Triển khai các ví dụ cho lỗ hổng bảo mật web theo OWASP (A05, A07-A10)

Nguyễn Thế Tiến Đạt[1], Lê Dương Tấn Minh[1], Bùi Võ Duy Vũ[1]

[1] Khoa Khoa học và Kỹ thuật Thông tin, Trường Đại học Công nghệ Thông tin, Thành phố Hồ Chí Minh, Việt Nam

Email: 22520865@gm.uit.edu.vn, 22520225@gm.uit.edu.vn, 22521685@gm.uit.edu.vn

***Tóm tắt*—** Bảo mật web là một vấn đề quan trọng trong bối cảnh kỹ thuật số hiện nay, khi các ứng dụng web được sử dụng rộng rãi trong nhiều lĩnh vực như tài chính, y tế và giáo dục. Đồ án OWASP (Open Web Application Security Project) đã làm nổi bật các lỗ hổng phổ biến trong các ứng dụng web, nhấn mạnh nhu cầu về các kỹ thuật phòng ngừa và giảm thiểu hiệu quả. Bài báo cáo này này tập trung vào việc triển khai các biện pháp bảo mật cho năm lỗ hổng OWASP quan trọng: A05 - Cấu hình bảo mật sai, A07 - Lỗi trong xác thực và nhận dạng, A08 - Lỗi về tính toàn vẹn của phần mềm và dữ liệu, A09 - Lỗi trong ghi nhật ký và giám sát bảo mật, và A10 - Giả mạo yêu cầu phía máy chủ (SSRF). Thông qua việc cung cấp các ví dụ thực tế và giải thích chi tiết về từng lỗ hổng, nghiên cứu này nhằm nâng cao hiểu biết về các điểm yếu bảo mật này và đưa ra các giải pháp thực tiễn để ngăn chặn các cuộc tấn công tiềm ẩn.

Từ khóa—OWASP, bảo mật web, A05, A07, A08, A09, A10, phòng ngừa lỗ hổng, các phương pháp bảo mật.

I. GIỚI THIỆU

Trong thời đại công nghệ số, các ứng dụng web ngày càng trở nên phổ biến và đóng vai trò quan trọng trong nhiều lĩnh vực như kinh doanh, y tế, giáo dục, và tài chính. Tuy nhiên, sự phát triển nhanh chóng của công nghệ cũng đi kèm với nhiều thách thức về bảo mật. Các lỗ hổng trong ứng dụng web có thể dẫn đến các cuộc tấn công mạng, gây ra tổn thất về tài chính, uy tín, và ảnh hưởng đến quyền riêng tư của người dùng.

OWASP (Open Web Application Security Project) là một tổ chức phi lợi nhuận quốc tế với mục tiêu cải thiện bảo mật phần mềm. OWASP công bố danh sách các lỗ hổng bảo mật phổ biến trong ứng dụng web, giúp các nhà phát triển và chuyên gia bảo mật nhận diện và khắc phục các điểm yếu trong hệ thống của mình. Trong danh sách OWASP Top 10, một số lỗ hổng tiêu biểu bao gồm:

- A05: Security Misconfiguration (Cấu hình bảo mật sai)
- A07: Identification and Authentication Failures (Lỗi trong xác thực và nhận dạng)
- A08: Software and Data Integrity Failures (Lỗi về tính toàn vẹn của phần mềm và dữ liệu)
- A09: Security Logging and Monitoring Failures (Lỗi trong ghi nhật ký và giám sát bảo mật)
- A10: Server-Side Request Forgery (SSRF) (Giả mạo yêu cầu phía máy chủ)

Mục tiêu của đồ án này là cung cấp các ví dụ minh họa và phương pháp triển khai bảo mật cho các lỗ hổng OWASP A05, A07, A08, A09 và A10. Thông qua việc phân tích và mô phỏng các lỗ hổng này, đồ án sẽ giúp mọi người rõ hơn về cách các lỗ hổng có thể bị khai thác, cũng như các biện pháp phòng ngừa và bảo vệ hệ thống khỏi các cuộc tấn công tiềm ẩn.

II. CÁC LỖ HỔNG OWASP(A05, A07-A10)

A. A05 - Security Misconfiguration (Cấu hình bảo mật sai)

1) Security Misconfiguration (cấu hình bảo mật sai) là một lỗ hổng bảo mật thường gặp khi hệ thống không được thiết lập và duy trì đúng các thông số bảo mật. Điều này có thể xảy ra trong nhiều lớp của ứng dụng, từ cấu hình máy chủ, hệ điều hành, ứng dụng đến cơ sở dữ liệu, khi các thiết lập không phù hợp hoặc lỗi thời.

2) Nguyên nhân dẫn đến Security Misconfiguration:

- Cấu hình mặc định không an toàn: Các hệ thống thường được cài đặt với cấu hình mặc định để dễ sử dụng, nhưng nếu không thay đổi sẽ dễ dàng bị khai thác.
- Cấp quyền không cần thiết: Hệ thống cung cấp quyền truy cập nhiều hơn mức cần thiết cho người dùng hoặc ứng dụng.
- Thiếu cập nhật bảo mật: Không cập nhật các bản vá lỗi hoặc cập nhật phiên bản có thể khiến hệ thống dễ bị tấn công.
- Hiển thị thông tin nhạy cảm: Khi xảy ra lỗi, hệ thống tiết lộ thông tin chi tiết như thông báo lỗi, cấu trúc hệ thống, hoặc mã nguồn, tạo điều kiện cho kẻ tấn công.

B. A07 - Identification and Authentication Failures (Lỗi trong xác thực và nhận dạng)

1) Identification and Authentication Failures (Lỗi trong xác thực và nhận dạng) là: Lỗi xác thực và nhận dạng xảy ra khi hệ thống không đảm bảo chắc chắn rằng người dùng là người mà họ khai báo. Điều này thường bao gồm các vấn đề như yếu điểm trong quản lý phiên đăng nhập (session), mật khẩu yếu, và các lỗi trong cơ chế xác thực hai yếu tố (2FA).

2) Nguyên nhân dẫn đến Identification and Authentication Failure:

- Đoán mật khẩu (Brute-force): Nếu mật khẩu yếu hoặc chính sách mật khẩu không đủ mạnh, kẻ tấn công có thể dùng brute-force để xâm nhập.
- Khai thác lỗ hổng quản lý phiên đăng nhập: Nếu hệ thống không quản lý tốt cookie hoặc token phiên, kẻ tấn công có thể cướp phiên để chiếm quyền truy cập.
- Phá vỡ 2FA: Nếu quá trình xác thực 2 yếu tố không được triển khai đúng cách, kẻ tấn công có thể bỏ qua hoặc phá vỡ lớp bảo vệ này.

C. A08 - Software and Data Integrity Failures (Lỗi về tính toàn vẹn của phần mềm và dữ liệu)

1) Software and Data Integrity Failures (Lỗi về tính toàn vẹn của phần mềm và dữ liệu) là: Lỗi này xảy ra khi hệ thống không đảm bảo tính toàn vẹn của phần mềm và dữ liệu trong quá trình lưu trữ và truyền tải. Điều này dễ dẫn đến các tình huống bị tiêm mã độc, mã độc ransomware, hoặc các phần mềm giả mạo.

2) Nguyên nhân dẫn đến Software and Data Integrity Failures:

- Thay thế mã độc trong các bản cập nhật phần mềm: Kẻ tấn công có thể chèn mã độc vào trong các bản cập nhật, làm lây nhiễm toàn bộ hệ thống.
- Giả mạo dữ liệu: Dữ liệu quan trọng bị sửa đổi hoặc giả mạo trong quá trình truyền tải có thể dẫn đến những hậu quả nghiêm trọng.

D. A09 - Security Logging and Monitoring Failures (Lỗi trong ghi nhật ký và giám sát bảo mật)

1) Security Logging and Monitoring Failures (Lỗi trong ghi nhật ký và giám sát bảo mật) là: Lỗ hổng này xảy ra khi hệ thống không ghi lại đầy đủ các sự kiện bảo mật hoặc không giám sát để phát hiện sớm các hành vi bất thường. Điều này dẫn đến việc không phát hiện kịp thời các cuộc tấn công hoặc hành vi xâm nhập.

2) Nguyên nhân dẫn đến Security Logging and Monitoring Failures :

- Giấu các hành động xâm nhập: Khi hệ thống không ghi lại đầy đủ, kẻ tấn công có thể thực hiện các hành động xâm nhập mà không bị phát hiện.
- Kéo dài thời gian phát hiện: Thiếu giám sát có thể khiến cho các hành vi bất thường chỉ được phát hiện sau khi thiệt hại đã xảy ra.

E. A10 - Server-Side Request Forgery (SSRF) (Giả mạo yêu cầu phía máy chủ)

1) Server-Side Request Forgery (SSRF) (Giả mạo yêu cầu phía máy chủ): SSRF là lỗ hổng cho phép kẻ tấn công giả mạo các yêu cầu phía máy chủ và khiến máy chủ gửi yêu cầu đến các dịch vụ hoặc máy chủ khác mà kẻ tấn công chỉ định. Điều này thường xảy ra khi hệ thống chấp nhận và thực thi các yêu cầu URL từ người dùng mà không kiểm tra hoặc giới hạn.

2) Nguyên nhân dẫn đến Server-Side Request Forgery (SSRF):

- Truy cập trái phép vào các tài nguyên nội bộ: Kẻ tấn công có thể sử dụng SSRF để truy cập vào tài nguyên nội bộ mà thường bị hạn chế truy cập từ bên ngoài, như cơ sở dữ liệu hoặc dịch vụ API nội bộ.
- Dò quét mạng: Kẻ tấn công có thể lợi dụng SSRF để tìm kiếm các dịch vụ đang hoạt động trong mạng nội bộ và chuẩn bị cho các cuộc tấn công tiếp theo.
- Tấn công từ chối dịch vụ (DoS): Gửi liên tiếp các yêu cầu có thể làm quá tải máy chủ đích hoặc dẫn đến từ chối dịch vụ.

III. KỊCH BẢN DEMO TẤN CÔNG VÀ GIẢI PHÁP

A. A05 - Security Misconfiguration (Cấu hình bảo mật sai)

1) Ý tưởng: thực hiện bằng phương pháp Clickjacking. Tạo ra một trang web với giao diện và đường link tương tự trang gốc. Giao diện của trang web giả này được xây dựng để người dùng khó nhận ra sự khác biệt so với trang chính thống, tạo cảm giác tin tưởng. Trên trang giả mạo, kẻ tấn công tạo một nút bấm giả, chẳng hạn "Nhận thưởng ngay" hoặc "Xác nhận thông tin nhận quà." Khi người dùng nhấn vào nút bấm giả mạo này, thao tác thực sự được thực hiện trên iframe của trang hợp lệ. Điều này

có nghĩa là người dùng không nhận thức được rằng họ đang thực hiện các thao tác trên tài khoản của mình hoặc đang cung cấp quyền truy cập cho hacker.

2) Giải pháp: Để ngăn ngừa lỗ hổng Security Misconfiguration (cấu hình bảo mật sai), cần thiết lập và kiểm tra cấu hình bảo mật ngay từ giai đoạn đầu khi triển khai hệ thống hoặc ứng dụng. Điều này bao gồm việc thay đổi các cài đặt mặc định, vô hiệu hóa các dịch vụ không cần thiết và sử dụng tường lửa để bảo vệ hệ thống khỏi các kết nối không mong muốn. Thực hiện kiểm tra cấu hình bảo mật thường xuyên và cập nhật phần mềm để vá các lỗ hổng bảo mật cũng rất quan trọng. Ngoài ra, cần áp dụng nguyên tắc "least privilege" trong quản lý quyền truy cập và sử dụng các công cụ tự động như SIEM để theo dõi các sự kiện bảo mật. Đảm bảo rằng dữ liệu nhạy cảm được mã hóa và lưu trữ an toàn, đồng thời thiết lập chính sách bảo mật nghiêm ngặt và đào tạo nhân viên để tuân thủ các quy trình bảo mật. Bằng cách thực hiện những biện pháp này, hệ thống sẽ giảm thiểu được nguy cơ bị tấn công qua các cấu hình bảo mật sai.

3) Cụ thể trong demo:

```
const helmet = require('helmet'); // Import helmet
// Sử dụng helmet trước khi phục vụ nội dung tĩnh
app.use(helmet({
        frameguard: { action: 'deny' } // Thiết lập X-Frame-Options: DENY
}));
```

B. A07 - Identification and Authentication Failures (Lỗi trong xác thực và nhận dạng)

1) Ý tưởng: Nếu ứng dụng không giới hạn số lần nhập mật khẩu sai, kẻ tấn công có thể thực hiện phương pháp brute-force để đoán mật khẩu và chiếm quyền truy cập vào tài khoản. Việc lưu trữ mật khẩu dưới dạng plain-text (không mã hóa) sẽ làm tăng nguy cơ rò rỉ mật khẩu trong trường hợp cơ sở dữ liệu bị xâm nhập. Ngoài ra, nếu session không được bảo vệ đúng cách, kẻ tấn công có thể chiếm đoạt phiên làm việc của người dùng thông qua các cuộc tấn công XSS (Cross-site Scripting) hoặc hijacking session.

2) Giải pháp:

- Kiểm thử brute-force (Tấn công đăng nhập bằng cách thử mật khẩu nhiều lần): Sử dụng Flask-Limiter để giới hạn số lần đăng nhập sai. Ví dụ, giới hạn số lần đăng nhập là 5 lần trong 1 phút. Nếu người dùng nhập sai quá 5 lần, tài khoản sẽ bị khóa trong 1 phút.
- Kiểm thử lưu mật khẩu dạng plain-text: Sử dụng bcrypt để mã hóa mật khẩu trước khi lưu vào cơ sở dữ liệu. Đây là cách mã hóa mạnh mẽ và an toàn, giúp bảo vệ mật khẩu ngay cả khi dữ liệu bị xâm nhập.
- Kiểm thử bảo mật phiên (Session Security): Cấu hình bảo mật cho các session cookies, bao gồm SESSION_COOKIE_SECURE,SESSION_COOKIE_HTTPONLY, và SESSION_COOKIE_SAMESITE để tăng cường bảo mật và ngăn ngừa các cuộc tấn công qua JavaScript hoặc mạng không an toàn.

3) Cụ thể trong demo:

- Kiểm thử brute-force (Tấn công đăng nhập bằng cách thử mật khẩu nhiều lần):

```
from flask_limiter import Limiter
limiter = Limiter(app)
@app.route('/login', methods=['POST'])
@limiter.limit("5 per minute")  # Giới hạn 5 lần đăng nhập mỗi phút
def login():
    # Xử lý đăng nhập
```

- Kiểm thử lưu mật khẩu dạng plain-text:

```
from bcrypt import hashpw, gensalt
hashed_password=hashpw(password.encode('utf-8'), gensalt())
```

- Kiểm thử bảo mật phiên (Session Security):

```
app.config["SESSION_COOKIE_SECURE"] = True  # Chỉ gửi cookie qua HTTPS
app.config["SESSION_COOKIE_HTTPONLY"] = True  # Chặn JavaScript truy cập cookie
app.config["SESSION_COOKIE_SAMESITE"] = "Lax"  # Giảm nguy cơ cross-site attack
```

C. A08 - Software and Data Integrity Failures (Lỗi về tính toàn vẹn của phần mềm và dữ liệu)

1) Ý tưởng:

- Kiểm thử upload file không an toàn: Nếu hệ thống không kiểm tra kỹ lưỡng loại file được tải lên, người dùng có thể tải lên các file độc hại (ví dụ: mã Python, PHP) để thực hiện tấn công.
- Kiểm tra tính toàn vẹn thư viện: Việc sử dụng thư viện không rõ nguồn gốc hoặc không được kiểm tra tính toàn vẹn có thể làm tăng nguy cơ bị tấn công.
- Kiểm thử Dependency Injection (Injection thông qua dependency không đáng tin cậy): Nếu ứng dụng sử dụng các thư viện bên ngoài không đáng tin cậy hoặc không được kiểm tra kỹ lưỡng, có thể xảy ra injection thông qua dependency.

2) Giải pháp:

- Kiểm thử upload file không an toàn: Để bảo vệ ứng dụng khỏi các tấn công liên quan đến việc tải lên file, chúng ta chỉ cho phép các loại file hợp lệ như .txt, .jpg, .png và kiểm tra MIME type của file. Ngoài ra, sử dụng werkzeug.utils.secure_filename để đảm bảo tên file an toàn, tránh ký tự nguy hiểm. Cuối cùng, giới hạn kích thước file tải lên để ngăn ngừa các tấn công từ chối dịch vụ (DoS). Những biện pháp này giúp bảo vệ ứng dụng khỏi các mối đe dọa tiềm ẩn từ file tải lên.
- Kiểm tra tính toàn vẹn thư viện: Để đảm bảo tính toàn vẹn của các thư viện trong dự án, chúng ta sử dụng công cụ pip-audit để kiểm tra các thư viện đã cài đặt trong môi trường Python, giúp phát hiện các lỗ hổng bảo mật hoặc thư viện không đáng tin cậy. Thêm vào đó, việc khóa phiên bản thư viện trong tệp requirements.txt là một biện pháp quan trọng để đảm bảo rằng ứng dụng chỉ sử dụng các phiên bản ổn định và đã được kiểm tra, tránh việc sử dụng các thư viện không rõ nguồn gốc hoặc có thể gây ra sự cố bảo mật.
- Kiểm thử Dependency Injection (Injection thông qua dependency không đáng tin cậy): Để đảm bảo tính bảo mật khi cài đặt các thư viện bên ngoài, chúng ta sử dụng cờ --only-binary :all: để chỉ cài đặt các phiên bản nhị phân đã được xác minh, thay vì cài đặt từ mã nguồn có thể chứa mã độc. Điều này giúp giảm thiểu nguy cơ sử dụng các thư viện không đáng tin cậy. Bên cạnh đó, việc sử dụng công cụ pip-audit để quét các dependency giúp phát hiện các lỗ hổng bảo mật tiềm ẩn trong các thư viện đã cài đặt, từ đó bảo vệ ứng dụng khỏi các mối đe dọa từ các thư viện không an toàn.

3) Cụ thể trong demo:

- Kiểm thử upload file không an toàn:

  ```
  from werkzeug.utils import secure_filename
  filename = secure_filename(file.filename)
  ```
- Kiểm tra tính toàn vẹn thư viện: Khóa phiên bản thư viện (flask==2.0.3 werkzeug==2.0.3)
- Kiểm thử Dependency Injection (Injection thông qua dependency không đáng tin cậy): Sử dụng pip-audit để quét các dependency và phát hiện các lỗ hổng bảo mật (pip install fake_dependency --only-binary :all:)

D. A09 - Security Logging and Monitoring Failures (Lỗi trong ghi nhật ký và giám sát bảo mật)

1) Ý tưởng: Lợi dụng việc không ghi lại log của một website để tấn công. Dùng burp suite để demo một cuộc tấn công website login_vul thông qua lỗ hổng bảo mật log missing.

2) Giải pháp: Để đảm bảo hệ thống bảo mật hiệu quả, việc thiết lập và quản lý hệ thống ghi nhật ký là rất quan trọng. Trước tiên, cần ghi lại tất cả các sự kiện quan trọng như đăng nhập, đăng xuất, truy cập tài nguyên, thay đổi cấu hình, lỗi và cảnh báo, kèm theo thông tin chi tiết như địa chỉ IP, thời gian, người dùng, hành động và kết quả. Các nhật ký này cần được lưu trữ an toàn, tránh bị tấn công. Tiếp theo, việc phân tích nhật ký thường xuyên là rất cần thiết; các công cụ SIEM có thể giúp thu thập, phân tích và trực quan hóa nhật ký để phát hiện các hoạt động bất thường. Cần thiết lập các cảnh báo tự động khi có sự kiện đáng ngờ. Đồng thời, việc mã hóa nhật ký chứa thông tin nhạy cảm như mật khẩu và token xác thực là điều kiện cần để bảo vệ dữ liệu. Để đối phó với các sự cố an ninh, các sự kiện được xác định rõ ràng và quy trình ứng phó cần được thiết lập chi tiết, đồng thời diễn tập thường xuyên để kiểm tra hiệu quả. Cuối cùng, việc đào tạo nhân viên về nhận thức bảo mật và kỹ năng phát hiện, báo cáo các hoạt động bất thường sẽ giúp nâng cao khả năng bảo vệ hệ thống và ngăn ngừa các mối đe dọa.

3) Cụ thể trong demo: Thêm log vào các sự kiện quan trọng, đảm bảo đăng nhập, kiểm soát truy cập và xác thực đầu vào phía máy chủ được ghi lại. Dùng burp suite để demo cách ngăn chặn, hạn chế lỗ hổng bảo mật A09 thông qua việc tấn công website login.

E. A10 - Server-Side Request Forgery (SSRF) (Giả mạo yêu cầu phía máy chủ)

1) Ý tưởng: Lợi dụng lỗ hổng SSRF trong ứng dụng web, nơi người dùng có thể truy cập vào endpoint admin mà không có quyền (cho phép kẻ tấn công gửi yêu cầu từ máy chủ đến localhost, gây ra rủi ro bảo mật nghiêm trọng). Cụ thể việc sử dụng các dịch vụ như ngrok để mở rộng truy cập có thể tạo ra lỗ hổng bảo mật, khiến thông tin nhạy cảm bị lộ ra ngoài. Bằng cách sử dụng một yêu cầu đến một địa chỉ IP cụ thể, kẻ tấn công có thể vượt qua các rào cản bảo mật.

2) Giải pháp: Để hạn chế rủi ro và ngăn chặn các cuộc tấn công, cần thực hiện các biện pháp bảo mật toàn diện. Đầu tiên, việc kiểm tra và lọc dữ liệu đầu vào là rất quan trọng, bao gồm việc sử dụng whitelisting để chỉ cho phép các URL hợp lệ, blacklist để chặn các URL chứa từ khóa nguy hiểm, và sử dụng biểu thức chính quy (Regex) để kiểm tra định dạng của URL. Cũng cần xác thực xem

URL có trỏ đến các máy chủ được phép không. Thứ hai, hạn chế quyền truy cập bằng cách áp dụng nguyên tắc ít đặc quyền, phân tách mạng thành các vùng khác nhau, và sử dụng firewall để chặn các kết nối không đáng tin cậy. Quản lý lỗi cũng đóng vai trò quan trọng, tránh tiết lộ thông tin nhạy cảm qua các thông báo lỗi và ghi lại tất cả các yêu cầu HTTP để phục vụ điều tra. Để tăng cường bảo mật, cần sử dụng các công cụ như WAF (Web Application Firewall) để ngăn chặn các cuộc tấn công SSRF, SAST và DAST để phát hiện lỗ hổng trong mã nguồn và ứng dụng đang chạy. Cuối cùng, việc cập nhật phần mềm và vá các lỗ hổng là cần thiết để bảo vệ hệ thống khỏi các mối đe dọa mới.

3) Cụ thể trong demo:

```
def fetch():
    blacklist = [file://, 'localhost', '127.0.0.1'...]
    (whitelist = [])

    url ...
    request ...
return ...
```

IV. KẾT LUẬN

1) Đánh giá kết quả đạt được**:** Báo cáo đã triển khai và mô phỏng các lỗ hổng OWASP quan trọng, bao gồm A05, A07, A08, A09, và A10. Mỗi lỗ hổng được trình bày chi tiết qua:
   - A05 (Cấu hình bảo mật sai): Xây dựng kịch bản demo bằng cách lợi dụng lỗ hổng clickjacking và đưa ra giải pháp sử với tiêu chí bảo mật như X-Frame-Options.
   - A07 (Lỗi trong xác thực và nhận dạng): Mô phỏng các hình thức tấn công như brute-force và bảo mật phiên người dùng, đồng thời đề xuất giải pháp cụ thể bằng cách áp dụng Flask-Limiter và mã hóa mật khẩu bằng bcrypt.
   - A08 (Lỗi về tính toàn vẹn của phần mềm và dữ liệu): Đưa ra các phương pháp kiểm tra an toàn khi tải tệp và quản lý dependency thông qua công cụ như pip-audit và các giải pháp bảo mật mã nguồn.
   - A09 (Lỗi trong ghi nhật ký và giám sát bảo mật): Mô phỏng sự thiếu hụt log trong website bằng công cụ như Burp Suite và đề xuất sử dụng SIEM để cải thiện việc giám sát bảo mật.
   - A10 (Giả mạo yêu cầu phía máy chủ - SSRF): Minh họa cách khai thác SSRF để truy cập trái phép vào tài nguyên nội bộ và áp dụng các biện pháp như kiểm tra whitelist/blacklist và phân quyền nghiêm ngặt.

2) Hạn chế:
   - Thiếu sự tích hợp công cụ tự động hóa: Mặc dù đề cập đến một số công cụ như HelmetJS và pip-audit, báo cáo chưa thực hiện đánh giá trên diện rộng bằng các giải pháp tự động hóa như Dynamic Application Security Testing (DAST) hoặc Static Application Security Testing (SAST).
   - Chưa phân tích tác động sâu hơn: Báo cáo mới chỉ dừng lại ở việc mô tả các giải pháp kỹ thuật mà chưa đưa ra các phân tích định lượng về mức độ giảm thiểu rủi ro hoặc lợi ích kinh tế khi áp dụng các giải pháp này.

3) Hướng phát triển:
   - Nghiên cứu và tích hợp thêm các công cụ SIEM, WAF để tăng cường khả năng phát hiện lỗ hổng.
   - Tập trung vào các kỹ thuật tự động hóa kiểm tra bảo mật, giúp rút ngắn thời gian kiểm thử và triển khai.

TÀI LIỆU THAM KHẢO

[1] OWASP Foundation, *OWASP Top 10 - 2021*. Available: https://owasp.org/Top10/.

[2] PortSwigger, "Using Burp Suite to Test for the OWASP Top Ten," *PortSwigger Web Security*. Available: https://portswigger.net/support/using-burp-to-test-for-the-owasp-top-ten.

[3] HelmetJS, "HelmetJS Documentation: Enhancing Security with HTTP Headers," Available: https://helmetjs.github.io/.

[4] Flask-Limiter, "Rate Limiting with Flask-Limiter," *Flask-Limiter Documentation*. Available: https://flask-limiter.readthedocs.io/.

[5] Python Software Foundation, "pip-audit: Python Dependency Auditing," Available: https://pypi.org/project/pip-audit/.