Bộ Giáo Dục Và Đào Tạo

Trường Đại Học Ngoại Ngữ - Tin Học Thành Phố Hồ Chí Minh

Khoa Công Nghệ Thông Tin

MÔN HỌC : PENTESTING

ĐỀ TÀI : PENTEST WEB APPLICATION THEO FRAMEWORK OWASP TOP 10

Giáo Viên Hướng Dẫn : **THS. PHẠM ĐÌNH THẮNG**

Thành Viên :

Phạm Hoàng Gia Bảo **MSSV: 22DH110298**

Tp. Hồ Chí Minh, ngày 01 tháng 04 năm 2025

MỤC LỤC

DANH MỤC BẢNG

CHƯƠNG 1: GIỚI THIỆU

1. Tổng quan về đề tài

1.1. Lý do chọn đề tài

Sự gia tăng các cuộc tấn công mạng nhắm vào ứng dụng web đã trở thành một vấn đề nghiêm trọng trong những năm gần đây. Các lỗ hổng bảo mật phổ biến như SQL Injection và Broken Access Control đã được tận dụng bởi các hacker để xâm phạm hệ thống và đánh cắp dữ liệu quan trọng. Theo OWASP, Broken Access Control đã trở thành lỗ hổng bảo mật nguy hiểm nhất trong năm 2021, cho phép kẻ tấn công truy cập trái phép vào các tài nguyên hệ thống mà không cần sự cho phép hợp lệ.

OWASP Top 10 cung cấp một khung phân loại lỗ hổng bảo mật toàn diện, được cập nhật định kỳ dựa trên dữ liệu thực tế từ hàng nghìn ứng dụng web trên toàn thế giới. Việc nghiên cứu và áp dụng các phương pháp kiểm thử chủ động dựa trên OWASP Top 10 là rất cần thiết, đặc biệt khi 78% ứng dụng web tại Việt Nam chưa tuân thủ các tiêu chuẩn bảo mật cơ bản.

1.2. Mục tiêu đề tài

Đề tài hướng đến các mục tiêu chính sau:

- Phân tích và kiểm thử các lỗ hổng bảo mật phổ biến trên website.
- Đánh giá rủi ro dựa trên mức độ nghiêm trọng của từng lỗ hổng.
- Đề xuất giải pháp khắc phục để tăng cường bảo mật.
- Rèn luyện kỹ năng sử dụng các công cụ kiểm thử như Burp Suite, Wireshark, Kali Linux,…

1.3. Giới hạn đề tài

Do thời gian và nguồn lực có hạn, đề tài tập trung vào:

- Kiểm thử trên một website mẫu (STA Hospital) với các tính năng cơ bản.
- Phân tích các lỗ hổng thuộc OWASP Top 10 2021.
- Sử dụng một số công cụ phổ biến như Burp Suite, Wireshark, Kali Linux.
- Không thực hiện tấn công trên các hệ thống thực tế không được phép.

2. **Đối tượng và phạm vi nghiên cứu**

Đối tượng nghiên cứu: Các lỗ hổng bảo mật trên web

- Broken Access Control
- Cryptographic Failures
- Injection
- Insecure Design
- Security Misconfiguration
- Indenfication and Authentication Failures
- Software and data integrity failures
- Security logging and monitoring failures

Phạm vi nghiên cứu:

- Môi trường kiểm thử: website mô phỏng STA Hospital
- Các công cụ sử dụng: Burp suite, Wireshark, Checkmarx
- Phương pháp: White Box Testing và Gray Box Testing

3. **Phương pháp nghiên cứu**

- Sử dụng công cụ tự động (Checkmarx, OWASP ZAP) và kiểm tra thủ công
- Thu thập thông tin
- Báo cáo và đề xuất
- Đánh giá rủi ro và đưa ra giải pháp khắc phục

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

1. Giới thiệu

1.1. Giới thiệu về Penetration Testing

Penetration Testing, hay còn gọi là kiểm thử thâm nhập, là một phương pháp đánh giá an ninh mạng bằng cách mô phỏng các cuộc tấn công vào hệ thống thông tin của một tổ chức. Quá trình này được thực hiện bởi các chuyên gia bảo mật, thường được gọi là "tin tặc có đạo đức" (ethical hackers), nhằm tìm ra các lỗ hổng bảo mật trước khi chúng bị khai thác bởi những kẻ tấn công thực sự. Kiểm thử thâm nhập có thể được thực hiện theo hai phương pháp chính: Blackbox và Whitebox. Trong

đó, Blackbox mô phỏng một cuộc tấn công từ một người không có kiến thức về hệ thống, còn Whitebox thực hiện với sự hiểu biết đầy đủ về hệ thống.

1.2. Mục tiêu của kiểm thử thâm nhập

Mục tiêu chính của kiểm thử thâm nhập là:

- Xác định và đánh giá các lỗ hổng bảo mật: Tìm ra các điểm yếu trong hệ thống, ứng dụng, hoặc cơ sở hạ tầng mạng mà có thể bị khai thác bởi kẻ tấn công.
- Đánh giá hiệu quả của các biện pháp bảo mật hiện có: Kiểm tra xem các hệ thống bảo vệ như tường lửa, hệ thống phát hiện xâm nhập (IDS/IPS) có hoạt động hiệu quả hay không.
- Cải thiện an ninh hệ thống: Cung cấp các khuyến nghị và giải pháp để khắc phục các lỗ hổng được phát hiện, giúp nâng cao mức độ an toàn cho hệ thống

1.3. Lợi ích của việc kiểm thử thâm nhập

Lợi ích của việc thực hiện kiểm thử thâm nhập bao gồm:

- Phát hiện và khắc phục lỗ hổng bảo mật: Trước khi chúng bị khai thác bởi kẻ tấn công, giúp ngăn chặn các cuộc tấn công tiềm ẩn.
- Tăng cường an ninh hệ thống: Giúp xác định và cải thiện các biện pháp bảo vệ hiện có, giảm thiểu rủi ro bị tấn công.
- Đảm bảo tuân thủ và giảm thiểu rủi ro: Hỗ trợ tổ chức trong việc tuân thủ các tiêu chuẩn bảo mật và giảm thiểu rủi ro về danh tiếng và tài chính

2. Phương pháp kiểm thử thâm nhập

2.1. Phân loại các phương pháp

Các phương pháp kiểm thử thâm nhập thường được phân loại dựa trên mức độ thông tin được cung cấp cho người kiểm thử và phạm vi của cuộc kiểm thử. Dưới đây là một số phương pháp phổ biến:

- Black Box (Kiểm thử hộp đen): Đây là phương pháp kiểm thử mà người kiểm thử không có bất kỳ thông tin nào về hệ thống. Họ sẽ tiếp cận hệ thống như một kẻ tấn công bên ngoài, không biết gì về cấu trúc và chi tiết bên trong hệ thống.

- White Box (Kiểm thử hộp trắng): Trong phương pháp này, người kiểm thử được cung cấp đầy đủ thông tin về hệ thống, bao gồm kiến trúc, mã nguồn, và thông tin mạng. Điều này giúp tiết kiệm thời gian và chi phí trong việc lập kế hoạch và trinh sát.
- Gray Box (Kiểm thử hộp xám): Đây là phương pháp kết hợp giữa hộp đen và hộp trắng. Người kiểm thử được cung cấp một số thông tin về hệ thống nhưng không đầy đủ, thường được sử dụng để mô phỏng một cuộc tấn công từ bên trong.

Pentest nội bộ và bên ngoài:

- Nội bộ: Đánh giá hệ thống nội bộ để xác định cách kẻ tấn công có thể di chuyển trong mạng.
- Bên ngoài: Đánh giá các hệ thống truy cập Internet để tìm lỗ hổng có thể bị khai thác từ bên ngoài

2.2. Ưu nhược điểm của các phương pháp kiểm thử

Black Box

Ưu điểm:

- Mô phỏng chính xác nhất các cuộc tấn công từ bên ngoài.
- Giúp đánh giá khả năng chống lại các cuộc tấn công mà không cần biết thông tin chi tiết về hệ thống.

Nhược điểm:

- Tốn nhiều thời gian và chi phí hơn do phải thu thập thông tin từ đầu.
- Có thể bỏ qua một số lỗ hổng nếu không có đủ thông tin.

White Box

Ưu điểm:

- Tiết kiệm thời gian và chi phí do đã có thông tin đầy đủ.
- Có thể phát hiện được nhiều lỗ hổng hơn nhờ kiến thức sâu về hệ thống.

Nhược điểm:

- Không mô phỏng được các cuộc tấn công từ bên ngoài một cách chính xác.

- Cần có sự hợp tác và cung cấp thông tin đầy đủ từ phía tổ chức.

Gray Box

Ưu điểm:

- Kết hợp được ưu điểm của cả Black Box và White Box.
- Có thể mô phỏng các cuộc tấn công từ bên trong một cách hiệu quả.

Nhược điểm:

- Cần có sự cân bằng giữa mức độ thông tin được cung cấp và thực tế của cuộc tấn công.
- Có thể không phù hợp với tất cả các loại hệ thống hoặc tổ chức.

Pentest Nội bộ và Bên ngoài

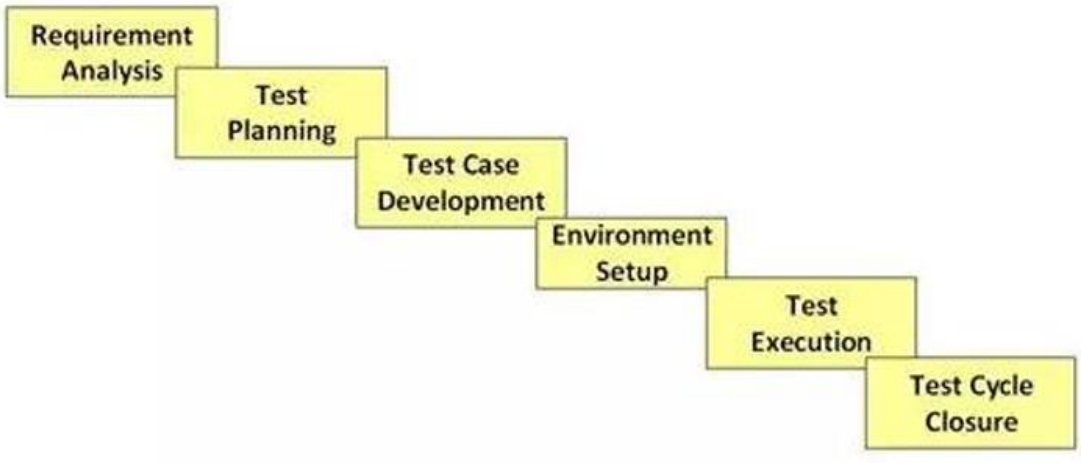
Ưu điểm:

- Cung cấp cái nhìn toàn diện về bảo mật của hệ thống từ cả bên trong và bên ngoài.
- Giúp xác định các lỗ hổng và rủi ro cụ thể cho từng môi trường.

Nhược điểm:

- Cần có nguồn lực và chuyên môn sâu để thực hiện cả hai loại kiểm thử.
- Có thể tốn nhiều thời gian và chi phí hơn nếu thực hiện riêng biệt.

3. Giai đoạn kiểm thử



3.1. Phân tích yêu cầu

Phân tích yêu cầu chức năng và phi chức năng từ khách hàng. Làm rõ yêu cầu mơ hồ và xác định môi trường kiểm thử.

3.2. Lập kế hoạch kiểm thử

Xây dựng kế hoạch kiểm thử, bao gồm phạm vi, hướng tiếp cận, quy trình, tài nguyên, nhân lực và lịch trình kiểm thử.

3.3. Phát triển kịch bản kiểm thử

Viết Test Case chi tiết với dữ liệu đầu vào, hành động, kết quả mong đợi và thực tế. Có thể tạo Test Script nếu sử dụng kiểm thử tự động.

3.4. Thiết lập môi trường kiểm thử

Thiết lập môi trường kiểm thử và thực hiện kiểm thử khói để kiểm tra tính ổn định của môi trường và sản phẩm.

3.5. Thực hiện kiểm thử

Thực hiện kiểm thử dựa trên Test Case. Log lỗi, gửi lỗi cho developers và kiểm tra lại sau khi fix. Đảm bảo tuân thủ tiến độ.

3.6. Kết thúc chu kỳ kiểm thử

Chuẩn bị báo cáo kiểm thử, tổng hợp kết quả, đánh giá tiêu chí hoàn thành và bàn giao sản phẩm.

3.7. Công cụ sử dụng

Các công cụ kiểm thử thâm nhập là các phần mềm được thiết kế để giúp các chuyên gia bảo mật kiểm tra và đánh giá các hệ thống, ứng dụng hoặc mạng máy tính để xác định các lỗ hổng bảo mật và rủi ro tiềm ẩn. Dưới đây là một số công cụ kiểm thử thâm nhập phổ biến được sử dụng rộng rãi trong cộng đồng bảo mật:

• **Burp Suite:** Burp Suite là một công cụ kiểm thử thâm nhập ứng dụng web rất mạnh mẽ và phổ biến. Nó cung cấp các chức năng proxy, spider, scanner, intruder, repeater và sequencer để kiểm tra các lỗ hổng bảo mật trong các ứng dụng web.

• **OWASP Zed Attack Proxy (ZAP):** ZAP là một công cụ mã nguồn mở được phát triển bởi cộng đồng OWASP. Nó cung cấp các tính năng proxy, scanner, fuzzer và một loạt các tiện ích để phát hiện và khắc phục lỗ hổng bảo mật trong ứng dụng web.

• **Nmap:** Nmap là một công cụ quét mạng mã nguồn mở được sử dụng để khám phá máy chủ, cổng mạng, và các dịch vụ chạy trên máy chủ. Nó cung cấp khả năng phát hiện các cổng mở và kiểm tra các dịch vụ đang chạy trên các máy chủ.

4. Checkmarx

Checkmarx là một nền tảng bảo mật ứng dụng hàng đầu (Application Security - AppSec), tập trung vào việc phát hiện và ngăn chặn các lỗ hổng bảo mật trong mã nguồn (SAST - Static Application Security Testing) và các thành phần phụ thuộc (SCA - Software Composition Analysis). Dưới đây là tổng quan lý thuyết về Checkmarx:

4.1. Tổng quan về Checkmarx

Checkmarx cung cấp các giải pháp bảo mật ứng dụng theo phương pháp "Shift Left" (tích hợp bảo mật sớm vào quy trình phát triển phần mềm - SDLC). Các sản phẩm chính:

- Checkmarx SAST: Phân tích tĩnh mã nguồn để tìm lỗ hổng.
- Checkmarx SCA: Quản lý rủi ro từ thư viện mã nguồn mở.
- Checkmarx DAST (Dynamic Application Security Testing): Kiểm tra ứng dụng đang chạy.
- Checkmarx IaC Security: Bảo mật cơ sở hạ tầng dưới dạng mã (Infrastructure as Code).

4.2. Checkmarx SAST (Static Application Security Testing)

Nguyên lý hoạt động

- Phân tích mã nguồn (source code) hoặc mã byte (bytecode) mà không cần chạy ứng dụng.
- Sử dụng mô hình dữ liệu dòng (data flow analysis) để theo dõi luồng dữ liệu từ điểm đầu vào (source) đến điểm nguy hiểm (sink).
- Phát hiện các lỗ hổng như SQL Injection, XSS, CSRF, RCE, v.v.

Ưu điểm

- Phát hiện sớm lỗ hổng trong giai đoạn viết mã.
- Hỗ trợ đa ngôn ngữ (Java, C#, JavaScript, Python, Go, Kotlin, v.v.).

- Tích hợp với các công cụ CI/CD (Jenkins, Azure DevOps, GitHub Actions).

Nhược điểm

- Có thể có dương tính giả (false positives) do phân tích tĩnh.
- Hiệu suất chậm với dự án lớn.

4.3. Checkmarx SCA (Software Composition Analysis)

Nguyên lý hoạt động

- Quét các thư viện mã nguồn mở (open-source) và thành phần phụ thuộc (dependencies) trong dự án.
- So sánh với cơ sở dữ liệu lỗ hổng (như CVE, NVD) để đánh giá rủi ro.
- Cảnh báo về giấy phép (license compliance).

Ưu điểm

- Phát hiện các lỗ hổng đã biết (ví dụ: Log4Shell, Heartbleed).
- Đánh giá mức độ nghiêm trọng dựa trên CVSS score.

4.4. Quy trình làm việc với Checkmarx

Tích hợp vào SDLC:Quét mã nguồn qua CLI hoặc plugin IDE.

Tự động hóa qua CI/CD pipeline.

Phân tích kết quả:

- Báo cáo chi tiết lỗ hổng (vị trí, mức độ, cách khắc phục).
- Phân loại theo tiêu chuẩn OWASP Top 10, CWE, PCI DSS.

Khắc phục:

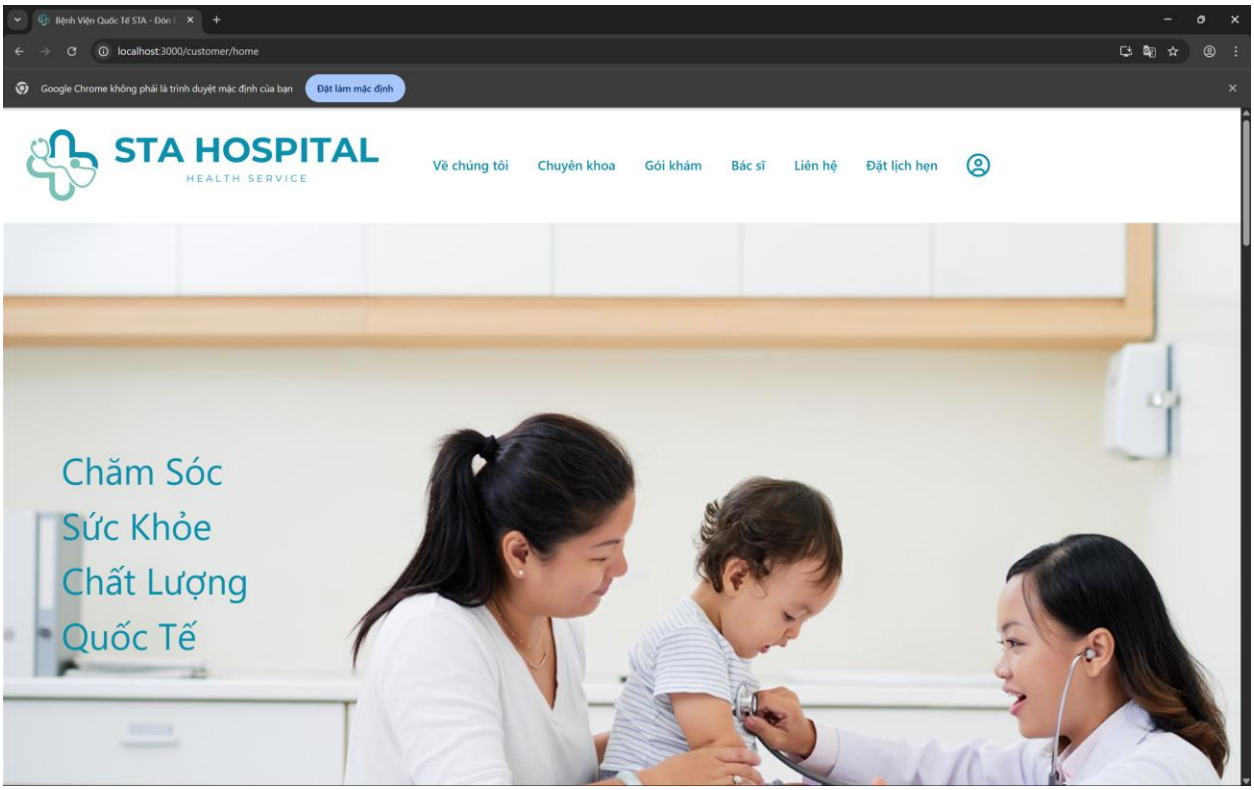
- Gợi ý sửa lỗi (remediation guidance).
- Theo dõi tiến độ sửa lỗi.

4.5. Ứng dụng thực tế

- DevSecOps: Checkmarx giúp tự động hóa bảo mật trong quy trình phát triển.
- Compliance: Đáp ứng các chuẩn bảo mật như GDPR, HIPAA, PCI DSS.

CHƯƠNG 3: TRIỂN KHAI

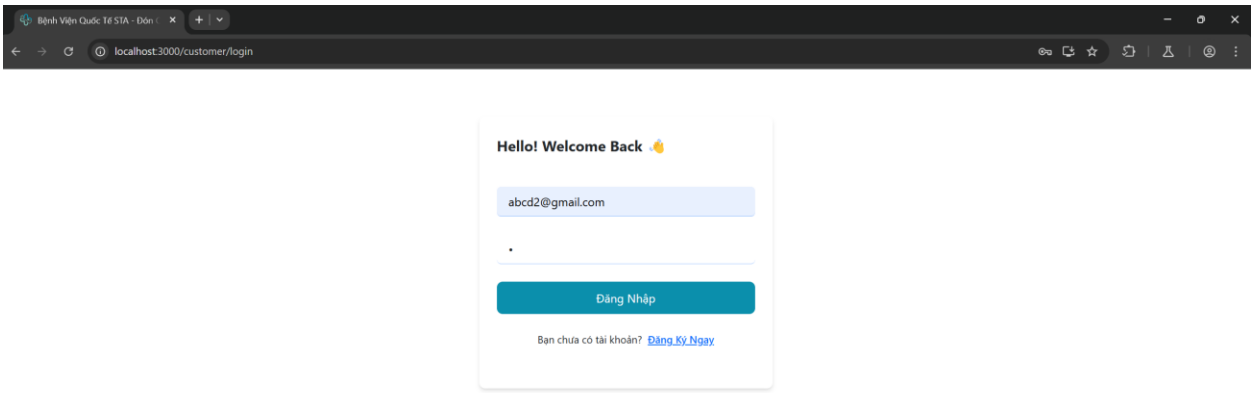
Web Hospital Service



Hình 1. Trang web STA HOSPITAL

A1: Broken Access Control

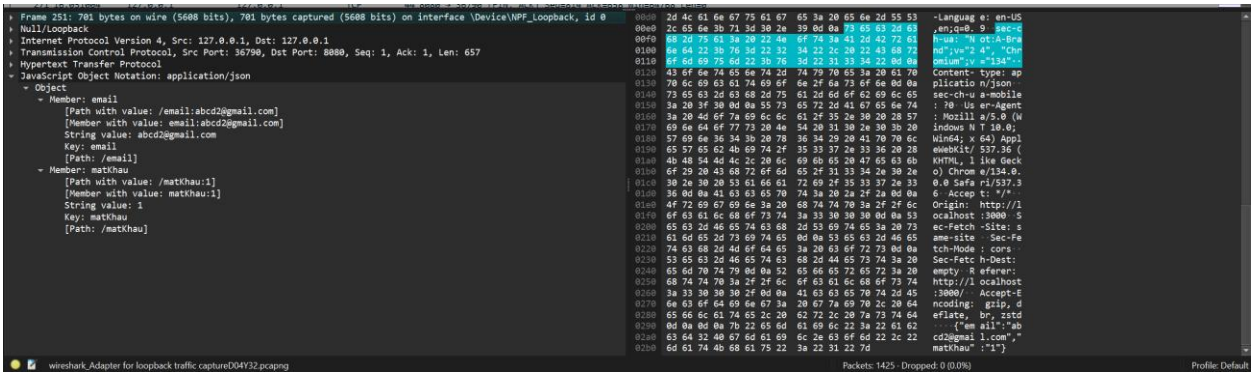
Client Privacy violation



Hình 2. Đăng nhập

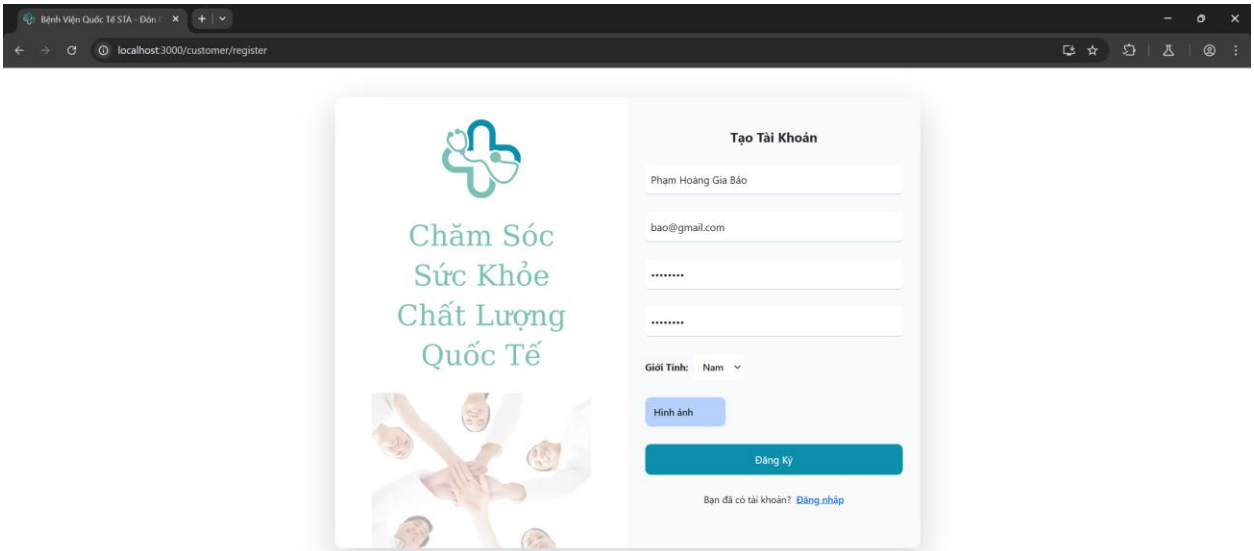
Chạy web và thử đăng nhập trong lúc đó thì chặn bằng burp suite và bắt gói tin wireshark

Hình 3. Burp suite bắt gói gửi

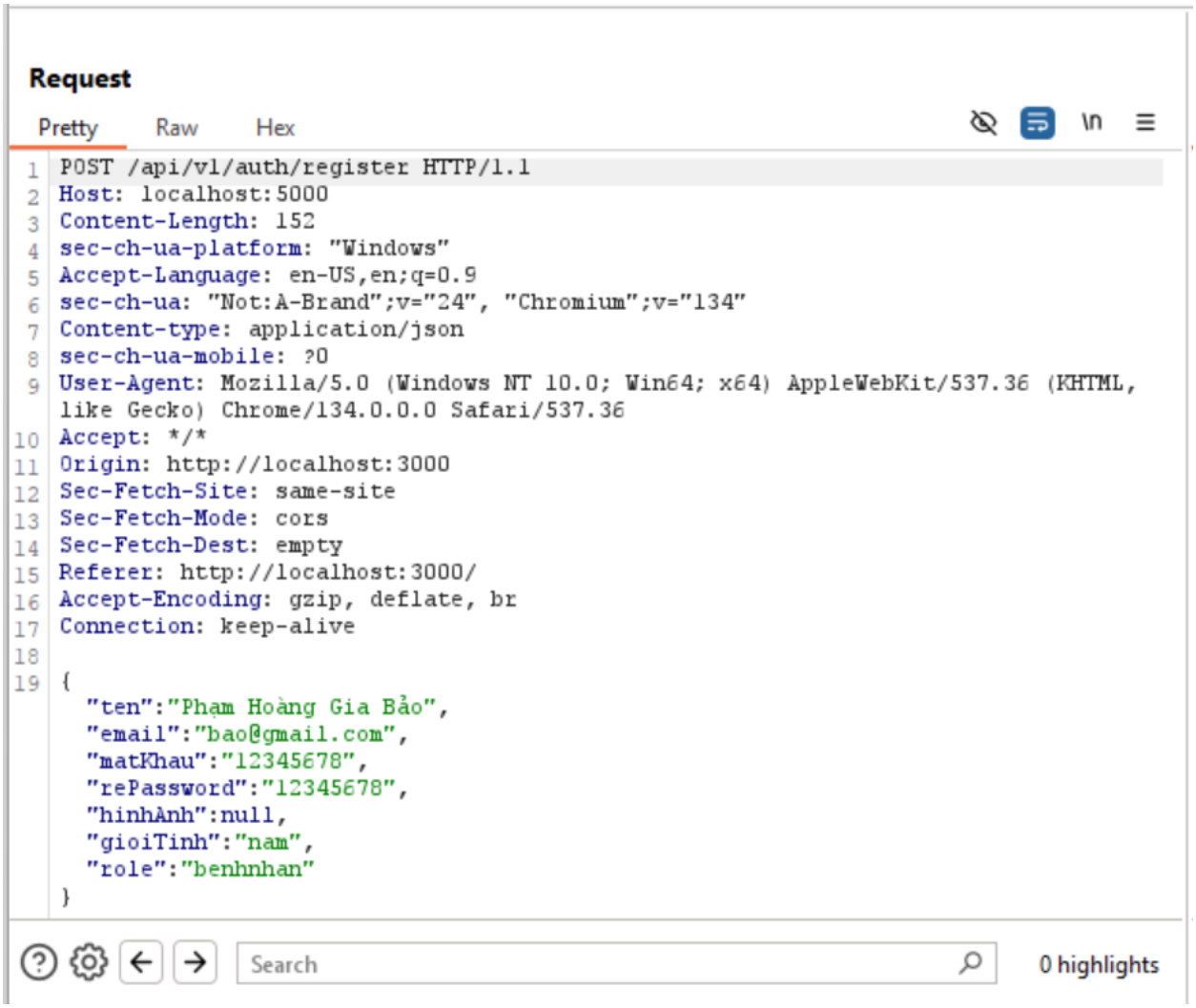


Hình 4. Bắt bằng wireshark đăng nhập

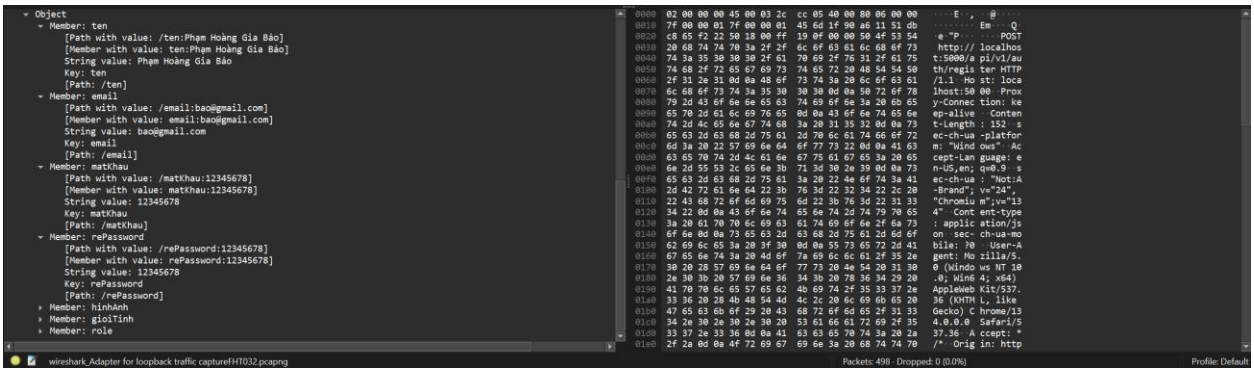
Chạy web và đăng kí thử tài khoản sau đó làm tương tự



Hình 5. Trang đăng kí



Hình 6. Burp suite trang đăng kí



Hình 7. WireShark trang đăng kí

Thông báo quét lỗi của Checkmarx:

```
try {
  const res = await fetch('${BASE_URL}/api/v1/auth/register', {
    method: 'post',
    headers: {
      'Content-type': 'application/json'
    },
    body: JSON.stringify(formData)  // Gửi formData lên backend
  });
```

Hình 8. Checkmarx Client Privacy Violation

Sửa lỗi bảo mật:

Thay đổi hàm:

body: JSON.stringify({ data: encryptedData }) // Gửi dữ liệu đã mã hóa

Thêm hàm này vào:

import CryptoJS from "crypto-js";

const secretKey = "mySecretKey"; // Khóa bí mật, lưu ý không hardcode trong frontend

const encryptData = (data) => {

return CryptoJS.AES.encrypt(JSON.stringify(data), secretKey).toString();

};

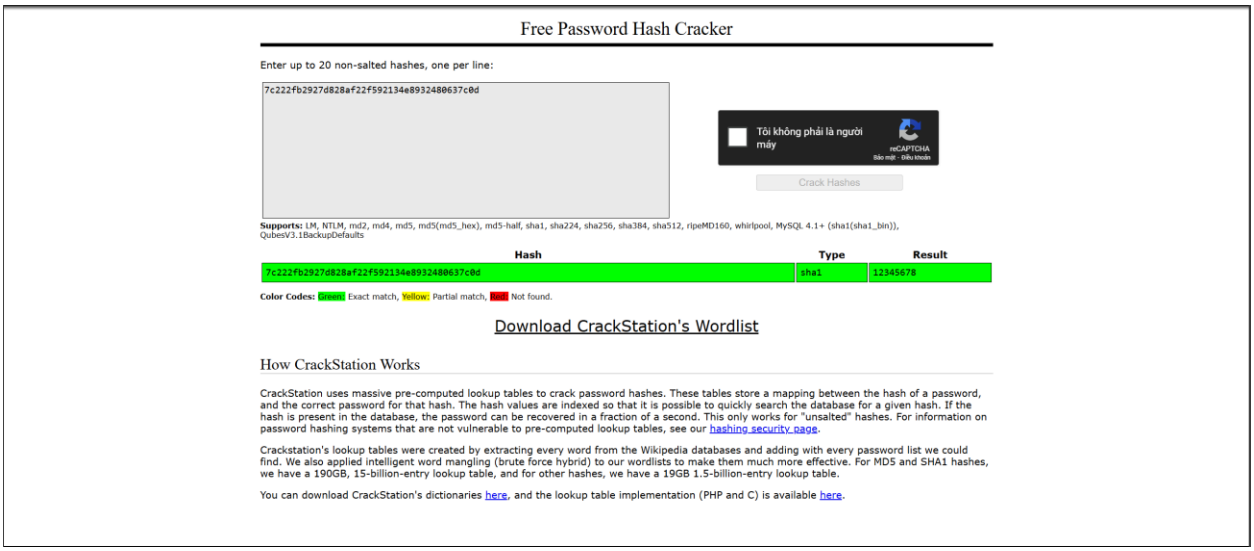
A2: Cryptographic Failures

Client Weak Cryptographic Hash

- Hệ thống sử dụng mã hoá SHA-256, MD5 hoặc SHA-1 thì mật khẩu có thể bị bẻ khóa dễ dàng bằng Rainbow Table Attack.
- Hacker có thể tạo danh sách hash sẵn và so sánh với database của bạn bằng.

```
_id: ObjectId('6719c4c2637d3316d99da181')
email : "test3@gmail.com"
password : "7c222fb2927d828af22f592134e8932480637c0d"
name : "John Doe"
photo : "photo_url"
role : "admin"
gender : "male"
appointments : Array (empty)
createdAt : 2024-10-24T03:53:38.992+00:00
updatedAt : 2024-10-24T04:17:56.662+00:00
__v : 0
```

Hình 9. Tài khoản trong cơ sở dữ liệu



Hình 10. Mật khẩu đã phá giải

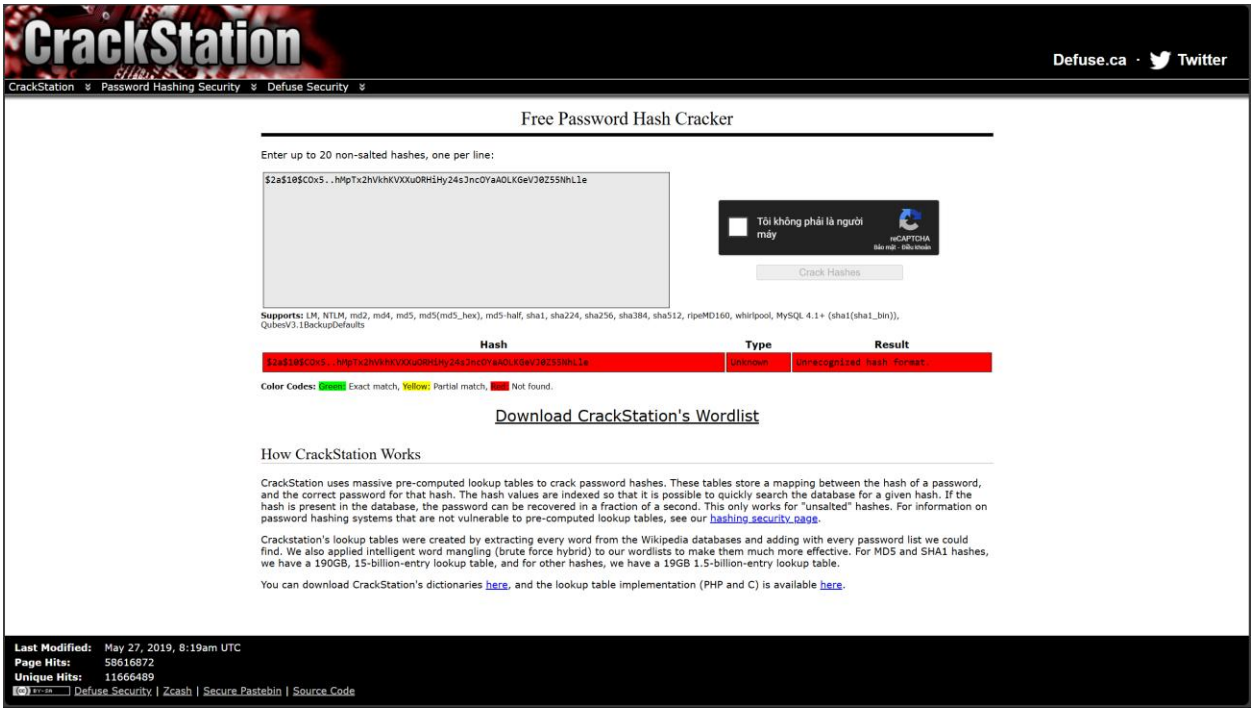
Phương án chống lại:

Thay đổi cách mã hoá sử dụng mã hoá lồng

```
var hash = crypto
  .createHash('sha1')
  .update(entity, 'utf8')
  .digest('base64')
  .substring(0, 27)
```

```
_id: ObjectId('6719c4c2637d3316d99da181')
email : "test3@gmail.com"
password : "$2a$10$COx5..hMpTx2hVkhKVXXuORHiHy24sJncOYaAOLKGeVJ0Z55NhLle"
name : "John Doe"
photo : "photo_url"
role : "admin"
gender : "male"
appointments : Array (empty)
createdAt : 2024-10-24T03:53:38.992+00:00
updatedAt : 2024-10-24T04:17:56.662+00:00
__v : 0
```


Hình 11. Mật khẩu sau mã hoá mới



Hình 12. Không thể crack

A3: Injection

Stored XSS

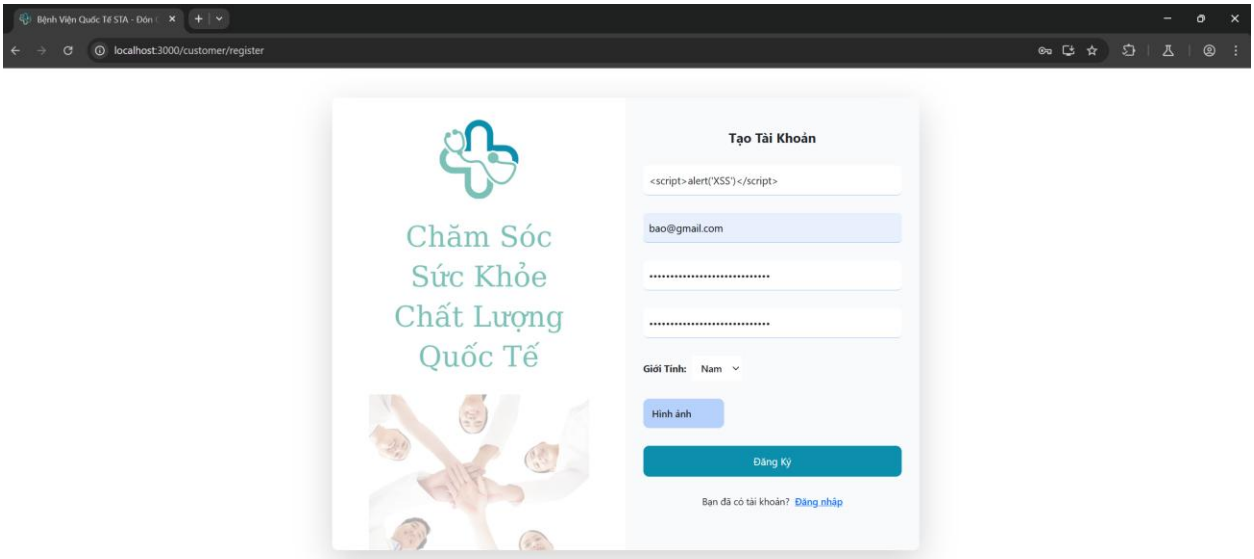
Thông báo của Checkmarx

```
export const getSingleUser = async (req, res) => {
    const id = req.params.id;

    try {
        const getaUser = await BenhNhan.findById(id).select("-matKhau");

        res.status(200).json({ success: true, message: 'Tìm người dùng thành công', data: getaUser });
    } catch (err) {
        res.status(500).json({ success: false, message: 'Tìm người dùng không thành công' });
    }
}
```

Hình 13. Checkmarx Stored XSS



Hình 14. Điền tấn công XSS vào

Phương pháp xử lý:

```
// Lọc XSS trước khi gửi về frontend
    const sanitizedUser = Object.keys(getaUser._doc).reduce((acc, key) => {
      acc[key] = xss(getaUser._doc[key]);  // Làm sạch dữ liệu
      return acc;
    }, {});
```

Thêm đoạn này vào sau trước khi gửi về database, Lọc dữ liệu trước khi trả về

A4: Insecure Design

Missing Encryption of Sensitive Data

Thông báo của Checkmarx

```
const hashPassword = crypto
    .createHash('md5')
    .update('${username}:mongo:${password}', 'utf8')
    .digest('hex');
```

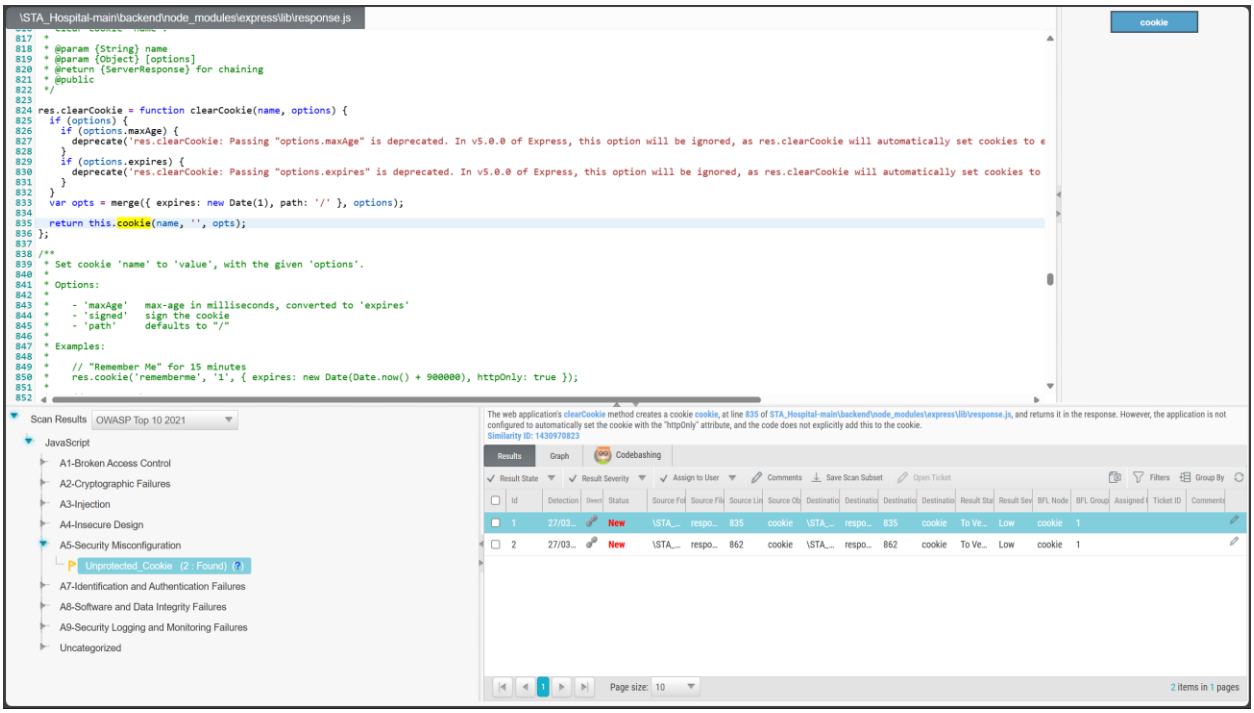
Hình 15. CheckMarx Missing Encryption of Sensitive Data

Cách xử lý sử dụng mã hoá SHA-256 thay cho md5 để đảm bảo an toàn bảo mật hơn

A5: Security Misconfiguration

Unprotected Cookie

Đoạn code ở file JavaScript không được cấu hình httpOnly cho cookie khiến cho cookie có thể bị lộ và kẻ tấn công có thể lợi dụng điều này để tấn công XSS hay Session Hijacking



Hình 16. Cookie

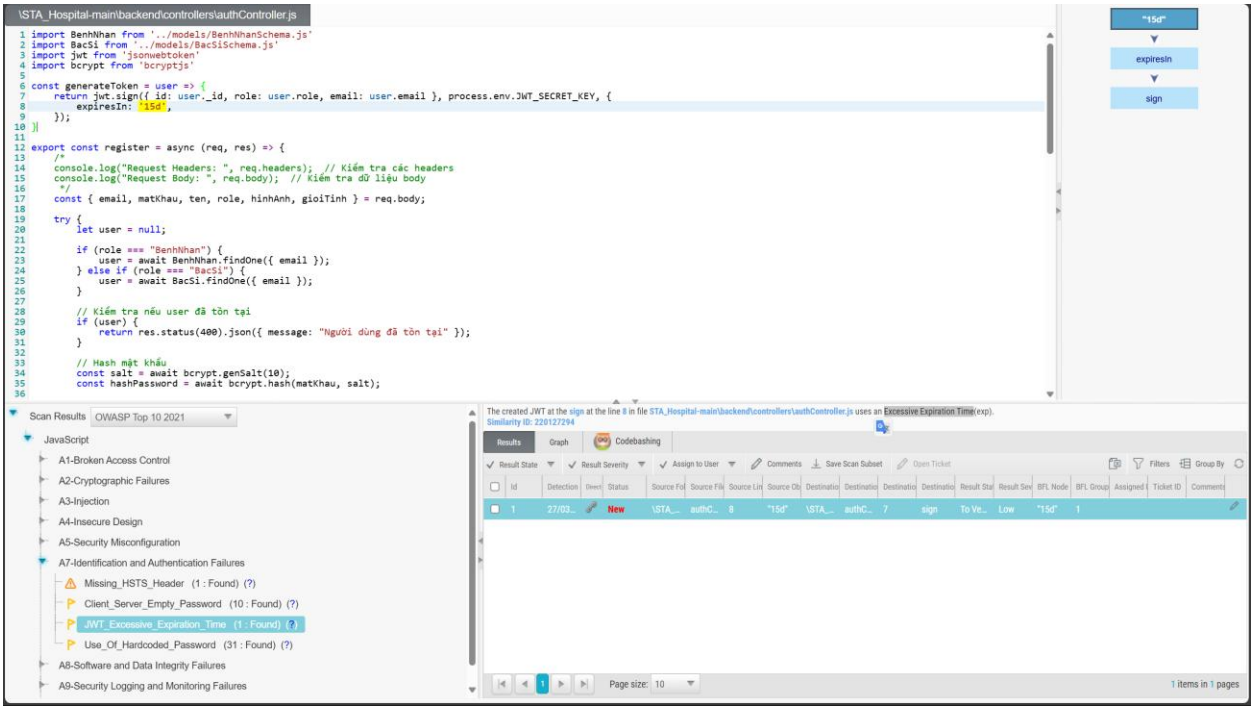
Phương pháp xử lý:

```
var opts = merge({
  expires: new Date(1),  // Đặt thời gian hết hạn về quá khứ
  path: '/',
  secure: true,          // Chỉ gửi cookie qua HTTPS
```

```
    httpOnly: true,         // Ngăn JavaScript truy cập cookie
    sameSite: 'Strict'      // Ngăn chặn CSRF
  }, options);
```

A7: Idenfication and Authentication Failures

JWT Excessive Expiration Time

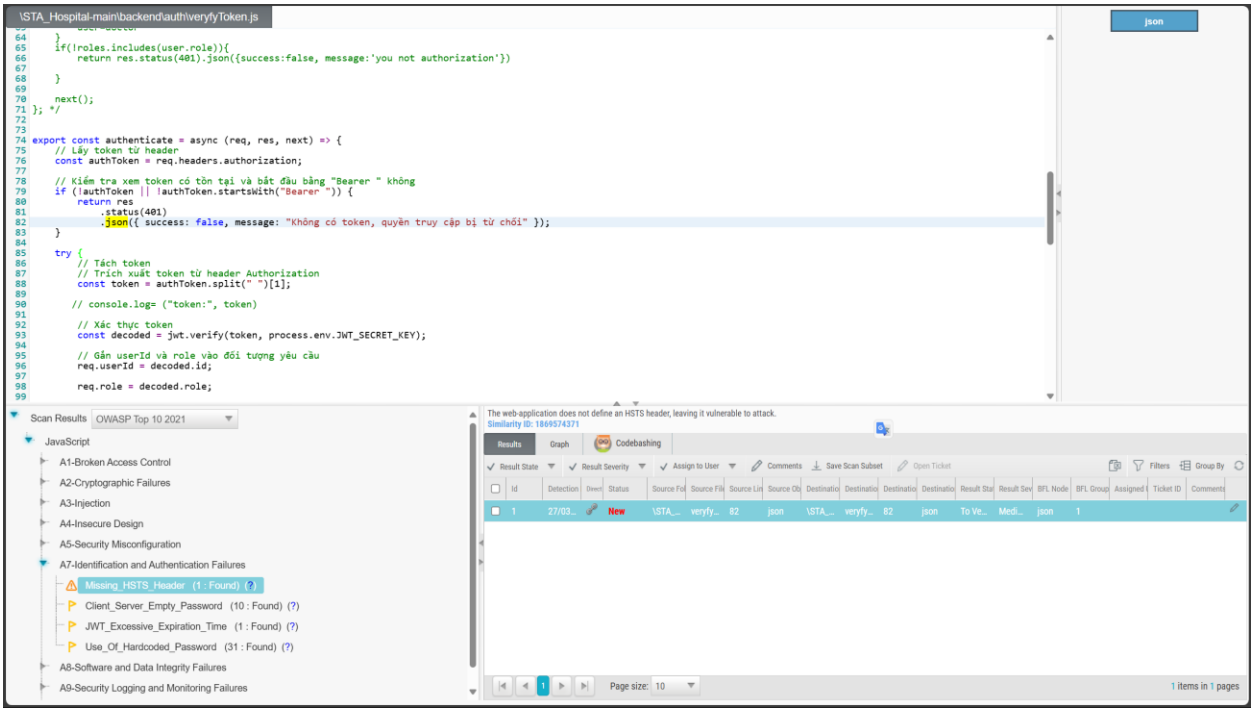


Hình 17. JWT Excessive Expiration Time

Nên giảm thời gian xuống dưới 1 ngày để bảo mật hơn

Missing HSTS Header

- Không có HSTS → Trình duyệt có thể tải trang qua HTTP nếu bị tấn công downgrade attack.
- Dữ liệu nhạy cảm trong JWT Token → Nếu không ép buộc HTTPS, token có thể bị đánh cắp khi truyền qua HTTP.



Hình 18. Missing HSTS Header

Phương pháp xử lý: Bật trong app.js hoặc server.js

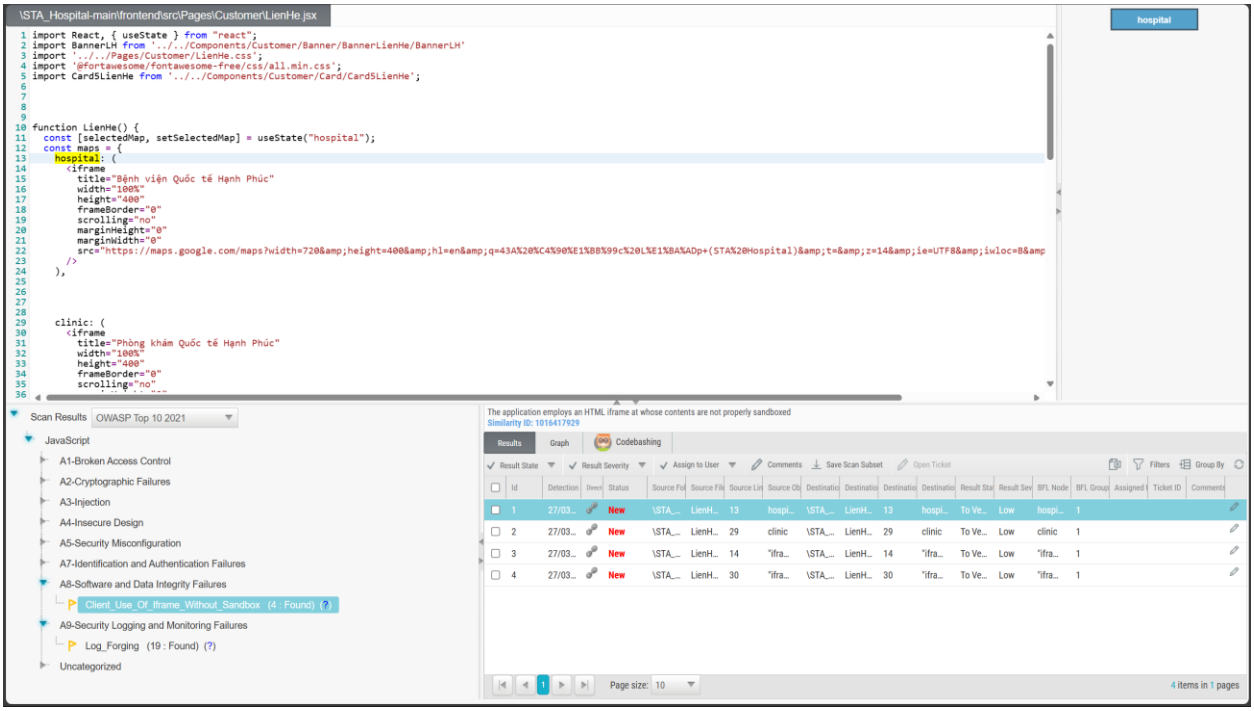
const helmet = require("helmet");

app.use(helmet()); // Kích hoạt toàn bộ bảo mật HTTP Headers, bao gồm HSTS

A8: Software and data intergrity failures

Client use of iframe without sandbox

- Không có sandbox → Nếu trang nhúng có mã độc, nó có thể tấn công trang chính của bạn.
- Không có allow → Iframe có thể có quyền truy cập vào các API nguy hiểm của trình duyệt.



Hình 19. Client use of iframe without sandbox

Phương pháp chống bảo mật:

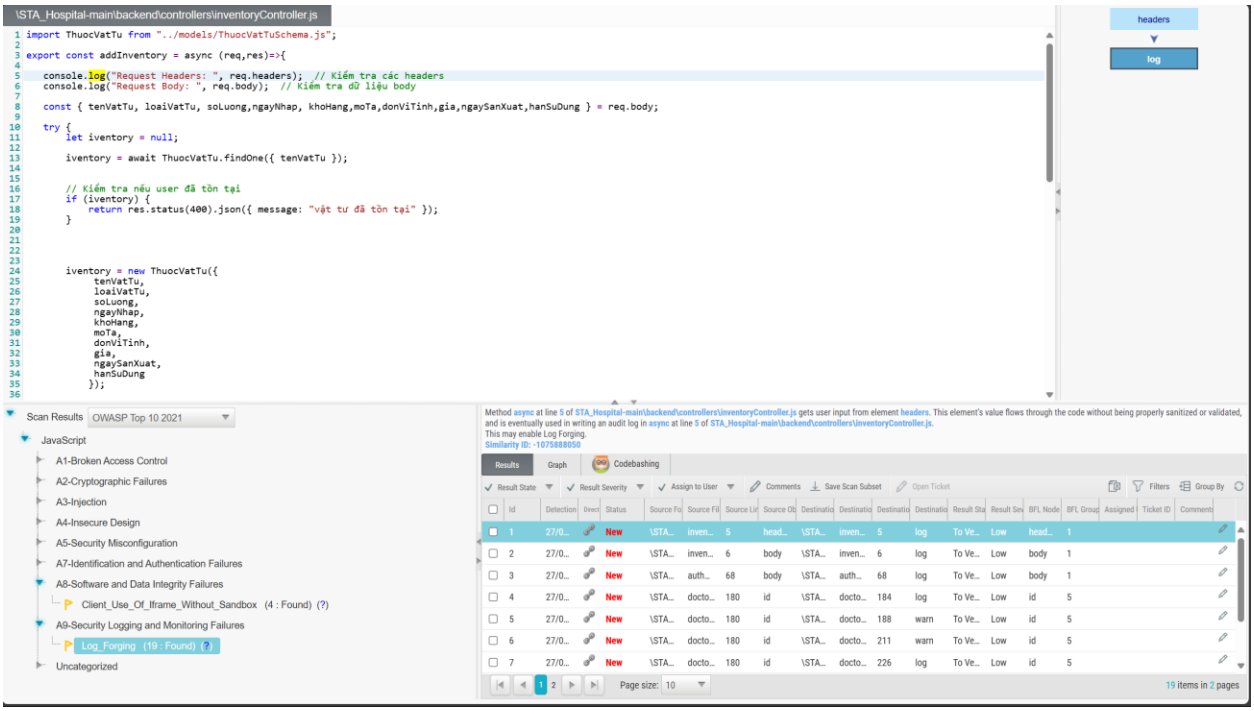
sandbox="allow-scripts allow-same-origin" // Thêm sandbox để bảo mật

allow="geolocation; fullscreen" // Chỉ cho phép các tính năng cần thiết

A9: Security Logging and Monitoring Failures

Log Forging

- Kẻ tấn công có thể chèn ký tự đặc biệt vào log
  - Ví dụ: req.body = "User logged out\nAdmin deleted all data" → Log sẽ bị giả mạo.
- Mã độc ANSI Escape → Có thể thay đổi màu sắc hoặc che giấu log thật.
- Chèn mã JSON hoặc SQL vào log → Có thể gây nhầm lẫn khi debug hoặc bị khai thác.



Hình 20. Log Forging

Phương pháp thực hiện:

const cleanHeaders = JSON.stringify(req.headers).replace(/[\n\r]/g, " ");

const cleanBody = JSON.stringify(req.body).replace(/[\n\r]/g, " ");

CHƯƠNG 4: ĐÁNH GIÁ VÀ KẾT LUẬN

1. Đánh giá kết quả

Quá trình kiểm thử xâm nhập (Penetration Testing) trên website STA HOSPITAL đã giúp chúng em phát hiện nhiều lỗ hổng bảo mật nghiêm trọng, bao gồm:

- Broken Access Control: Thông tin người dùng không được mã hóa, dễ bị lộ qua gói tin mạng
- Cryptographic Failures: Sử dụng hàm băm MD5 không có salt, dễ bị brute-force.
- Injection (XSS): Lỗ hổng Reflected XSS cho phép chèn mã độc vào URL.
- Security Misconfiguration: Cookie không được cấu hình HttpOnly, dẫn đến nguy cơ Session Hijacking.

- Identification and Authentication Failures: Thiếu HSTS Header và Session Fixation, làm tăng rủi ro bị tấn công.

Các lỗ hổng này đều nằm trong danh sách OWASP Top 10 2021, chứng tỏ tầm quan trọng của việc kiểm thử bảo mật định kỳ.

2. **Khó khan và hạn chế**

Trong quá trình thực hiện, nhóm đã gặp một số khó khăn:

- Thiếu thông tin chi tiết: Một số phần của website không được cung cấp đầy đủ mã nguồn, gây khó khăn trong việc phân tích sâu.
- Giới hạn thời gian: Do thời gian có hạn, nhóm chưa thể kiểm thử toàn diện tất cả các chức năng của website.
- Công cụ hạn chế: Một số công cụ như Burp Suite và Wireshark yêu cầu cấu hình phức tạp, đòi hỏi thời gian làm quen.

3. **Giải pháp đề xuất**

Để cải thiện bảo mật cho website, nhóm đề xuất các giải pháp sau:

- Mã hóa dữ liệu nhạy cảm: Sử dụng thuật toán mã hóa mạnh như AES hoặc SHA-256 kết hợp với salt.
- Cấu hình Cookie an toàn: Bật HttpOnly và Secure Flag cho cookie để ngăn chặn Session Hijacking.
- Triển khai HSTS: Áp dụng HTTP Strict Transport Security (HSTS) để chống tấn công MITM.
- Kiểm tra đầu vào: Xử lý kỹ dữ liệu đầu vào để ngăn chặn XSS và SQL Injection.
- Đào tạo nhận thức bảo mật: Nâng cao kiến thức bảo mật cho đội ngũ phát triển và người dùng.

4. **Kết luận**

Đề tài "Penetration Testing Website" đã giúp nhóm hiểu rõ hơn về quy trình kiểm thử xâm nhập, các phương pháp phát hiện lỗ hổng và cách khắc phục chúng. Kết quả thu được không chỉ có ý nghĩa học thuật mà còn góp phần nâng cao nhận thức về an toàn

thông tin trong môi trường thực tế. Trong tương lai, nhóm hy vọng có thể tiếp tục nghiên cứu sâu hơn về các công cụ và kỹ thuật kiểm thử hiện đại để đóng góp vào việc xây dựng các hệ thống an toàn và bảo mật hơn.