nhân. Nếu token được lưu trong localStorage hoặc JS-accessible cookie, hacker có thể lấy token bằng script.

Cơ chế Access Token + Refresh Token

Trong hệ thống xác thực người dùng sử dụng JWT (JSON Web Token), một mô hình bảo mật phổ biến là dùng **hai loại token**:

- **Access Token:** Token ngắn hạn, được gửi kèm mỗi request API (qua header Authorization: Bearer <token>). Ví dụ: thời gian sống 15 phút.
- **Refresh Token:** Token dài hạn, dùng để xin cấp lại Access Token mới khi nó hết hạn. Refresh Token không trả về FE trong body, mà lưu dưới dạng cookie

HTTP-Only để JavaScript không thể truy cập → tránh nguy cơ bị đánh cắp bởi tấn công XSS.

Mục đích bảo mật:

- Hạn chế rủi ro khi token bị lộ: Nếu chỉ dùng 1 token duy nhất sống dài (ví dụ 30 ngày), khi hacker lấy được token, hắn có thể sử dụng liên tục đến khi token hết hạn.
- Nếu chỉ dùng 1 token duy nhất sống dài (ví dụ 30 ngày), khi hacker lấy được token, hắn có thể sử dụng liên tục đến khi token hết hạn.

Refresh Token được lưu trong cơ sở dữ liệu (ví dụ: mảng user.refreshTokens[]) và có thể bị xóa hiệu lực bất kỳ lúc nào.

Nhờ đó hệ thống có thể:

- Logout toàn bộ thiết bị của user
- Vô hiệu hóa token nghi ngờ bị đánh cắp
- Reset thông tin bảo mật (ví dụ user đổi mật khẩu) Nếu chỉ dùng 1 token dài hạn → không thể thu hồi.

KẾT LUẬN

Qua quá trình demo, chúng em đã chứng minh rằng chỉ với một request JSON đơn giản, một người dùng bình thường hoàn toàn có thể lợi dụng lỗ hổng Mass Assignment để leo thang đặc quyền, truy cập vào các API quản trị, chỉnh sửa hoặc xóa các dữ liệu quan trọng như bài đăng phòng trọ, trạng thái duyệt tin, giao dịch nạp tiền hay thông tin tài khoản. Điều này đặc biệt nguy hiểm đối với các hệ thống backend sử dụng ORM/ODM hoặc các framework hỗ trợ binding tự động như Mongoose hay Sequelize, nơi lập trình viên rất dễ vô tình bỏ sót bước kiểm soát dữ liệu đầu vào.

Từ các thử nghiệm thực tế, nhóm nhận thấy rằng kỹ thuật field whitelist (chỉ cho phép backend cập nhật những trường hợp lệ đã được định nghĩa trước) là giải pháp hiệu quả, rõ ràng và ít ảnh hưởng đến logic nghiệp vụ nhất. Khi áp dụng whitelist, kể cả khi attacker gửi thêm các trường nhạy cảm như role, isAdmin, isVerified, balance, ownerId, hay trạng thái bài đăng (status), hệ thống vẫn an toàn vì backend sẽ tự động loại bỏ toàn bộ những thuộc tính không được phép cập nhật.

Bên cạnh đó, nhóm còn kết hợp thêm các cơ chế bảo mật khác như phân quyền theo vai trò, middleware kiểm tra quyền truy cập, validate dữ liệu bằng Joi/Yup/Zod, và logging request để theo dõi hành vi bất thường. Những biện pháp này góp phần tăng mức độ an toàn và đảm bảo mỗi thao tác cập nhật đều tuân thủ đúng quy tắc của hệ thống.

Kết quả đạt được từ đề tài không chỉ giúp nhóm hiểu rõ hơn về cơ chế hoạt động của lỗ hổng Mass Assignment, mà còn củng cố tư duy quan trọng trong phát triển backend: “Không bao giờ tin dữ liệu từ client”. Kiến thức này có tính ứng dụng rất cao trong các hệ thống thực tế, đặc biệt là các API có thao tác CRUD, hệ thống quản lý người dùng, xử lý giao dịch và phân quyền đa cấp như nền tảng thuê phòng trọ.

Tóm lại, đề tài đã góp phần nâng cao nhận thức về bảo mật API, chỉ ra những rủi ro tiềm ẩn khi gán dữ liệu không kiểm soát và đưa ra giải pháp thiết thực, hiệu quả để phòng tránh. Trong bối cảnh các cuộc tấn công ngày càng tinh vi, việc tuân thủ nguyên tắc phát triển an toàn ngay từ đầu sẽ giúp hệ thống giảm thiểu rủi ro, đảm bảo tính bảo mật, toàn vẹn và uy tín của dữ liệu.

TÀI LIỆU THAM KHẢO

[1]. Bài giảng của thầy Cấn Đức Điệp môn học Phát triển phần mềm web an toàn.

[2]. Lỗ hổng Mass Assignment và cách phòng tránh (https://viblo.asia/p/lo-hong-mass-assignment-va-cach-phong-tranh-3Q75wn6BlWb).

[3]. OWASP Foundation. Mass Assignment Cheat Sheet.

(https://cheatsheetseries.owasp.org/cheatsheets/Mass_Assignment_Cheat_Sheet)

[4]. A10 Networks. (2020). What is Mass Assignment?

(https://www.a10networks.com/glossary/what-is-mass-assignment/)

[5]. Vaadata – Mass Assignment: Attacks and Security Tips.

(https://www.vaadata.com/blog/what-is-mass-assignment-attacks-and-security-tips/)