

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN, ĐHQG-HCM  
KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG



**BÁO CÁO ĐỒ ÁN MÔN HỌC**

***ĐỀ TÀI: SỬ DỤNG CODEQL ĐỂ TÌM VÀ KHAI THÁC LỖ HỔNG  
NOSQL INJECTION***

**Môn học:** NT213.P11.ANTT

**GVHD:** TS. Phạm Văn Hậu, Ngô Khánh Khoa

**Thực hiện bởi nhóm B10:**

Họ và tên	MSSV	Vai trò
Nguyễn Huỳnh Duy	22520330	Trưởng nhóm
Võ Nguyễn Thái Học	22520489	Thành viên
Võ Nhật Hoàng	22520481	Thành viên
Lê Vũ Ca	22520140	Thành viên

**Thời gian thực hiện:** T10/2024 – T11/2204

## MỤC LỤC

MỤC LỤC .....	2
Chương I. TỔNG QUAN. ....	3
1. Giới thiệu. ....	3
2. Mục đích nghiên cứu. ....	3
3. Đối tượng và phạm vi nghiên cứu. ....	3
4. Phương pháp nghiên cứu. ....	3
Chương II. CƠ SỞ LÝ THUYẾT. ....	4
1. Tìm hiểu về NoSQL và NoSQL Injection. ....	4
1.1. NoSQL là gì. ....	4
1.2. Tìm hiểu về NoSQL Injection. ....	5
2. Tìm hiểu về CodeQL. ....	8
2.1. CodeQL là gì. ....	8
2.2. Kiến trúc và các thành phần của CodeQL. ....	8
2.3. Cách CodeQL hoạt động. ....	11
2.4. Tìm hiểu về QL. ....	12
2.5. Một số cách triển khai CodeQL. ....	15
Chương III. CÀI ĐẶT CODEQL CLI. ....	16
Chương IV. SỬ DỤNG CODEQL & TRIỂN KHAI KỊCH BẢN. ....	18
1. Tạo qldatabase cho repo/source code mà chúng ta sẽ quét: ....	18
2. Thực hiện quét qldatabase vừa tạo này: ....	19
3. Thực hiện khai thác: ....	21
4. Thực hiện sửa chữa source code và tạo lại qldatabase: ....	23
5. Thực hiện quét “fixed.qldatabase”: ....	25
Chương V. KẾT LUẬN. ....	27
NGUỒN THAM KHẢO .....	28

## Chương I. TỔNG QUAN.

### 1. Giới thiệu.

Với sự phát triển nhanh chóng của công nghệ, bảo mật thông tin được lưu trữ và truyền đi trở thành một vấn đề cực kì quan trọng trong việc phát triển các hệ thống, ứng dụng, ... Các lỗ hổng liên quan đến web, cụ thể là các lỗ hổng biến như: XSS, XSRF, các loại Injection cùng rất nhiều lỗ hổng có thể ảnh hưởng đến toàn hệ thống, làm ảnh hưởng đến sự hoạt động bình thường của dịch vụ, ứng dụng. Các lỗ hổng này đến từ nhiều lí do khác nhau chẳng hạn như con người, quá trình phát triển, bất cẩn trong quá trình code, v.v. Đặc biệt là lỗ hổng **NoSQL Injection** khi các cơ sở dữ liệu NoSQL ngày càng được nhiều hệ thống, dự án, dịch vụ sử dụng.

Với mục tiêu, giảm thiểu và tránh các lỗ hổng trong quá trình phát triển, thiết lập dịch vụ, rất nhiều nhà nghiên cứu, doanh nghiệp, tập thể, cá nhân đã phát triển các công cụ phát hiện lỗ hổng, dựa trên lỗi logic, lỗi sử dụng hàm, hard code, ... Đặc biệt, công cụ **CodeQL**, một dự án phát hiện lỗ hổng dựa trên các CWE đã được phát hiện, đã giúp rất nhiều nhà phát triển phát hiện và sửa chữa kịp thời.

### 2. Mục đích nghiên cứu.

Mục đích của đề tài này là nghiên cứu chi tiết về CodeQL, cách cài đặt, sử dụng cũng như các hoạt động của CodeQL; điều gì làm nó dần trở nên phổ biến hơn, cũng như lí do gì mà CodeQL được github tích hợp vào hệ thống phát hiện lỗi ở dự án, repository. Từ đó, dùng CodeQL được cài đặt để phát hiện lỗ hổng NoSQL Injection trong một số dự án.

### 3. Đối tượng và phạm vi nghiên cứu.

Đối tượng nghiên cứu: Mã nguồn mở CodeQL, NoSQL Injection.

Phạm vi nghiên cứu: Tập trung vào tìm hiểu các thức hoạt động, cài đặt và sử dụng CodeQL để phát hiện lỗ hổng và khai thác lỗ hổng NoSQL Injection ở trong dịch vụ ứng dụng.

### 4. Phương pháp nghiên cứu.

Phương pháp nghiên cứu tài liệu: Tìm hiểu từ các document trên github docs, github blog, microsoft, repository chính thức của codeql và codeql-action.

Phương pháp thực nghiệm: Tiến hành cài đặt và triển khai CodeQL CLI trên máy local để kiểm chứng các lý thuyết đã nghiên cứu. Thực hiện các demo với các tính năng.

Phương pháp phân tích, đánh giá: Sau khi triển khai, tiến hành phân tích kết quả và đánh giá hiệu quả của CodeQL trong việc phát hiện các lỗ hổng, cụ thể NoSQL Injection. Đưa ra các nhận xét về ưu điểm và nhược điểm của CodeQL dựa trên thực nghiệm đã thực hiện.

## **Chương II. CƠ SỞ LÝ THUYẾT.**

### **1. Tìm hiểu về NoSQL và NoSQL Injection.**

#### **1.1. NoSQL là gì.**

- NoSQL (viết tắt của "Not Only SQL") là một loại cơ sở dữ liệu được thiết kế để xử lý dữ liệu phi cấu trúc hoặc bán cấu trúc, khác với cơ sở dữ liệu quan hệ truyền thống (SQL). Nó được tối ưu hóa để làm việc với khối lượng lớn dữ liệu, thường phân tán trên nhiều máy chủ, và hỗ trợ truy vấn linh hoạt hơn.
- Thuật ngữ NoSQL được giới thiệu lần đầu vào năm 1998 sử dụng làm tên gọi chung cho các lightweight open source relational database (cơ sở dữ liệu quan hệ nguồn mở nhỏ) nhưng không sử dụng SQL cho truy vấn. Vào năm 2009, Eric Evans, nhân viên của Rackspace giới thiệu lại thuật ngữ NoSQL trong một hội thảo về cơ sở dữ liệu nguồn mở phân tán. Thuật ngữ NoSQL đánh dấu bước phát triển của thế hệ database mới: distributed (phân tán) + non-relational (không ràng buộc). Đây là 2 đặc tính quan trọng nhất.

#### **a. Ưu điểm:**

- Tính linh hoạt: Cơ sở dữ liệu NoSQL thường cung cấp các sơ đồ linh hoạt giúp công đoạn phát triển nhanh hơn và có khả năng lặp lại cao hơn. Mô hình dữ liệu linh hoạt biến cơ sở dữ liệu NoSQL thành lựa chọn lý tưởng cho dữ liệu không được tổ chức thành cấu trúc hoặc có cấu trúc chưa hoàn chỉnh.
- Khả năng mở rộng: Cơ sở dữ liệu NoSQL thường được thiết kế để tăng quy mô bằng cách sử dụng các cụm phần cứng được phân phối thay vì tăng quy mô bằng cách bổ sung máy chủ mạnh và tốn kém. Một số nhà cung cấp dịch vụ đám mây xử lý các hoạt động này một cách không công khai dưới dạng dịch vụ được quản lý đầy đủ.
- Hiệu suất cao: Cơ sở dữ liệu NoSQL được tối ưu hóa cho các mô hình dữ liệu cụ thể và các mẫu truy cập. Chúng cho phép hiệu suất cao hơn so với khi bạn đang cố gắng thực hiện chức năng tương tự với cơ sở dữ liệu quan hệ.
- Tính sẵn sàng cao: NoSQL có thể phân tán dữ liệu trên nhiều máy chủ, giúp tăng khả năng sẵn sàng và khả năng phục hồi sau lỗi, qua đó giúp NoSQL trở thành lựa chọn lý tưởng cho các ứng dụng đòi hỏi tính khả dụng cao.

- Có các chức năng cao cấp: Cơ sở dữ liệu NoSQL cung cấp các API và kiểu dữ liệu cực kỳ thiết thực được xây dựng riêng cho từng mô hình dữ liệu tương ứng.

**b. Nhược điểm:**

- Tính nhất quán: NoSQL có thể không đảm bảo tính nhất quán dữ liệu ở mức độ cao như các cơ sở dữ liệu quan hệ. Điều này có nghĩa là có thể xảy ra tình trạng dữ liệu bị đọc không nhất quán trên các máy chủ khác nhau.
- Sự phức tạp: Việc quản lý và vận hành các hệ thống NoSQL có thể phức tạp hơn so với các cơ sở dữ liệu quan hệ. Bởi, NoSQL không có một mô hình dữ liệu và bộ công cụ quản trị thống nhất.
- Thiếu tiêu chuẩn: Hiện tại NoSQL chưa có tiêu chuẩn chung, dẫn đến sự khác biệt giữa các hệ thống NoSQL khác nhau, có thể gây khó khăn cho việc di chuyển dữ liệu giữa các hệ thống NoSQL khác nhau.

**c. Các loại NoSQL database:**

1. Key – value database
2. Document database
3. Graph database:
4. In-memory database
5. Search database

## 1.2. Tìm hiểu về NoSQL Injection.

**a. NoSQL Injection là gì?**

NoSQL injection là một lỗ hổng cho phép kẻ tấn công can thiệp vào các truy vấn mà ứng dụng gửi đến cơ sở dữ liệu NoSQL. Lỗ hổng này có thể giúp kẻ tấn công:

- Bỏ qua cơ chế xác thực hoặc các biện pháp bảo vệ.
- Trích xuất hoặc chỉnh sửa dữ liệu.
- Gây từ chối dịch vụ (DoS).
- Thực thi mã trên máy chủ.

**b. Các loại NoSQL Injection**

- **Syntax injection**
  - Xảy ra khi có thể phá vỡ cú pháp truy vấn NoSQL, cho phép kẻ tấn công chèn payload riêng.
  - Phương pháp này tương tự như SQL injection, nhưng tấn công khác biệt do NoSQL sử dụng nhiều loại cú pháp truy vấn và cấu trúc dữ liệu khác nhau.
- **Operator injection**

- Xảy ra khi có thể sử dụng các toán tử truy vấn của NoSQL để thao túng truy vấn.

### c. Syntax injection

- Ta có thể phát hiện lỗ hổng bằng cách thử phá vỡ cú pháp truy vấn thông qua việc gửi các chuỗi hoặc ký tự đặc biệt để kích hoạt lỗi hoặc hành vi bất thường nếu dữ liệu đầu vào không được lọc hoặc xử lý đúng cách.
- Ví dụ: Hãy xem xét một ứng dụng mua sắm hiển thị sản phẩm theo các danh mục khác nhau. Khi người dùng chọn danh mục đồ uống có ga, trình duyệt của họ yêu cầu URL sau:

❖ *https://insecure-website.com/product/lookup?category=fizzy*

Điều này khiến ứng dụng gửi truy vấn JSON để lấy các sản phẩm có liên quan từ bộ productsru tập trong cơ sở dữ liệu MongoDB:

❖ *this.category == 'fizzy'*

Thử chèn một chuỗi fuzz:

❖ *""`{\$Foo}\$Foo \xYZ*

Sử dụng chuỗi fuzz này để thực hiện đòn tấn công sau:

❖ *https://insecure-*

*website.com/product/lookup?category='%22%60%7b%0d%0a%3b%24Foo%7d%0d%0a%24Foo%20%5cxYZ%00*

Nếu điều này gây ra thay đổi so với phản hồi ban đầu thì có thể thông tin đầu vào của người dùng không được lọc hoặc khử trùng đúng cách.

### d. Operator injection

- Cơ sở dữ liệu NoSQL thường sử dụng toán tử truy vấn, cung cấp các cách để chỉ định các điều kiện mà dữ liệu phải đáp ứng để được đưa vào kết quả truy vấn. Ví dụ về toán tử truy vấn MongoDB bao gồm:
  - \$where- Phù hợp với các tài liệu thỏa mãn biểu thức JavaScript.
  - \$ne- Phù hợp với tất cả các giá trị không bằng một giá trị được chỉ định.
  - \$in- Phù hợp với tất cả các giá trị được chỉ định trong một mảng.
  - \$regex- Chọn các tài liệu có giá trị khớp với biểu thức chính quy đã chỉ định.
- Ta có thể chèn toán tử truy vấn để thao tác các truy vấn NoSQL. Để thực hiện việc này, hãy gửi các toán tử khác nhau một cách có hệ thống vào một loạt các đầu vào của người dùng, sau đó xem xét các phản hồi để tìm thông báo lỗi hoặc các thay đổi khác.
- Trong các thông báo JSON, ta có thể chèn các toán tử truy vấn dưới dạng các đối tượng lồng nhau. Ví dụ: {"username":"wiener"} trở thành {"username":{"\$ne":"invalid"}}.

- Đối với các đầu vào dựa trên URL, ta có thể chèn toán tử truy vấn thông qua tham số URL. Ví dụ: `username=wiener` trở thành `username[$ne]=invalid`. Nếu cách này không hiệu quả, ta có thể thử cách sau:
  1. Chuyển đổi phương thức yêu cầu từ GET thành POST.
  2. Đổi Content-Type tiêu đề thành `application/json`.
  3. Thêm JSON vào nội dung tin nhắn.
  4. Chèn toán tử truy vấn vào JSON.
- Cách thực hiện tấn công:
  1. Hãy xem xét một ứng dụng dễ bị tấn công chấp nhận tên người dùng và mật khẩu trong nội dung POST yêu cầu:
 

```
{"username": "wiener", "password": "peter"}
```
  2. Kiểm tra từng đầu vào với một loạt toán tử. Ví dụ, để kiểm tra xem đầu vào tên người dùng có xử lý toán tử truy vấn hay không, ta có thể thử lệnh tiêm sau: `{"username": {"$ne": "invalid"}, "password": "peter"}`
  3. Nếu `$ne` toán tử được áp dụng, nó sẽ truy vấn tất cả người dùng có tên người dùng không bằng `invalid`. Nếu cả tên người dùng và mật khẩu đều được xử lý bởi toán tử, có thể bỏ qua xác thực bằng cách sử dụng payload sau:
 

```
{"username": {"$ne": "invalid"}, "password": {"$ne": "invalid"}}.
```

 Truy vấn này trả về tất cả thông tin đăng nhập trong đó cả tên người dùng và mật khẩu đều không bằng `invalid`. Do đó, ta sẽ đăng nhập vào ứng dụng với tài khoản của user đầu tiên trong danh sách ta nhận được.
  4. Nếu muốn tìm cách truy cập vào một tài khoản cụ thể như `admin`, ta có thể xây dựng một payload bao gồm tên người dùng đã biết hoặc tên người dùng mà bạn đã đoán. Ví dụ:
 

```
{"username": {"$in": ["admin", "administrator", "superadmin"]}, "password": {"$ne": ""}}
```

#### e. Cách phòng ngừa NoSQL Injection

- Kiểm tra và lọc dữ liệu đầu vào, sử dụng allowlist gồm các ký tự được cho phép.
- Sử dụng WAF (Web Application Firewall).
- Chèn dữ liệu đầu vào của người dùng bằng các truy vấn có tham số thay vì nối dữ liệu đầu vào của người dùng trực tiếp vào truy vấn.
- Để ngăn chặn operator injection, áp dụng allowlist với danh sách các keys hợp lệ.

## 2. Tìm hiểu về CodeQL.

### 2.1. CodeQL là gì.

- CodeQL là công cụ phân tích được các nhà phát triển sử dụng để tự động kiểm tra bảo mật và được các nhà nghiên cứu bảo mật sử dụng để thực hiện phân tích biến thể.
- CodeQL là một công cụ phân tích mã tĩnh mạnh mẽ do Semmle phát triển (được GitHub mua lại vào năm 2019) và dựa trên hơn một thập kỷ nghiên cứu của một nhóm từ Đại học Oxford. CodeQL sử dụng phân tích luồng dữ liệu và phân tích vết bản để tìm lỗi mã, kiểm tra chất lượng mã và xác định lỗ hổng. Hiện tại, các ngôn ngữ được hỗ trợ bao gồm C/C++, C#, Go, Java, Kotlin, JavaScript, Python, Ruby, TypeScript và Swift.
- CodeQL dựa trên ngôn ngữ truy vấn mạnh mẽ được gọi là QL. Hiểu QL giúp cho ta có cái nhìn tốt hơn về việc đọc hiểu cũng như viết mã phân tích với CodeQL.
- Trong CodeQL, mã nguồn được xử lý như dữ liệu. Các lỗ hổng bảo mật, lỗi, và các vấn đề khác được mô hình hóa dưới dạng các truy vấn có thể thực thi trên cơ sở dữ liệu được trích xuất từ mã nguồn. Bạn có thể chạy các truy vấn CodeQL tiêu chuẩn được viết bởi các nhà nghiên cứu của GitHub và cộng đồng, hoặc tự viết truy vấn của riêng bạn để thực hiện các phân tích tùy chỉnh. Các truy vấn tìm lỗi tiềm năng sẽ đánh dấu kết quả trực tiếp trong tệp mã nguồn.

### 2.2. Kiến trúc và các thành phần của CodeQL.

#### a. Phân tích biến thể (Variant analysis)

- Phân tích biến thể là quá trình sử dụng một lỗ hổng bảo mật đã biết làm gốc để tìm các vấn đề tương tự trong mã của bạn. Đây là một kỹ thuật mà các kỹ sư bảo mật sử dụng để xác định các lỗ hổng tiềm năng và đảm bảo những mối đe dọa này được khắc phục đúng cách trên nhiều mã nguồn.
- Sử dụng CodeQL để truy vấn mã là cách hiệu quả nhất để thực hiện phân tích biến thể. Bạn có thể dùng các truy vấn CodeQL tiêu chuẩn để xác định lỗ hổng gốc, hoặc tìm lỗ hổng mới bằng cách viết các truy vấn CodeQL tùy chỉnh của riêng bạn. Sau đó, bạn có thể phát triển hoặc cải thiện truy vấn để tự động tìm các biến thể logic của cùng một lỗi mà có thể bị bỏ sót khi sử dụng các kỹ thuật thủ công truyền thống.

#### b. Cơ sở dữ liệu CodeQL (CodeQL databases)

- Cơ sở dữ liệu CodeQL chứa dữ liệu có thể truy vấn được trích xuất từ một mã nguồn, cho một ngôn ngữ cụ thể tại một thời điểm cụ thể. Cơ sở dữ liệu



này bao gồm một đại diện phân cấp đầy đủ của mã nguồn, bao gồm cây cú pháp trừu tượng (AST), đồ thị luồng dữ liệu, và đồ thị luồng điều khiển

- Mỗi ngôn ngữ có một lược đồ cơ sở dữ liệu riêng biệt xác định các mối quan hệ được sử dụng để tạo cơ sở dữ liệu. Lược đồ này cung cấp giao diện giữa phân tích từ vựng ban đầu trong quá trình trích xuất và phân tích phức tạp của trình đánh giá truy vấn CodeQL. Ví dụ, trong cơ sở dữ liệu CodeQL cho một chương trình Java, có hai bảng quan trọng:
  - **Bảng biểu thức:** Chứa một hàng cho mỗi biểu thức trong mã nguồn đã được phân tích trong quá trình xây dựng.
  - **Bảng câu lệnh:** Chứa một hàng cho mỗi câu lệnh trong mã nguồn đã được phân tích trong quá trình xây dựng.
- Thư viện CodeQL định nghĩa các lớp để cung cấp một lớp trừu tượng trên các bảng này, giúp dễ dàng viết các truy vấn.

### c. Các truy vấn (CodeQL queries)

- Là các truy vấn được viết bằng ngôn ngữ QL (Query Language) để phân tích mã nguồn, phát hiện các lỗi bảo mật, lỗi logic, và các vấn đề tiềm ẩn trong phần mềm.
- Truy vấn chuẩn (Standard Queries):
  - Được cung cấp bởi GitHub và cộng đồng.
  - Tập trung vào các lỗi bảo mật phổ biến hoặc các vấn đề chất lượng mã.
- Truy vấn tùy chỉnh (Custom Queries):
  - Được viết bởi người dùng để đáp ứng các nhu cầu phân tích cụ thể.
  - Ví dụ: Tìm các cách sử dụng API hoặc kiểm tra mã tuân thủ các tiêu chuẩn nội bộ.

### d. Bộ truy vấn (Query Suites)

- Là tập hợp các truy vấn CodeQL được nhóm lại để phục vụ cho các phân tích cụ thể. Thay vì phải chạy từng truy vấn riêng lẻ, query suite cho phép bạn chọn và chạy nhiều truy vấn cùng lúc, giúp tiết kiệm thời gian và cải thiện hiệu quả phân tích
- Định nghĩa bộ truy vấn được lưu trữ trong các tệp YAML có phần đuôi là “.qls”. File này chứa một chuỗi các lệnh trong đó mỗi lệnh là một ánh xạ YAML với (thường là) một khóa duy nhất. Các lệnh được thực thi theo thứ tự xuất hiện trong định nghĩa bộ truy vấn. Sau khi tất cả các lệnh trong định nghĩa bộ đã được thực thi, kết quả là một tập hợp các truy vấn đã chọn.
- CodeQL cung cấp hai query suites mặc định:
  - **Default Query Suite:**
    - Được sử dụng mặc định trong quá trình quét mã trên GitHub.

- Chứa các truy vấn bảo mật có độ chính xác cao, ít kết quả sai.
- **Security-Extended Query Suite:**
  - Bao gồm các truy vấn của default và thêm các truy vấn bảo mật khác.
  - Tập trung vào tìm kiếm thêm các lỗ hổng nhưng có thể trả về một số kết quả sai (false positives) nhiều hơn.

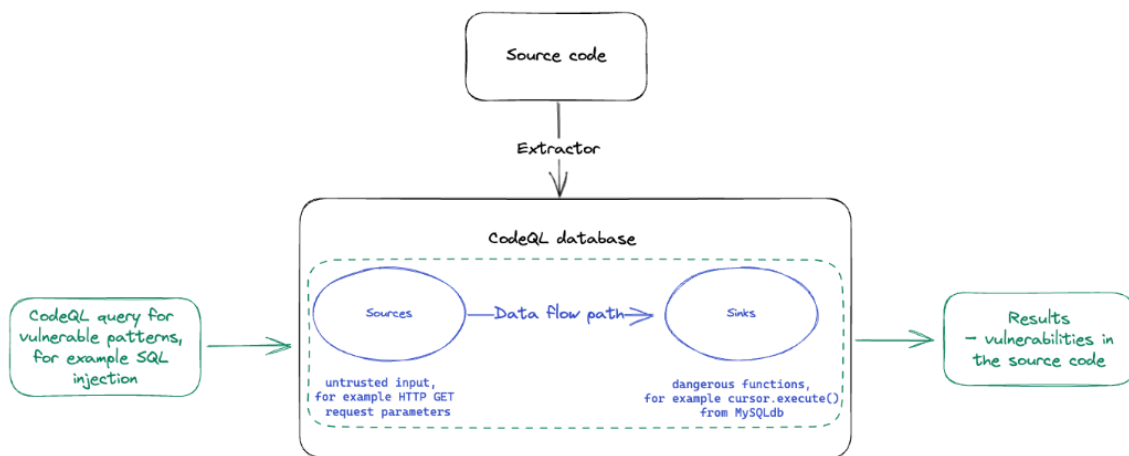
#### e. CodeQL packs

- Gói CodeQL được sử dụng để tổ chức các tệp dùng trong phân tích CodeQL, giúp dễ dàng tạo, chia sẻ, phụ thuộc, và chạy các truy vấn và thư viện CodeQL một cách dễ dàng. Chúng chứa các truy vấn, tệp thư viện, bộ truy vấn và metadata quan trọng. Với các gói CodeQL và các lệnh quản lý gói trong CodeQL CLI, ta có thể xuất bản các truy vấn tùy chỉnh của mình và tích hợp chúng vào phân tích cơ sở mã mong muốn.
- Có ba loại gói CodeQL:
  - **Gói truy vấn:** Được thiết kế để thực thi, bao gồm tất cả các phụ thuộc và biểu diễn tiền biên dịch của từng truy vấn.
  - **Gói thư viện:** Được sử dụng bởi các gói truy vấn hoặc gói thư viện khác, không chứa truy vấn.
  - **Gói mô hình:** Mở rộng phân tích quét mã để bao gồm các phụ thuộc không được hỗ trợ mặc định (đang ở giai đoạn beta).
- Cấu trúc của một gói được định nghĩa trong tệp “qlpack.yml”. Nội dung của gói CodeQL (các truy vấn hoặc thư viện được sử dụng trong phân tích CodeQL) được bao gồm trong cùng thư mục với qlpack.yml, hoặc các thư mục con của nó.
- Ví dụ về tệp “qlpack.yml”:

yml

```
name: codeql/java-queries
version: 0.0.6-dev
groups: java
suites: codeql-suites
extractor: java
defaultSuiteFile: codeql-suites/java-code-scanning.qls
dependencies:
  codeql/java-all: "*"
  codeql/suite-helpers: "*"
```

### 2.3. Cách CodeQL hoạt động.



- CodeQL analysis bao gồm ba bước:
  1. Chuẩn bị mã bằng cách tạo cơ sở dữ liệu CodeQL.
  2. Chạy truy vấn CodeQL trên cơ sở dữ liệu.
  3. Diễn giải kết quả truy vấn.
- a. **Tạo cơ sở dữ liệu CodeQL (Database creation)**
  - Để tạo cơ sở dữ liệu, CodeQL trước tiên trích xuất một biểu diễn quan hệ duy nhất của từng tệp mã nguồn trong dự án.
    - **Đối với các ngôn ngữ biên dịch (compiled languages):**
      - Việc trích xuất được thực hiện bằng cách theo dõi quá trình biên dịch thông thường. Mỗi lần trình biên dịch xử lý một tệp mã nguồn, một bản sao của tệp đó được tạo và tất cả thông tin liên quan về mã nguồn được thu thập. Thông tin này bao gồm:
        - **Dữ liệu cú pháp (syntactic data):** Cây cú pháp trừu tượng (Abstract Syntax Tree - AST).
        - **Dữ liệu ngữ nghĩa (semantic data):** Liên kết tên (name binding) và thông tin kiểu dữ liệu.
    - **Đối với các ngôn ngữ thông dịch (interpreted languages):** Bộ trích xuất (extractor) chạy trực tiếp trên mã nguồn, giải quyết các phụ thuộc để cung cấp một biểu diễn chính xác của toàn bộ mã nguồn.
    - **Bộ trích xuất theo ngôn ngữ:** Mỗi ngôn ngữ được CodeQL hỗ trợ có một bộ trích xuất riêng để đảm bảo quá trình trích xuất chính xác

nhất. Đối với dự án sử dụng nhiều ngôn ngữ, cơ sở dữ liệu sẽ được tạo riêng cho từng ngôn ngữ một.

- **Kết quả trích xuất:** Sau khi trích xuất, tất cả dữ liệu cần thiết cho việc phân tích (dữ liệu quan hệ, các tệp mã nguồn sao chép, và lược đồ cơ sở dữ liệu đặc thù của ngôn ngữ xác định các mối quan hệ dữ liệu) được nhập vào một thư mục duy nhất, được gọi là cơ sở dữ liệu CodeQL (CodeQL database).

#### b. Thực thi truy vấn (Query execution)

- Sau khi tạo cơ sở dữ liệu CodeQL, một hoặc nhiều truy vấn sẽ được thực thi trên cơ sở dữ liệu đó. Các truy vấn CodeQL được viết bằng ngôn ngữ truy vấn hướng đối tượng đặc biệt gọi là QL.
- Ta có thể chạy các truy vấn được kiểm tra từ kho lưu trữ CodeQL (hoặc các truy vấn tùy chỉnh mà bạn tự viết) bằng tiện ích mở rộng CodeQL cho VS Code hoặc CodeQL CLI .

#### c. Kết quả truy vấn (Query results)

- Bước cuối cùng là chuyển đổi kết quả từ quá trình thực thi truy vấn thành dạng dễ hiểu hơn trong ngữ cảnh mã nguồn, làm nổi bật các vấn đề tiềm ẩn mà truy vấn được thiết kế để tìm kiếm.
- Hiển thị kết quả:
  - Một số truy vấn hiển thị thông báo đơn giản tại một vị trí trong mã nguồn.
  - Một số khác hiển thị chuỗi vị trí đại diện cho các bước trong luồng dữ liệu hoặc luồng điều khiển, cùng thông báo giải thích ý nghĩa của kết quả.
- Metadata của truy vấn:
  - Các truy vấn chứa thuộc tính metadata chỉ định cách diễn giải kết quả.
  - Truy vấn không có metadata sẽ không được diễn giải; kết quả sẽ được hiển thị dưới dạng bảng và không hiển thị trong mã nguồn.
- Xuất kết quả: Sau khi diễn giải, kết quả được xuất ra để xem xét và xử lý.
  - Trong CodeQL for Visual Studio Code, kết quả truy vấn được hiển thị tự động trong mã nguồn.
  - Kết quả từ CodeQL CLI có thể được xuất sang nhiều định dạng khác nhau để sử dụng với các công cụ khác.

## 2.4. Tìm hiểu về QL.

### a. QL là gì

- QL là một ngôn ngữ truy vấn khai báo, hướng đối tượng, được tối ưu hóa để phân tích hiệu quả các cấu trúc dữ liệu phân cấp; đặc biệt là các cơ sở dữ liệu đại diện cho các sản phẩm phần mềm.
- Mục đích của một ngôn ngữ truy vấn là cung cấp một nền tảng lập trình, nơi ta có thể đặt câu hỏi về thông tin được lưu trữ trong cơ sở dữ liệu. Hệ thống quản lý cơ sở dữ liệu quản lý việc lưu trữ và quản trị dữ liệu và cung cấp cơ chế truy vấn. Một truy vấn thường tham chiếu đến các thực thể cơ sở dữ liệu có liên quan và chỉ định các điều kiện khác nhau (gọi là các mệnh đề) mà kết quả phải thỏa mãn. Đánh giá truy vấn liên quan đến việc kiểm tra các mệnh đề này và tạo ra kết quả.

#### **b. QL syntax**

- Cú pháp của QL tương tự như SQL, nhưng ngữ nghĩa của QL dựa trên Datalog, một ngôn ngữ lập trình logic khai báo thường được sử dụng như một ngôn ngữ truy vấn. Điều này khiến QL chủ yếu là một ngôn ngữ logic, và tất cả các thao tác trong QL đều là các thao tác logic. Hơn nữa, QL kế thừa các mệnh đề đệ quy từ Datalog và thêm hỗ trợ cho các phép toán tổng hợp, giúp các truy vấn phức tạp trở nên ngắn gọn và đơn giản.
- Ví dụ, xem xét một cơ sở dữ liệu chứa các mối quan hệ cha-con cho các cá nhân. Nếu ta muốn tìm số lượng hậu duệ của một người, thông thường ta sẽ:
  - Tìm một hậu duệ của người đó; tức là, con hoặc một hậu duệ của con.
  - Đếm số lượng hậu duệ tìm được từ bước trước.
- Khi ta viết quá trình này trong QL, nó gần giống với cấu trúc trên. Lưu ý rằng ví dụ này sử dụng đệ quy để tìm tất cả các hậu duệ của người được cho, và một phép toán tổng hợp để đếm số lượng hậu duệ. Việc chuyển các bước này thành truy vấn cuối cùng mà không thêm chi tiết thủ tục là khả thi nhờ vào tính khai báo của ngôn ngữ. Mã QL có thể trông giống như sau:

ql

```

Person getADescendant(Person p) {
    result = p.getAChild() or
    result = getADescendant(p.getAChild())
}

int getNumberOfDescendants(Person p) {
    result = count(getADescendant(p))
}

```

### c. Tính hướng đối tượng

- Hướng đối tượng là một tính năng quan trọng của QL. Các lợi ích của hướng đối tượng là điều đã được biết đến rộng rãi: nó tăng tính mô-đun, cho phép ẩn thông tin và cho phép tái sử dụng mã nguồn. QL cung cấp tất cả các lợi ích này mà không làm giảm nền tảng logic của nó. Điều này được thực hiện bằng cách định nghĩa một mô hình đối tượng đơn giản, trong đó các lớp được mô phỏng như các mệnh đề và kế thừa là sự suy luận. Các thư viện có sẵn cho tất cả các ngôn ngữ được hỗ trợ sử dụng rộng rãi các lớp và kế thừa.

### d. QL và ngôn ngữ lập trình mục đích chung

Dưới đây là một số sự khác biệt về khái niệm và chức năng giữa các ngôn ngữ lập trình mục đích chung và QL:

- QL không có bất kỳ tính năng mệnh lệnh nào như gán giá trị cho biến hoặc các thao tác trên hệ thống tập tin.
- QL hoạt động trên các tập hợp bộ ba và một truy vấn có thể được coi là một chuỗi phức tạp của các thao tác trên tập hợp định nghĩa kết quả của truy vấn.
- Ngữ nghĩa dựa trên tập hợp của QL khiến việc xử lý các bộ giá trị trở nên rất tự nhiên mà không cần lo lắng về việc lưu trữ, lập chỉ mục và duyệt qua chúng một cách hiệu quả.
- Trong các ngôn ngữ lập trình hướng đối tượng, việc khởi tạo một lớp bao gồm việc tạo một đối tượng bằng cách cấp phát bộ nhớ vật lý để lưu trữ trạng thái của đối tượng đó. Trong QL, các lớp chỉ là những đặc tính logic mô tả các tập hợp giá trị đã tồn tại.

## 2.5. Một số cách triển khai CodeQL.

### a. Sử dụng CodeQL qua GitHub Actions

- GitHub Actions cho phép tự động hóa quy trình phân tích bảo mật và kiểm tra mã nguồn trực tiếp từ kho lưu trữ GitHub của bạn.
- **Cách triển khai:**
  - Thiết lập **CodeQL Action** trong một workflow của GitHub Actions để tự động phân tích mã nguồn.
  - CodeQL tự động chạy và phân tích mã trong kho lưu trữ của bạn mỗi khi có thay đổi (pull request, push).
- **Ưu điểm:**
  - Dễ dàng tích hợp với GitHub.
  - Tự động hóa quy trình kiểm tra bảo mật và phân tích mã nguồn.

### b. Sử dụng CodeQL với Visual Studio Code (VS Code)

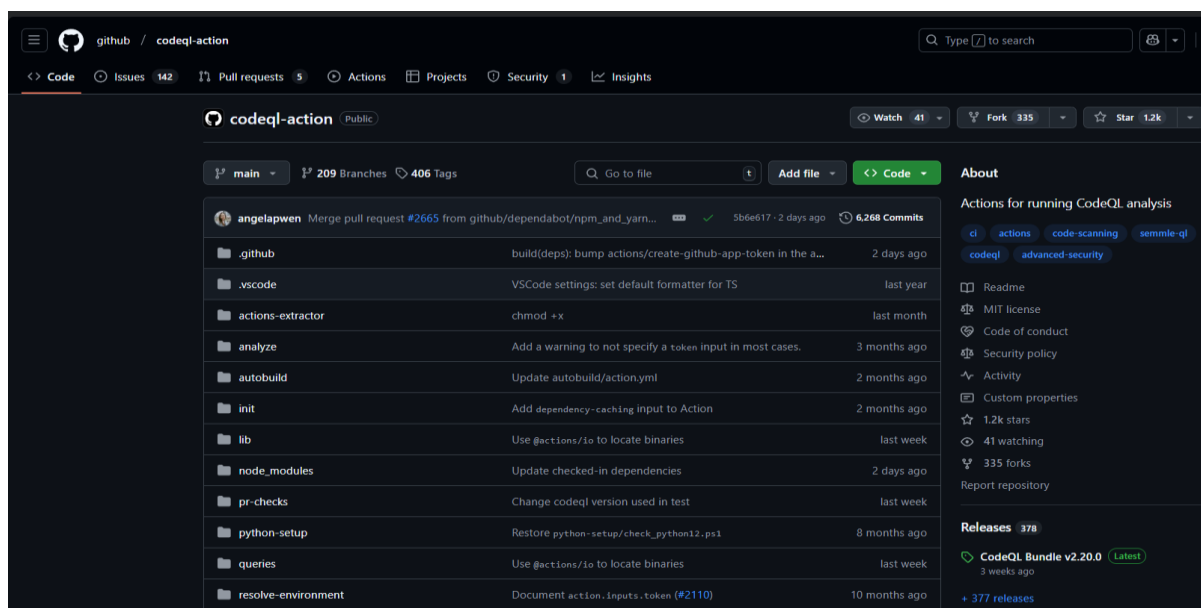
- Nếu bạn muốn sử dụng CodeQL trong môi trường phát triển tích hợp (IDE), **VS Code** cung cấp một tiện ích mở rộng để chạy CodeQL trực tiếp từ trong IDE.
- **Cách triển khai:**
  - Cài đặt **CodeQL extension** cho Visual Studio Code.
  - Mở dự án của bạn trong VS Code và chạy các truy vấn CodeQL để phân tích mã nguồn.
- **Ưu điểm:**
  - Quản lý mã nguồn trực tiếp từ IDE.
  - Dễ dàng viết và kiểm tra các truy vấn CodeQL.
  - Hiển thị kết quả phân tích trực tiếp trong mã nguồn.

### c. Sử dụng CodeQL qua Command Line Interface (CLI)

- CodeQL cung cấp một **CLI** giúp người dùng thực hiện phân tích mã nguồn và chạy truy vấn trực tiếp từ dòng lệnh.
- **Cách triển khai:**
  - Tải xuống và cài đặt **CodeQL CLI**.
  - Sử dụng các lệnh như “codeql database create” để tạo cơ sở dữ liệu, “codeql query run” để chạy các truy vấn, và “codeql database analyze” để phân tích kết quả.
- **Ưu điểm:**
  - Thích hợp cho các dự án không sử dụng GitHub.
  - Tự động hóa quy trình phân tích mã nguồn.

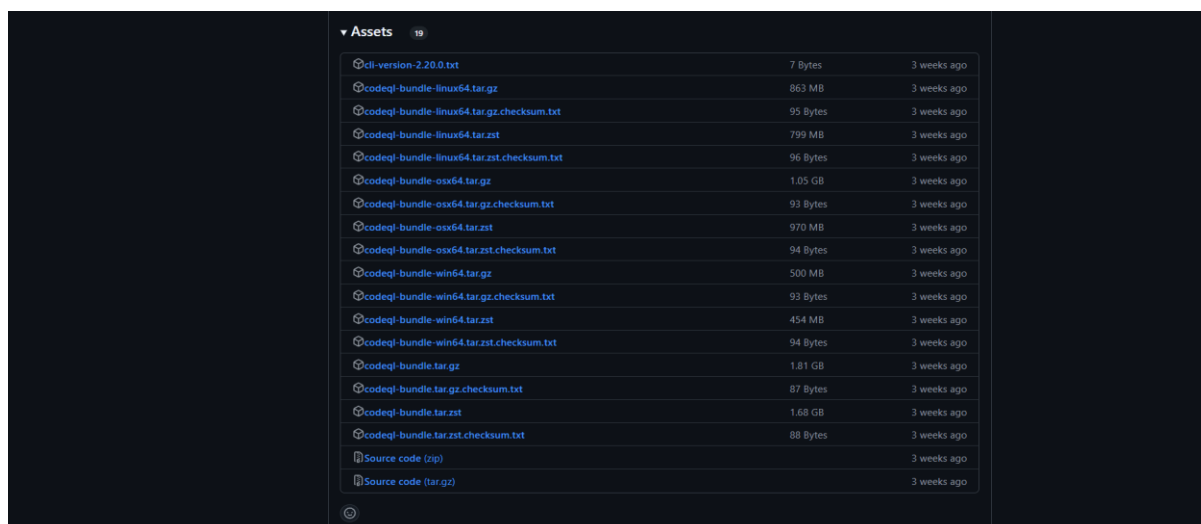
## Chương III. CÀI ĐẶT CODEQL CLI

### Repo Chính thức codeql-action



#### *Repo codeql-action*

Chúng ta sẽ tải source code cho CodeQL CLI tại releases của codeql-action, file “codeql-bundle-linux64.tar.zst”.



#### *Bundle-Asset*

### Giải nén và export PATH – Cụ thể trên arch linux 64



```
File Actions Edit View Help
# see /usr/share/doc/bash/examples/startup-files for examples.
# the files are located in the bash-doc package.

# the default umask is set in /etc/profile; for setting the umask
# for ssh logins, install and configure the libpam-umask package.
#umask 022

# if running bash
if [ -n "$BASH_VERSION" ]; then
    # include .bashrc if it exists
    if [ -f "$HOME/.bashrc" ]; then
        . "$HOME/.bashrc"
    fi
fi

# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/bin" ] ; then
    PATH="$HOME/bin:$PATH"
fi

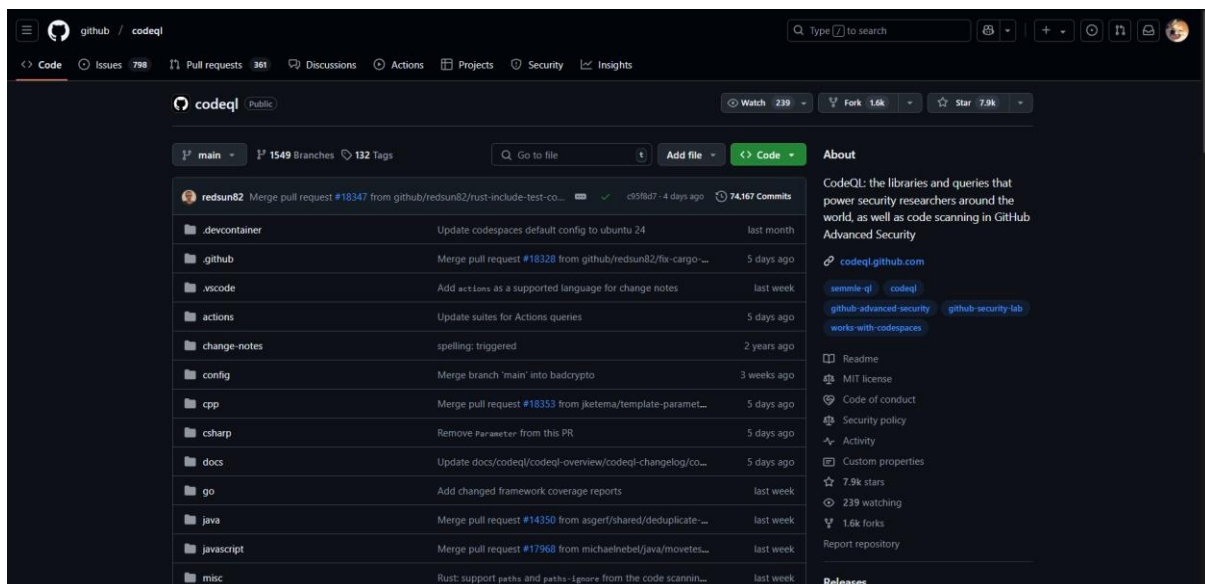
# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/.local/bin" ] ; then
    PATH="$HOME/.local/bin:$PATH"
fi

export GITROB_ACCESS_TOKEN=sqp_c824c564a1d108fd18e65e46ac7ccc1b205d9650
export PATH=$PATH:/home/kali/codeql-home/codeql

(kali@kali) - [~/codeql-home]
$ ls
codeql  codeql-bundle-linux64.tar.zst  codeql-repo
```

*Giải nén file zst vừa tải + export PATH*

Git clone codeql từ repo chính thức về đặt tên là codeql-repo, repo này chứa các file .ql và .qls hỗ trợ codeql cli về cách phát hiện lỗ hổng dựa trên các CWE.



*Repo chính thức của codeql*

**Kiểm tra codeql đã sử dụng được chưa**

```

$ ls
codeql  codeql-bundle-linux64.tar.zst  codeql-repo

(kali@kali) - [~/codeql-home]
$ codeql
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Usage: codeql <command> <argument> ...
Create and query CodeQL databases, or work with the QL language.

GitHub makes this program freely available for the analysis of open-source software and certain other uses, but it is not itself free software. Type codeql
--license to see the license terms.

--license          Show the license terms for the CodeQL toolchain.
Common options:
-h, --help          Show this help text.
-v, --verbose        Incrementally increase the number of progress messages printed.
-q, --quiet          Incrementally decrease the number of progress messages printed.
Some advanced options have been hidden; try --help -v for a fuller view.
Commands:
query              Compile and execute QL code.
bqrs               Get information from .bqrs files.
database            Create, analyze and process CodeQL databases.
dataset            [Plumbing] Work with raw QL datasets.
test               Execute QL unit tests.
resolve            [Deep plumbing] Helper commands to resolve disk locations etc.
execute            [Deep plumbing] Low-level commands that need special JVM options.
version            Show the version of the CodeQL toolchain.
generate           Commands that generate useful output.
github             Commands useful for interacting with the GitHub API through CodeQL.
pack               Commands to manage QL packages.
diagnostic          [Experimental] Create, process, and export diagnostic information.
    
```

*codeql cli*

## Chương IV. SỬ DỤNG CODEQL & TRIỂN KHAI KỊCH BẢN.

Source code: web challenge chứa lỗ hổng NoSQL Injection – “nosql\_nofix” (chưa sửa lỗ hổng) và “nosql\_fixed” (đã sửa lỗ hổng).

### 1. Tạo qldatabase cho repo/source code mà chúng ta sẽ quét:

Dùng lệnh:

“codeql database create <tên-database> --language=javascript --source-root=<đường-dẫn-tới-mã>”

```

(kali@kali) - [~/Documents/BaoMatWeb&UngDung/DoAn]
$ codeql database create nofixed.qldatabase --language=javascript --source-root ./nosqli_nofix
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Initializing database at /home/kali/Documents/BaoMatWeb&UngDung/DoAn/nofixed.qldatabase.
    
```

*Tạo database cho source code trước khi phân tích*

Extractor cho Javascript, được thiết kế để hiểu javascript, gồm:

- ES5, ES6+, ESNext.
- Các framework và thư viện phổ biến (React, Angular, Node.js).
- TypeScript (nếu có).

Extractor sử dụng parser để xây dựng một Abstract Syntax Tree – AST hay một biểu diễn trừu tượng của mã nguồn.

Các thông tin được biểu diễn dưới dạng các bảng dữ liệu quan hệ như: Classes, Functions, Variables, Control Flow, ... và lưu trữ ở trong file có đuôi .trap

```

[2024-12-26 09:54:11] [build-stdout] Done extracting /home/kali/codeql-home/codeql/javascript/tools/data/externs/vm/rhino.js (4 ms)
[2024-12-26 09:54:11] [build-stdout] Extracting /home/kali/codeql-home/codeql/javascript/tools/data/externs/vm/v8.js
[2024-12-26 09:54:11] [build-stdout] Done extracting /home/kali/codeql-home/codeql/javascript/tools/data/externs/vm/v8.js (5 ms)
[2024-12-26 09:54:11] [build-stdout] Extracting /home/kali/codeql-home/codeql/javascript/tools/data/externs/vm/jsshell.js
[2024-12-26 09:54:11] [build-stdout] Done extracting /home/kali/codeql-home/codeql/javascript/tools/data/externs/vm/jsshell.js (24 ms)
[2024-12-26 09:54:11] [build-stdout] Extracting /home/kali/codeql-home/codeql/javascript/tools/data/externs/vm/spidermonkey.js
[2024-12-26 09:54:11] [build-stdout] Done extracting /home/kali/codeql-home/codeql/javascript/tools/data/externs/vm/spidermonkey.js (2 ms)
Finalizing database at /home/kali/Documents/BaoMatWeb6UngDung/DoAn/nofixed_gldatabase.

```

*Extract các file .js thành các file .trap*

Sau khi được khởi tạo, codeql sẽ tổng hợp các file .trap này vào cơ sở dữ liệu. Cơ sở dữ liệu được tổ chức thành nhiều bảng, sẵn sàng cho truy vấn CodeQL.

```
Running TRAP import for CodeQL database at /home/kali/Documents/BaoMatWeb&UngDung/DoAn/nofixed.qldatabase ...
Importing TRAP files
Merging relations
Finished writing database (relations: 13.36 MiB; string pool: 4.78 MiB).
TRAP import complete (2.7s).
Finished zipping source archive (247.24 KiB).
Successfully created database at /home/kali/Documents/BaoMatWeb&UngDung/DoAn/nofixed.qldatabase.
```

*Tổng hợp lại ở nofixed.qldatabase*

## 2. Thực hiện quét qldatabase vừa tạo này:

Dùng lệnh: ``codeql database analyze --format=<format> --output=<output> <database's name>``

Trong đó:

- format: csv, sarifv2.1.0, sarif-latest, graphtext, dgml, dot
- output: tên file output

```
hoangvgn@ubuntu:~/Documents/BaoMatWeb/BoAn$ codeql database analyze --format=sarif-latest --output=nofixed_results.sarif nofixed.qldatabase
Running queries.
[1/99] Loaded /home/hoangvgn/codeql-home/codeql/qpacks/codeql/javascript-queries/1.2.5/Performance/ReDoS.qlx.
[2/99] Loaded /home/hoangvgn/codeql-home/codeql/qpacks/codeql/javascript-queries/1.2.5/Performance/PolynomialReDoS.qlx.
[3/99] Loaded /home/hoangvgn/codeql-home/codeql/qpacks/codeql/javascript-queries/1.2.5/AngularJS/DisablingSce.qlx.
```

*Thực hiện quét nofixed.qldatabase*

Trước khi quét, codeql sẽ load các file .qlx dùng để quét lên, gồm các lỗi hay gặp, các CWE đã biết. Sau đó, codeql sẽ dùng các file .ql đại diện để quét các lỗ hổng, weakness trong source code.

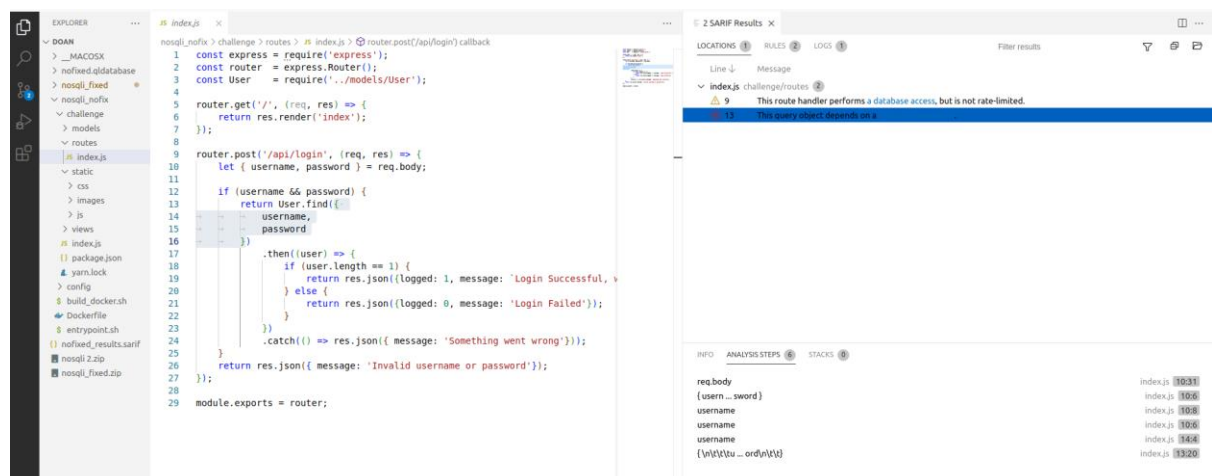
```
[1/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/Performance/ReDoS.qlx.
[2/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/Performance/PolynomialReDoS.qlx.
[3/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/AngularJS/DisablingSce.qlx.
[4/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/AngularJS/DoubleCompilation.qlx.
[5/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/AngularJS/InsecureUrlWhitelist.qlx.
[6/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/Security/CWE-400/DeepObjectResourceExhaustion.qlx.
[7/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/Security/CWE-346/CorsMisconfigurationForCredentials.qlx.
[8/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/Security/CWE-327/BrokenCryptoAlgorithm.qlx.
[9/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/Security/CWE-327/BadRandomness.qlx.
[10/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/Security/CWE-300/InsecureDependencyResolution.qlx.
[11/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/Security/CWE-829/InsecureDownload.qlx.
[12/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/Security/CWE-916/InsufficientPasswordHash.qlx.
[13/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/Security/CWE-022/ZipSlip.qlx.
[14/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/Security/CWE-022/TaintedPath.qlx.
[15/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/Security/CWE-611/Xxe.qlx.
[16/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/Security/CWE-200/PrivateFileExposure.qlx.
[17/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/Security/CWE-201/PostMessageStar.qlx.
[18/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/Security/CWE-178/CaseSensitiveMiddlewarePath.qlx.
[19/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/Security/CWE-078/ShellCommandInjectionFromEnvironment.qlx.
[20/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/Security/CWE-078/UnsafeShellCommandConstruction.qlx.
[21/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/Security/CWE-078/SecondOrderCommandInjection.qlx.
[22/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/Security/CWE-078/CommandInjection.qlx.
[23/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/Security/CWE-078/UselessUseofCat.qlx.
[24/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/Security/CWE-089/SqlInjection.qlx.
[25/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/Security/CWE-776/XmlBomb.qlx.
[26/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/Security/CWE-915/PrototypePollutingFunction.qlx.
[27/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/Security/CWE-915/PrototypePollutingMergeCall.qlx.
[28/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/Security/CWE-915/PrototypePollutingAssignment.qlx.
[29/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/Security/CWE-295/DisablingCertificateValidation.qlx.
[30/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/Security/CWE-798/HardcodedCredentials.qlx.
[31/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/Security/CWE-754/UnvalidatedDynamicMethodCall.qlx.
[32/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/Security/CWE-209/StackTraceExposure.qlx.
[33/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/Security/CWE-830/FunctionalityFromUntrustedSource.qlx.
```

*Thực hiện quét nofixed.qldatabase*

```
PrototypePollutingAssignment.qldatabase : [84/90 eval 61ms] Results written to codeql/javascript-queries/Security/CWE-915/PrototypePollutingAssignment.bqrs.
PrototypePollutingFunction.qldatabase : [85/90 eval 15ms] Results written to codeql/javascript-queries/Security/CWE-915/PrototypePollutingFunction.bqrs.
PrototypePollutingMergeCall.qldatabase : [86/90 eval 375ms] Results written to codeql/javascript-queries/Security/CWE-915/PrototypePollutingMergeCall.bqrs.
InsufficientPasswordHash.qldatabase : [87/90 eval 6ms] Results written to codeql/javascript-queries/Security/CWE-916/InsufficientPasswordHash.bqrs.
RequestForgery.qldatabase : [88/90 eval 129ms] Results written to codeql/javascript-queries/Security/CWE-918/RequestForgery.bqrs.
LinesOfCode.qldatabase : [89/90 eval 3ms] Results written to codeql/javascript-queries/Summary/LinesOfCode.bqrs.
LinesOfUserCode.qldatabase : [90/90 eval 790ms] Results written to codeql/javascript-queries/Summary/LinesOfUserCode.bqrs.
Shutting down query evaluator.
Interpreting results.
CodeQL scanned 4 out of 4 JavaScript/TypeScript files in this invocation. Typically CodeQL is configured to analyze a single CodeQL language per invocation, so check other invocations to determine overall coverage information.
```

*File .ql được sử dụng – Kết quả được tổng hợp*

Xem kết quả qua nofixed\_result.sarif bằng sarif viewer – extension của VSCode:



*Sarif Viewer – Extension của VSCode*

### 3. Thực hiện khai thác:

Chạy file ./build\_docker.sh để web có thể chạy trên local port 1337, set mật khẩu tùy ý:

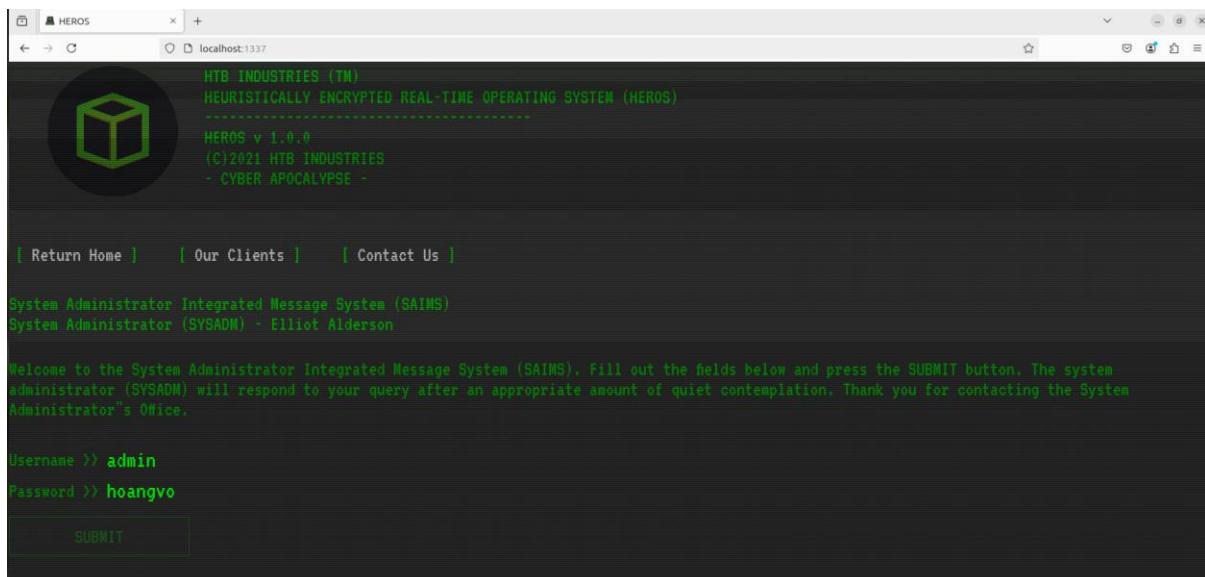
```
hoangvo@ubuntu:~/Documents/BaoMatWeb/0oAn/0oqil.rofi$ ./build_docker.sh,cg
Error response from daemon: No such container: web_wild_goosehunt
[+] Building 1.0s (15/15) FINISHED
=> [internal] load build definition from Dockerfile
=> [1/10] FROM docker.io/library/node:alpine@sha256:c61beb12a3c96373673cd52d7ecce2314e82bca5d541eef6bc6aee870c8c6f7
=> [internal] load build context
=> [internal] load metadata for docker.io/library/node:alpine
=> [internal] load .dockerignore
=> [internal] transfer context: 28
=> [1/10] FROM docker.io/library/node:alpine@sha256:c61beb12a3c96373673cd52d7ecce2314e82bca5d541eef6bc6aee870c8c6f7
=> [internal] load build context
=> [internal] load metadata for docker.io/library/node:alpine
=> [internal] transfer context: 1.25kB
=> [2/10] RUN echo 'http://dl-cdn.alpinelinux.org/alpine/v3.6/main' >> /etc/apk/repositories
=> [3/10] RUN echo 'http://dl-cdn.alpinelinux.org/alpine/v3.6/community' >> /etc/apk/repositories
=> [4/10] RUN apk add --update --no-cache supervisor mongodb
=> [5/10] RUN mkdir -p /app
=> [6/10] WORKDIR /app
=> [7/10] COPY challenge .
=> [8/10] RUN yarn
=> [9/10] COPY config/supervisord.conf /etc/supervisord.conf
=> [10/10] COPY entrypoint.sh /entrypoint.sh
=> exporting layers
=> writing image sha256:2829111de79de1d7ee3cad5775c3488ef1c114462f92c452d8755c1a5853985
=> naming to docker.io/library/web_wild_goose_hunt
```

Host Web

```
$ entrypoint.sh
1 #!/bin/ash
2
3 # Secure entrypoint
4 chmod 600 /entrypoint.sh
5 mkdir /tmp/mongodb
6 mongod --noauth --dbpath /tmp/mongodb/ &
7 sleep 2
8 mongo heros --eval "db.createCollection('users')"
9 mongo heros --eval 'db.users.insert( { username: "admin", password: "hoangvo" } )'
10 /usr/bin/supervisord -c /etc/supervisord.conf
```

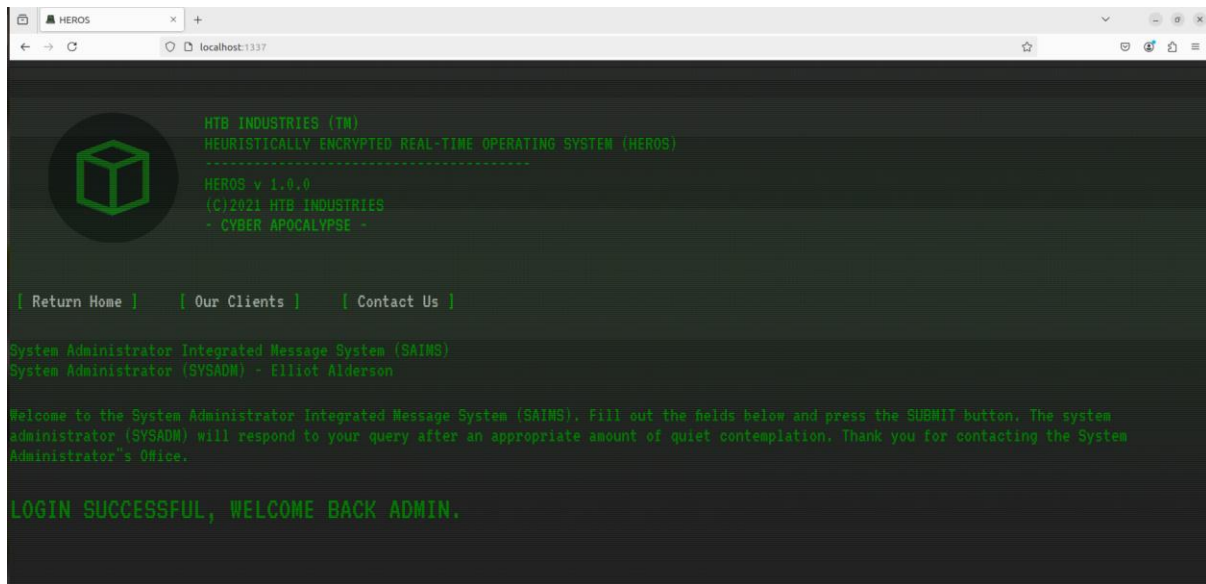
Mật khẩu cho user admin

Đăng nhập vào web,



Nhập thông tin

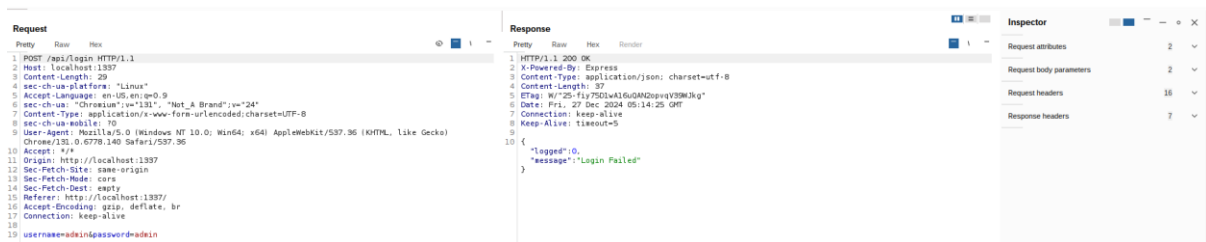




Đăng nhập thành công

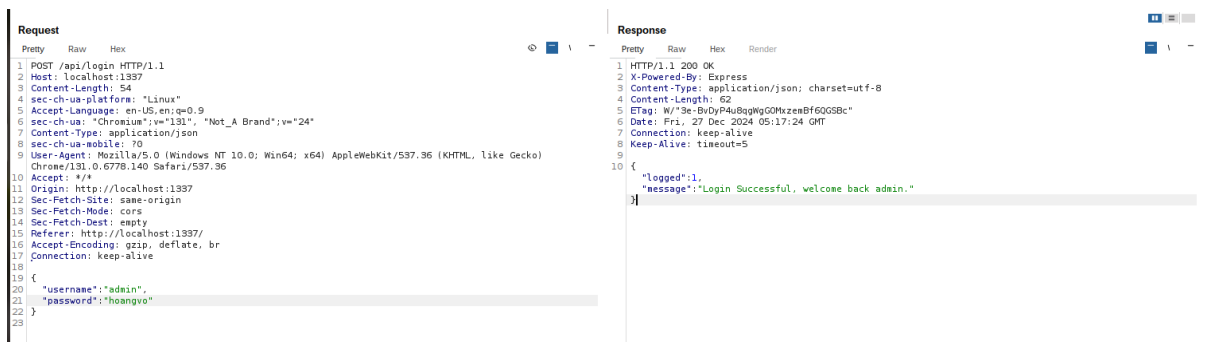
Thực hiện: khai thác lỗ hổng NoSQL Injection

#	Host	Method	URL	Params	Edited	Status code	Length	MIME type	Extension	Title	Notes	TLS	IP	Cookies	Time	Listener port	Start response
1	http://localhost:1337	GET	/			304	179						127.0.0.1		12/14/11 27 Dec...	8000	25
3	http://localhost:1337	GET	/login			304	265	script	js				127.0.0.1		12/14/11 27 Dec...	8000	2
7	http://localhost:1337	POST	/api/login		✓	200	272	json					127.0.0.1		12/14/25 27 Dec...	8000	11



Gửi tin bắt được khi đăng nhập

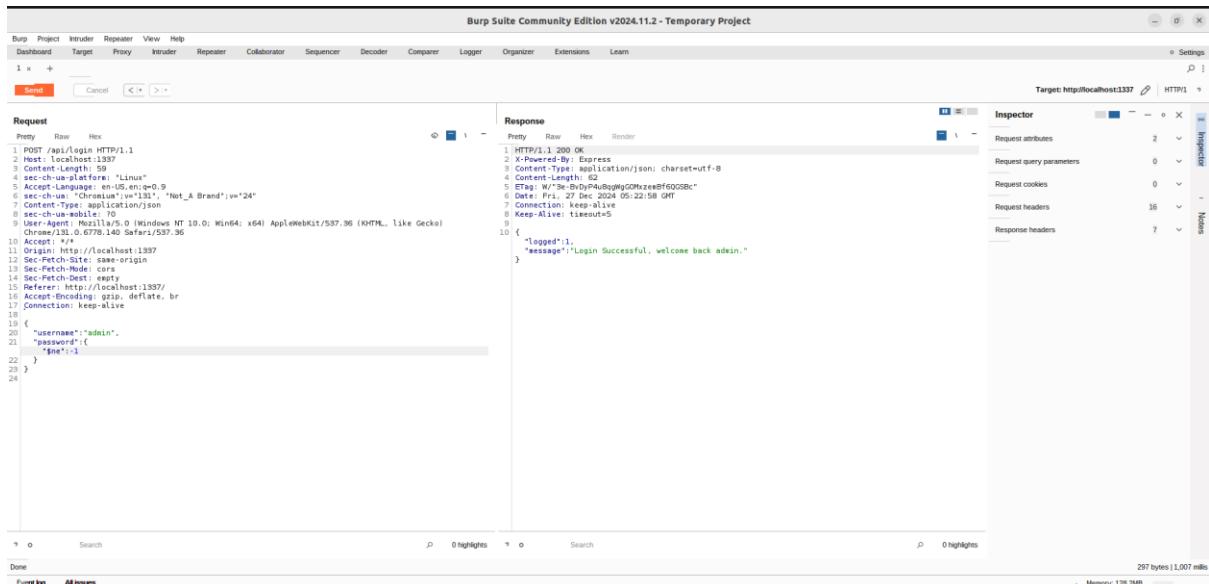
Send vào Repeater, thay đổi Content-Type thành application/json, nội dung gửi, username, password theo kiểu json -> Kết quả vẫn đăng nhập thành công



Send gói tin theo kiểu application/json

Thực hiện khai thác: thay password thành một kiểu json, thay vì “password” :  
 “<mật khẩu>” ta dùng “password” : {“\$ne” : “-1” }

Kết quả: Vẫn thành công

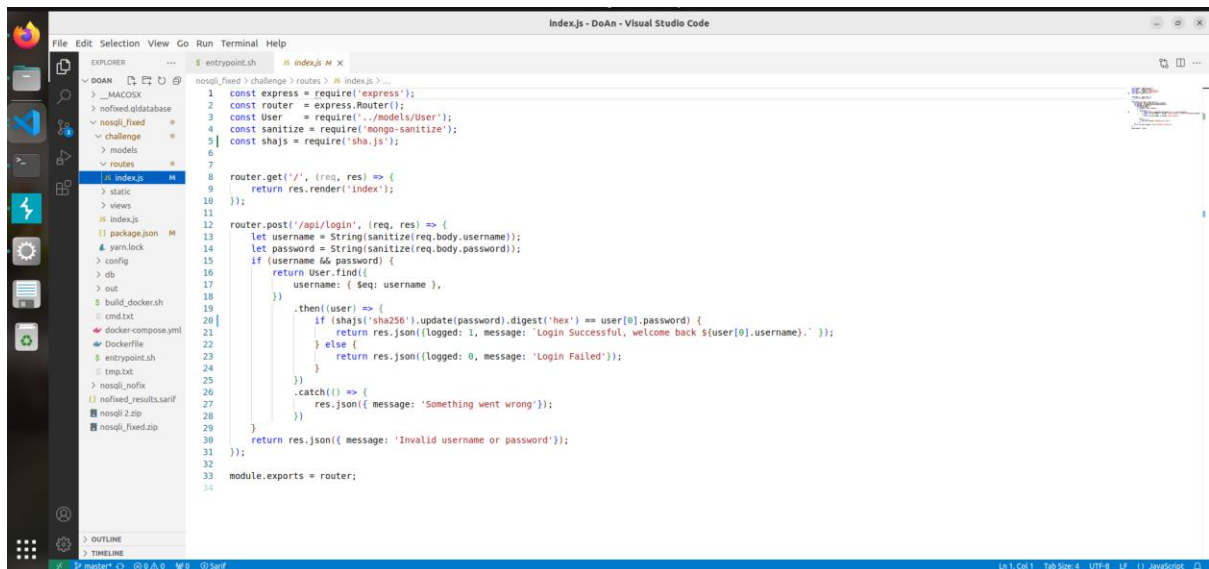


*Send gói tin sau khi sửa ở “password”*

Lí do xảy ra điều này, là do hàm truy vấn find, khi chỉnh sửa payload của gói tin như vậy, find sẽ xem “password” : {“\$ne” : “-1” } là một truy vấn, nó sẽ kiểm tra liệu password có khác -1 không nếu có thì nó trả về true. Do đó, chúng ta vẫn đăng nhập bình thường.

#### 4. Thực hiện sửa chữa source code và tạo lại qldatabase:

Vì lỗi được chỉ định CWE ở file ./challenge/routes/index.js, ta tập trung sửa chữa ở file này. Ý tưởng sẽ là kiểm tra username và password cách biệt:



```

1 const express = require('express');
2 const router = express.Router();
3 const User = require('../models/User');
4 const sanitize = require('mongo-sanitize');
5 const shajs = require('sha.js');
6
7
8 router.get('/', (req, res) => {
9   return res.render('index');
10 });
11
12 router.post('/api/login', (req, res) => {
13   let username = String(sanitize(req.body.username));
14   let password = String(sanitize(req.body.password));
15   if (username && password) {
16     return User.find(
17       { username: { $eq: username } },
18       (err, user) => {
19         if (user) {
20           if (shajs('sha256').update(password).digest('hex') === user[0].password) {
21             return res.json({ logged: 1, message: 'Login Successful, welcome back ' + user[0].username });
22           } else {
23             return res.json({ logged: 0, message: 'Login Failed' });
24           }
25         }
26       }
27     ).catch(() => {
28       res.json({ message: 'Something went wrong' });
29     });
30   }
31   return res.json({ message: 'Invalid username or password' });
32 });
33 module.exports = router;
34

```

*Code ./challenge/routes/index.js sau khi sửa*

Demo thử kiểm tra tính năng:

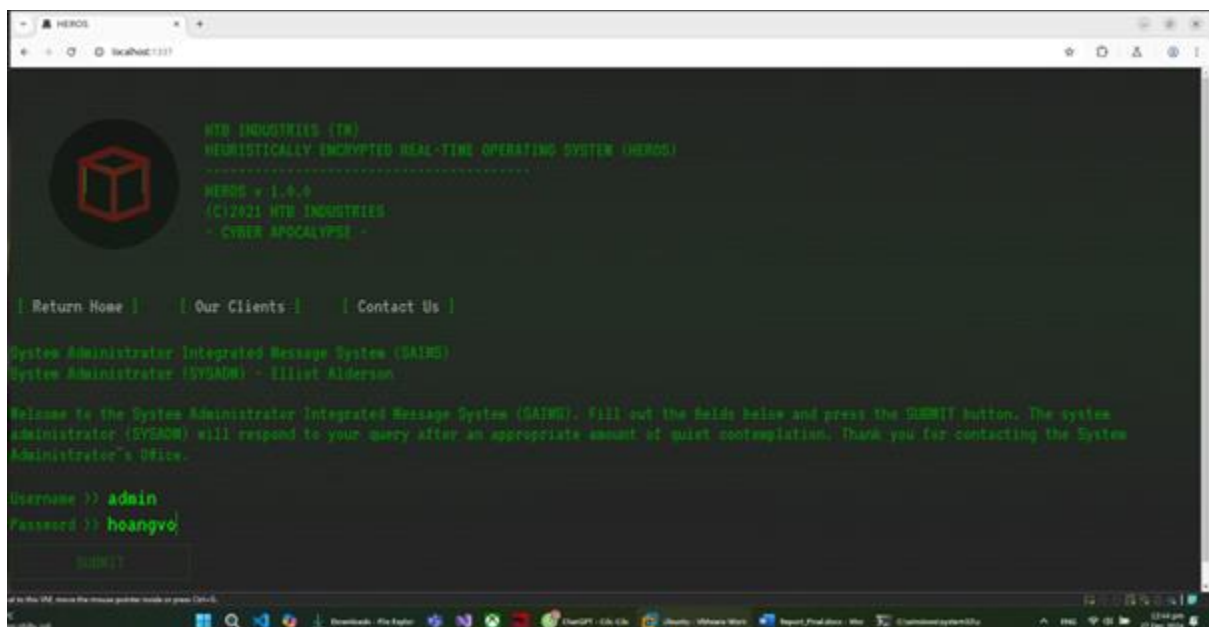


```

nosql_fixed > $ ./entrypoint.sh
1 #!/bin/ash
2
3 # Secure endpoint
4 chmod 600 ./entrypoint.sh
5 mkdir /tmp/mongodb
6 mongod --noauth --dbpath /tmp/mongodb/ &
7 sleep 3
8 mongo heros --eval 'db.createCollection('users')'
9 mongo heros --eval 'db.users.insert( { username: "admin", password: "0ddce3087ecf409ebf17dcff9b3a30b93dc4b0c2dd6465c53ea64449362680" } )'
10 mongo heros --eval 'db.users.insert( { username: "duyhuynh", password: "04f8996da763b7a969b1028ee3087569eaf3a635486ddab211d512c85b9df8fb" } )'
11 /usr/bin/supervisord -c /etc/supervisord.conf
12

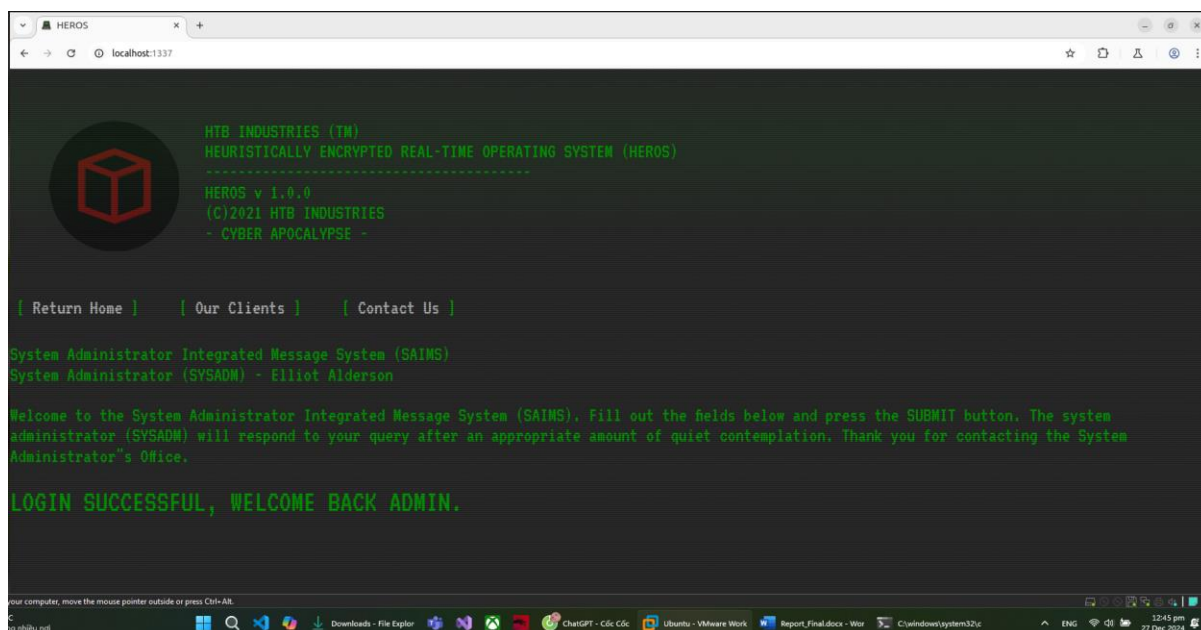
```

*File ./entrypoint.sh mới*



*Thử đăng nhập*





### *Đăng nhập thành công*

Về hoạt động sau khi sửa code, vẫn bình thường. Tiến hành tạo qldatabase cho folder source mới sửa:

```
`codeql database create fixed.qldatabase --language=javascript --source-root=nosqli_fixed`
```

```
hoangvn@ubuntu:~/Documents/BaoMatWeb/DoAn$ codeql database create fixed.qldatabase --language=javascript --source-root=nosqli_fixed
Initializing database at /home/hoangvn/Documents/BaoMatWeb/DoAn/fixed.qldatabase.
Running build command: []
Running command in /home/hoangvn/Documents/BaoMatWeb/DoAn/nosqli_fixed: [/home/hoangvn/codeql-home/codeql/javascript/tools/autobuild.sh]
```

### *Tạo qldatabase cho nosqli\_fixed*

```
Running TRAP import for CodeQL database at /home/hoangvn/Documents/BaoMatWeb/DoAn/fixed.qldatabase...
Importing TRAP files
Merging relations
Finished writing database (relations: 13.36 MiB; string pool: 4.78 MiB).
TRAP import complete (5.5s).
Finished zipping source archive (247.90 KiB).
Successfully created database at /home/hoangvn/Documents/BaoMatWeb/DoAn/fixed.qldatabase.
hoangvn@ubuntu:~/Documents/BaoMatWeb/DoAn$ ls
fixed.qldatabase  __MACOSX  nofixed.qldatabase  nofixed_results.sarif  'nosqli 2.zip'  nosqli_fixed  nosqli_fixed.zip  nosqli_nofix
```

### *Tạo qldatabase thành công*

## 5. Thực hiện quét “fixed.qldatabase”:

Dùng lệnh: `codeql database analyze --format=sarif-latest --output=fixed\_results.sarif fixed.qldatabase`

```
hoangvn@ubuntu:~/Documents/BaoMatWeb/DoAn$ codeql database analyze --format=sarif-latest --output=fixed_results.sarif fixed.qldatabase
Running queries.
[1/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/Performance/ReDoS.qlx.
[2/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/Performance/PolynomialReDoS.qlx.
[3/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/AngularJS/DisablingSce.qlx.
[4/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/AngularJS/DoubleCompilation.qlx.
[5/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/AngularJS/InsecureURWhitelist.qlx.
[6/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/Security/CWE-480/DeepObjectResourceExhaustion.qlx.
[7/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/Security/CWE-346/CorsMisconfigurationForCredentials.qlx.
[8/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/Security/CWE-327/BrokenCryptoAlgorithm.qlx.
[9/90] Loaded /home/hoangvn/codeql-home/codeql/qlpacks/codeql/javascript-queries/1.2.5/Security/CWE-327/BadRandomness.qlx.
```

### *Analyze fixed.qldatabase*

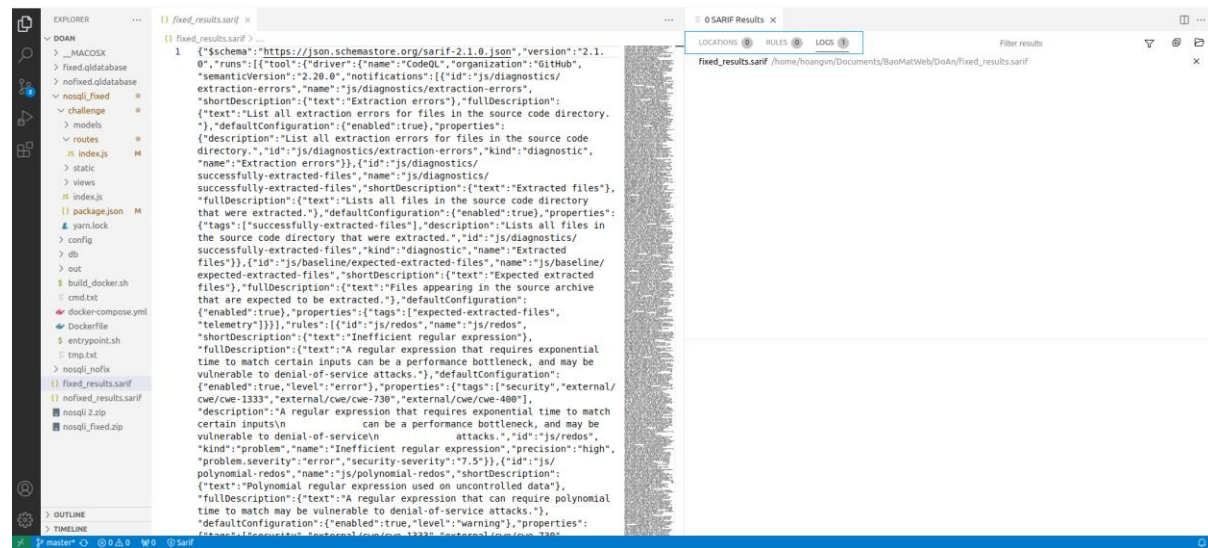
```

PrototypePollutingMergeCall.q1      : [86/90 eval 400ms] Results written to codeql/javascript-queries/Security/CWE-915/PrototypePollutingMergeCall.bqrs.
InsufficientPasswordHash.q1        : [87/90 eval 22ms] Results written to codeql/javascript-queries/Security/CWE-916/InsufficientPasswordHash.bqrs.
RequestForgery.q1                  : [88/90 eval 190ms] Results written to codeql/javascript-queries/Security/CWE-918/RequestForgery.bqrs.
LinesOfCode.q1                    : [89/90 eval 5ms] Results written to codeql/javascript-queries/Summary/LinesOfCode.bqrs.
LinesOfUserCode.q1                 : [90/90 eval 1.5s] Results written to codeql/javascript-queries/Summary/LinesOfUserCode.bqrs.
Shutting down query evaluator.
Interpreting results.
CodeQL scanned 4 out of 4 JavaScript/TypeScript files in this invocation. Typically CodeQL is configured to analyze a single CodeQL language per invocation, so check
other invocations to determine overall coverage information.
hoangvn@ubuntu:~/Documents/BaoMatWeb/DoAn$

```

*Quét dựa trên các lỗ hổng và CWE theo mặc định của codeql*

Kiểm tra file fixed\_results.sarif -> Kết quả: không còn xuất hiện lỗ hổng.



*fixed\_results.sarif*

## **Chương V. KẾT LUẬN.**

### **1. Kết luận:**

CodeQL thực sự là một công cụ hữu ích, nó giúp người lập trình, người phát triển dịch vụ, ứng dụng, dự án phát hiện các điểm yếu, lỗ hổng trong code, tạo điều kiện cho họ chỉnh sửa và cải thiện source code kịp thời. Điều này không chỉ giảm chi phí, tài nguyên trong việc kiểm thử, pentest mà còn giảm thời gian, công sức của tất cả mọi người.

Với sự đa dạng trong nền tảng sử dụng, cũng như số lượng ngôn ngữ lập trình mà codeql hỗ trợ, codeql trong vài năm gần đây đã trở nên phổ biến, và được Github dùng như một công cụ kiểm tra bảo mật source code quan trọng. Đặc biệt hơn, vì codeql không sử dụng các mô hình học máy, học sâu để phát hiện lỗ hổng mà dùng các truy vấn trên database tạo từ source code nên áp lực lên tài nguyên, nguồn lực cho các máy chủ, máy cá nhân chạy codeql được giảm đi đáng kể, từ đó cải thiện hiệu suất cả hệ thống.

### **2. Hướng phát triển đề án tương lai của nhóm:**

- Tìm thêm nhiều cách dùng dựa trên các truy vấn ql, qls cụ thể.
- Phát triển thêm qlpacks, qlsuites cho một số ngôn ngữ chưa được hỗ trợ.
- Tìm hiểu sâu hơn về cách tạo qldatabase, các analyze qldatabase của codeql.
- Kiểm tra xem codeql có thể phát hiện được lỗi logic hay các lỗi do người lập trình, người phát triển hay không (hardcode, ...)

## NGUỒN THAM KHẢO

1. <https://codeql.github.com/docs/writing-codeql-queries/codeql-queries/#codeql-queries>.
2. <https://codeql.github.com/docs/>
3. <https://github.com/github/codeql>
4. <https://github.com/github/codeql-action>
5. <https://docs.github.com/en/code-security/codeql-cli>
6. <https://aws.amazon.com/vi/nosql/>
7. <https://portswigger.net/web-security/nosql-injection>
8. <https://learn.microsoft.com/en-us/training/modules/code-scanning-with-github-codeql/>