**Báo Cáo Bảo Mật Dự Án Xây Dựng Website Dịch Vụ Công Quản Lý Thông Tin Tạm Trú Người Nước Ngoài Tại Hà Nội**

**1. Giới Thiệu**

Dự án xây dựng website dịch vụ công nhằm hỗ trợ công tác quản lý thông tin tạm trú của người nước ngoài trên địa bàn thành phố Hà Nội. Hệ thống giúp các cơ quan chức năng dễ dàng tiếp nhận, theo dõi và quản lý thông tin cư trú, đảm bảo tính chính xác và kịp thời trong công tác quản lý nhà nước. Bảo mật dữ liệu là yếu tố quan trọng hàng đầu nhằm bảo vệ thông tin cá nhân của người nước ngoài cũng như đảm bảo sự vận hành ổn định của hệ thống. Báo cáo này sẽ phân tích các biện pháp bảo mật đã được áp dụng trong dự án.

**2. Các Nguy Cơ Bảo Mật Trong Hệ Thống**

Hệ thống web thường đối mặt với nhiều nguy cơ bảo mật, trong đó bao gồm:

* **Tấn công SQL Injection**: Kẻ tấn công chèn mã SQL để thao tác dữ liệu.
* **Cross-Site Scripting (XSS)**: Kẻ tấn công chèn mã JavaScript độc hại.
* **Cross-Site Request Forgery (CSRF)**: Kẻ tấn công thực hiện yêu cầu mạo danh.
* **Brute Force Attack**: Thử tất cả các khả năng để đoán mật khẩu.
* **Data Exposure**: Dữ liệu bị lộ do truy vấn không được bảo vệ.

**3. Các Giải Pháp Bảo Mật Đã Áp Dụng**

**3.1 Xác Thực và Phân Quyền Người Dùng**

Xác thực và phân quyền người dùng là bước quan trọng để đảm bảo hệ thống chỉ cho phép những người có quyền hợp lệ truy cập. Để thực hiện điều này, hệ thống đã sử dụng **Passport.js**, một thư viện hỗ trợ nhiều cơ chế xác thực phổ biến như:

* **JWT (JSON Web Token):** JWT được sử dụng để xác thực và duy trì phiên làm việc của người dùng thông qua token. Sau khi đăng nhập thành công, một token được tạo ra và gửi đến client. Token này sẽ được sử dụng trong các yêu cầu tiếp theo để xác định danh tính người dùng, giúp loại bỏ nhu cầu lưu session trên server.
* **OAuth2:** Hỗ trợ đăng nhập bằng các tài khoản bên thứ ba như Google, Facebook. Điều này giúp người dùng dễ dàng đăng nhập mà không cần tạo tài khoản mới, đồng thời tăng tính bảo mật do không cần lưu trữ mật khẩu trực tiếp trên hệ thống.
* **Local Strategy:** Hệ thống cũng hỗ trợ xác thực bằng tên đăng nhập và mật khẩu truyền thống. Để bảo vệ dữ liệu, mật khẩu được hash trước khi lưu vào cơ sở dữ liệu.

**3.2 Bảo Vệ Header HTTP**

Các lỗ hổng bảo mật phổ biến thường xuất hiện từ việc không bảo vệ đúng cách các tiêu đề HTTP. Để khắc phục điều này, hệ thống tích hợp **Helmet.js**, giúp giảm thiểu các rủi ro bảo mật bằng cách thiết lập các tiêu đề bảo vệ như:

* **Content Security Policy (CSP):** Giúp ngăn chặn các cuộc tấn công XSS (Cross-Site Scripting) bằng cách chỉ cho phép tải tài nguyên từ những nguồn đáng tin cậy.
* **X-Frame-Options:** Ngăn chặn Clickjacking bằng cách không cho phép website bị nhúng trong iframe của trang web khác.
* **X-Powered-By:** Gỡ bỏ tiêu đề này để không tiết lộ thông tin về công nghệ server đang sử dụng, giúp giảm thiểu nguy cơ tấn công từ hacker.

**3.3 Bảo Mật Mật Khẩu**

Để bảo vệ thông tin đăng nhập của người dùng, hệ thống sử dụng **bcrypt.js** để hash mật khẩu trước khi lưu vào cơ sở dữ liệu. Việc này đảm bảo rằng nếu cơ sở dữ liệu bị rò rỉ, hacker cũng không thể dễ dàng lấy được mật khẩu gốc.

**bcrypt.js** hoạt động bằng cách thực hiện nhiều vòng băm mật khẩu với một "muối" (salt) ngẫu nhiên, giúp tăng độ khó khi tấn công bằng phương pháp brute force hay rainbow table.

**3.4 Chống Tấn Công Brute Force**

Brute force là kỹ thuật hacker sử dụng để thử hàng triệu mật khẩu khác nhau nhằm tìm ra mật khẩu đúng. Để giảm thiểu nguy cơ này, hệ thống áp dụng **express-rate-limit** để giới hạn số lần thử đăng nhập từ một IP trong một khoảng thời gian nhất định. Khi người dùng nhập sai mật khẩu quá số lần cho phép, hệ thống sẽ tạm thời khóa tài khoản hoặc yêu cầu xác minh bổ sung.

**3.5 Bảo Vệ API và Dữ Liệu**

API là cửa ngõ để ứng dụng giao tiếp với dữ liệu và người dùng, vì vậy cần có các biện pháp bảo vệ chặt chẽ:

* **Validation input:** Sử dụng **express-validator** để lọc dữ liệu đầu vào nhằm ngăn chặn các dữ liệu độc hại hoặc không hợp lệ.
* **Sử dụng ORM (Sequelize/Mongoose):** Hạn chế SQL Injection bằng cách sử dụng ORM thay vì viết truy vấn SQL thuần.
* **Chống CSRF (Cross-Site Request Forgery):** Sử dụng **CSRF token** và **SameSite cookies** để đảm bảo các yêu cầu quan trọng chỉ được thực hiện bởi những người dùng hợp lệ.

**BÁO CÁO KIỂM THỬa BẢO MẬT DỰ ÁN**

**1. Giới Thiệu**

Trong khuôn khổ việc phát triển và kiểm tra bảo mật dự án, nhóm đã sử dụng công cụ **OWASP ZAP (Zed Attack Proxy)** để quét và phát hiện các lỗ hổng bảo mật tiềm ẩn. Quá trình kiểm thử nhằm đánh giá mức độ an toàn trước khi triển khai hệ thống lên môi trường sản phẩm (production).

**2. Công Cụ Kiểm Thử**

* **Công cụ:** OWASP ZAP v2.13.0
* **Môi trường quét:** http://localhost:3000
* **Phương pháp:** Quét tự động và phân tích nguy cơ bảo mật theo các mức độ rủi ro

**3. Kết Quả Kiểm Thử**

Báo cáo OWASP ZAP đã phát hiện tổng cộng **12 cảnh báo**, trong đó:

* **Rủi ro trung bình:** 2 cảnh báo (16.7%)
* **Rủi ro thấp:** 3 cảnh báo (25.0%)
* **Cảnh báo thông tin:** 7 cảnh báo (58.3%)

**Các lỗ hổng chính:**

1. **CSP: Wildcard Directive** (16 cảnh báo, mức rủi ro trung bình)
2. **CSP: style-src unsafe-inline** (16 cảnh báo, mức rủi ro trung bình)
3. **Cookie No HttpOnly Flag** (4 cảnh báo, mức rủi ro thấp)
4. **Cookie without SameSite Attribute** (9 cảnh báo, mức rủi ro thấp)
5. **Cross-Domain JavaScript Source File Inclusion** (72 cảnh báo, mức rủi ro thấp)
6. **Authentication Request Identified** (3 cảnh báo, mức thông tin)
7. **Information Disclosure - Sensitive Information in URL** (11 cảnh báo, mức thông tin)
8. **User Agent Fuzzer** (312 cảnh báo, mức thông tin)

**5. Giải Thích Kết Quả Kiểm Thử**

Kết quả kiểm thử chỉ ra một số lỗ hổng và rủi ro bảo mật, tuy nhiên, phần lớn các cảnh báo này xuất hiện do hệ thống đang chạy trong **môi trường phát triển (dev environment)**, nơi chưa áp dụng đầy đủ các chính sách bảo mật chặt chẽ như khi triển khai chính thức. Điều này hoàn toàn bình thường và không phải là dấu hiệu của các lỗ hổng nghiêm trọng trong môi trường production. Tất cả các vấn đề được ghi nhận đều có thể xử lý dễ dàng thông qua các biện pháp cấu hình và tối ưu khi hệ thống đi vào hoạt động chính thức.

**5. 1. Thiếu Content Security Policy (CSP) Header**

CSP là một chính sách bảo mật quan trọng giúp ngăn chặn tấn công XSS bằng cách kiểm soát các nguồn tài nguyên được phép tải về và thực thi. Trong môi trường phát triển, việc thiếu CSP có thể là do hệ thống đang trong giai đoạn thử nghiệm, cần linh hoạt trong việc tải tài nguyên từ nhiều nguồn khác nhau, bao gồm cả thư viện bên ngoài.

**Cách khắc phục khi deploy:**

* Cấu hình CSP trong header của server để chỉ cho phép nội dung từ các nguồn tin cậy.
* Kiểm tra và loại bỏ các script không cần thiết hoặc được tải từ nguồn không đáng tin cậy.
* Có thể sử dụng CSP với chính sách default-src 'self' để giới hạn việc tải tài nguyên chỉ từ chính hệ thống.

**5. 2. File ẩn được phát hiện**

Trong quá trình phát triển, hệ thống có thể tạo ra một số file tạm thời hoặc file cấu hình nội bộ để phục vụ kiểm thử. Những file này thường không có giá trị trong môi trường production và có thể bị bỏ sót khi quét bảo mật.

**Cách khắc phục khi deploy:**

* Kiểm tra và loại bỏ các file không cần thiết trước khi đưa lên production.
* Sử dụng .gitignore để tránh đẩy các file nhạy cảm hoặc tạm thời lên hệ thống quản lý mã nguồn.
* Đảm bảo thiết lập quyền truy cập chặt chẽ để ngăn chặn người ngoài truy cập vào các thư mục nhạy cảm.

**5. 3. Cookie không có HttpOnly & SameSite**

Cảnh báo này xuất hiện do cookie trong môi trường phát triển có thể chưa được thiết lập với cờ bảo mật để dễ dàng kiểm thử và debug.

**Cách khắc phục khi deploy:**

* Cấu hình cookie với các thuộc tính bảo mật như HttpOnly, Secure, SameSite=Strict để ngăn chặn truy cập trái phép và tấn công CSRF.
* Đảm bảo tất cả cookie quan trọng chỉ có thể truy cập được qua HTTP chứ không bị lộ qua JavaScript.

**5. 4. Tải JavaScript từ nguồn bên ngoài**

Trong môi trường phát triển, hệ thống có thể tải nhiều thư viện từ các nguồn bên ngoài như CDN để thử nghiệm nhanh hơn. Tuy nhiên, điều này có thể làm tăng rủi ro nếu các thư viện này bị chỉnh sửa hoặc bị thay đổi nội dung độc hại.

**Cách khắc phục khi deploy:**

* Chỉ sử dụng các thư viện từ nguồn chính thức và đáng tin cậy.
* Nếu có thể, nên tải xuống và host các thư viện này trong hệ thống thay vì phụ thuộc vào CDN bên ngoài.
* Áp dụng CSP để hạn chế việc tải JavaScript từ các nguồn không xác định.

**5. 5. Lộ thông tin phiên bản server qua HTTP Header**

Một số web server mặc định gửi thông tin phiên bản của hệ thống, điều này có thể vô tình tiết lộ chi tiết về nền tảng đang sử dụng, giúp kẻ tấn công dễ dàng tìm kiếm các lỗ hổng phù hợp.

**Cách khắc phục khi deploy:**

* Cấu hình lại server để ẩn các thông tin không cần thiết. Ví dụ, với Nginx, có thể sử dụng server\_tokens off; hoặc với Apache, có thể dùng ServerSignature Off.
* Chặn hoặc thay đổi thông tin header trả về bằng cách cấu hình trong reverse proxy hoặc firewall.

**5. 6. Thiếu X-Content-Type-Options Header**

Header này giúp ngăn chặn trình duyệt tự động đoán định dạng file (MIME sniffing), tránh trường hợp nội dung bị thực thi dưới dạng khác với dự kiến.

**Cách khắc phục khi deploy:**

* Thêm header X-Content-Type-Options: nosniff vào phản hồi HTTP để buộc trình duyệt sử dụng đúng kiểu MIME được khai báo.
* Kiểm tra lại tất cả tài nguyên tĩnh (CSS, JavaScript, hình ảnh, v.v.) để đảm bảo có thiết lập đúng Content-Type.

**5. 7. User Agent Fuzzer**

Cảnh báo này xuất hiện do công cụ kiểm thử gửi nhiều request với các User-Agent khác nhau để tìm kiếm lỗ hổng. Đây không phải là một lỗi bảo mật thực sự mà chỉ là kết quả của quá trình kiểm thử.

**Cách khắc phục khi deploy:**

* Không cần hành động cụ thể đối với cảnh báo này.
* Có thể thiết lập firewall hoặc các bộ lọc trên server để chặn những User-Agent không mong muốn nếu thấy có hành vi đáng ngờ.

**5. 8. Bình luận chứa thông tin nhạy cảm**

Trong quá trình phát triển, lập trình viên có thể để lại các comment hoặc ghi chú trong code để tiện theo dõi. Tuy nhiên, nếu các comment này chứa thông tin nhạy cảm (API key, cấu hình server, thông tin tài khoản test, v.v.), chúng có thể vô tình bị lộ ra ngoài.

**Cách khắc phục khi deploy:**

* Rà soát và loại bỏ tất cả các comment không cần thiết trước khi deploy.
* Sử dụng công cụ tự động kiểm tra code (như ESLint, SonarQube) để phát hiện các thông tin nhạy cảm trong comment.
* Đảm bảo rằng không có thông tin quan trọng nào xuất hiện trong response gửi đến client.

A screenshot of a blue and white document

AI-generated content may be incorrect.

A screenshot of a report

AI-generated content may be incorrect.

A screenshot of a computer screen

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.