Code QL

1. What is code QL ?

CodeQL là công cụ phân tích dùng để tự động kiểm tra bảo mật và sử dụng trong nghiên cứu và phân tích các biến thể trong mã nguồn.

Phân tích biến thể: Là quá trình sử dụng lỗ hổng bảo mật đã biết làm hạt giống để tìm ra các vấn đề tương tự trong mã nguồn. Đó là một kỹ thuật mà các kỹ sư bảo mật sử dụng để xác định các lỗ hổng tiềm ẩn và đảm bảo các mối đe dọa này được khắc phục đúng cách trên nhiều cơ sở mã. . Ta có thể sử dụng các truy vấn CodeQL tiêu chuẩn để xác định các lỗ hổng hạt giống hoặc tìm các lỗ hổng mới bằng cách viết các truy vấn CodeQL tùy chỉnh . Sau đó, phát triển hoặc lặp lại truy vấn để tự động tìm các biến thể hợp lý của cùng một lỗi có thể bị bỏ sót bằng các kỹ thuật thủ công truyền thống.

Hiện tại CodeQL đang hỗ trợ cho các loại ngôn ngữ: **C/C++, C#, Go, Java, Python, Javascript, COBOL**

2. CodeQL Query

Có 2 loại query:

- **Alert queries**: truy vấn làm nổi bật các vấn đề ở các vị trí cụ thể trong mã.
- **Path queries**: các truy vấn mô tả luồng thông tin giữa source và sink trong mã.

Basic query structure :

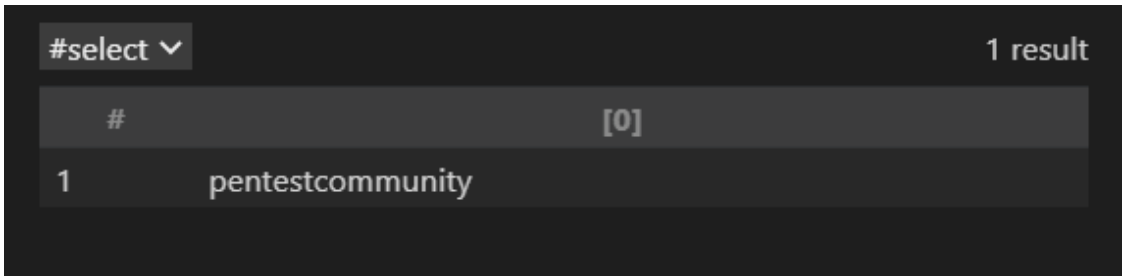
```
/**
 *
 * Query metadata
 *
 */

import /* ... CodeQL libraries or modules ... */

/* ... Optional, define CodeQL classes and predicates ... */

from /* ... variable declarations ... */
where /* ... logical formula ... */
select /* ... expressions ... */
```

Eg:

codeql-custom-queries-python > example.ql > {} example

```
1  //Metadata
2  /**
3   * @name Empty scope
4   * @kind path-problem
5   * @problem.severity warning
6   * @id python/example/empty-scope
7   */
8
9   // import library
10 import python
11
12 from string x , string y  //Khai báo dữ liệu <type> <varname> ,
13 where x = "pentest" and y = "community" /*Chứa aggregations: max(), min()...
14  predicates: được sử dụng để mô tả các quan hệ logic tạo nên một chương trình QL
15  logical formulas: Xác định quan hệ logic giữa các biến tự do được sử dụng trong biểu thức: and
16  or , not, exist(),forall.....
17 */
18 select x + y // biểu thức
```

Kết quả:



3. Một số khái niệm cơ bản

Predicate

Predicate được sử dụng để mô tả các mối quan hệ logic tạo nên một chương trình QL. Ví dụ:

```
predicate isCountry(string country) {
  country = "Germany"
  or
  country = "Belgium"
  or
  country = "France"
}

predicate hasCapital(string country, string capital) {
  country = "Belgium" and capital = "Brussels"
  or
  country = "Germany" and capital = "Berlin"
  or
  country = "France" and capital = "Paris"
}
```

The predicate `isCountry` is the set of one-tuples `{("Belgium"),("Germany"),("France")}`, while `hasCapital` is the set of two-tuples `{("Belgium","Brussels"),("Germany","Berlin"),("France","Paris")}`. The arity of these predicates is one and two, respectively.

Predicates không có dữ liệu trả về : predicate <name>(){…….}; VD: predicate isOK(){…….}

Predicates có dữ liệu trả về: <type> <name> (){…….} VD: int isOK() {…….}

Predicate sử dụng gần giống như function, có Đệ quy ,…

chi tiết: https://codeql.github.com/docs/ql-language-reference/predicates/

Source

Trong quá trình phân tích luồng dữ liệu, source được coi là nới bắt đầu của luồng dữ liệu.

Sink

Sink được coi là điểm kết thúc của luồng dữ liệu.

Flow

Luồng dữ liệu mô hình hóa cách dữ liệu chảy qua chương trình lúc chạy. Trong khi đó Abstract Syntax Tree phản ánh cấc trúc của chương trình.

4. Set up

Để có thể thực hiện truy vấn codeQL, ta cần:

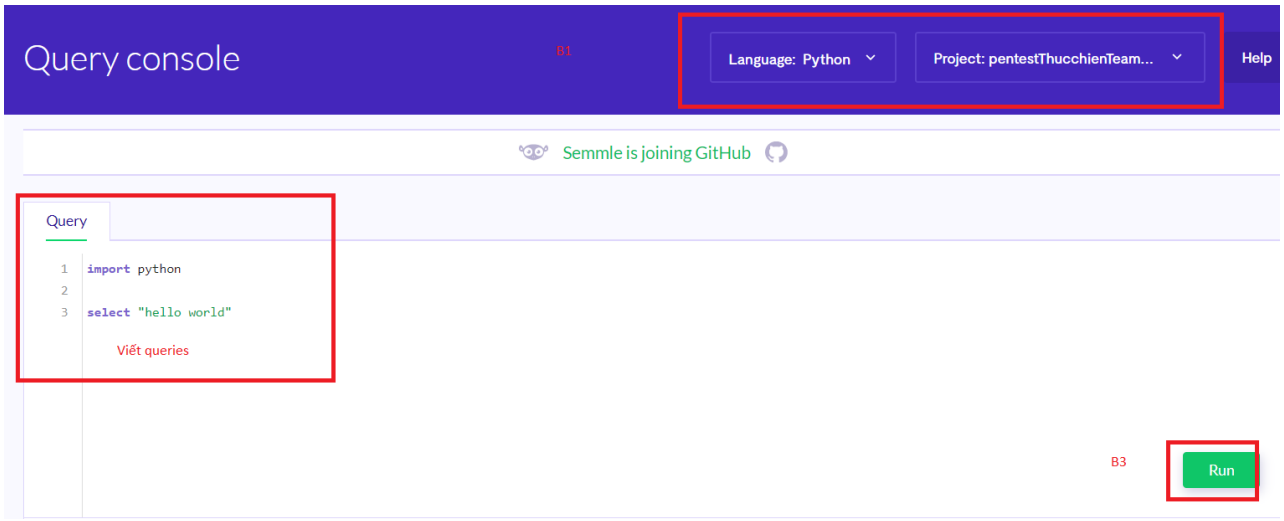
- ***Queries***
- ***Database*** để chạy câu truy vấn trong đó

CodeQL có thể thực hiện thông qua nền tảng online trên LGTM.com.

B1: Truy cập https://lgtm.com/query

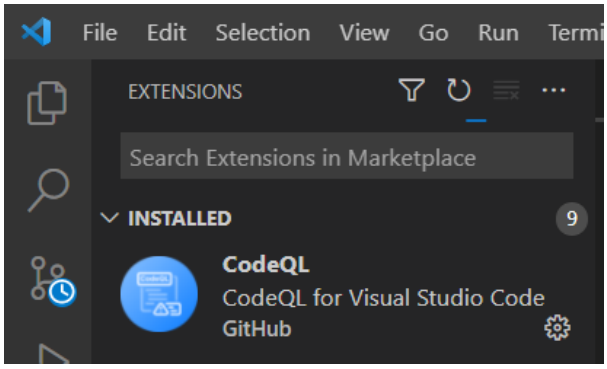
B2: Chọn ngôn ngữ và database

B3: Run queries



Chạy trên local: VS code extension

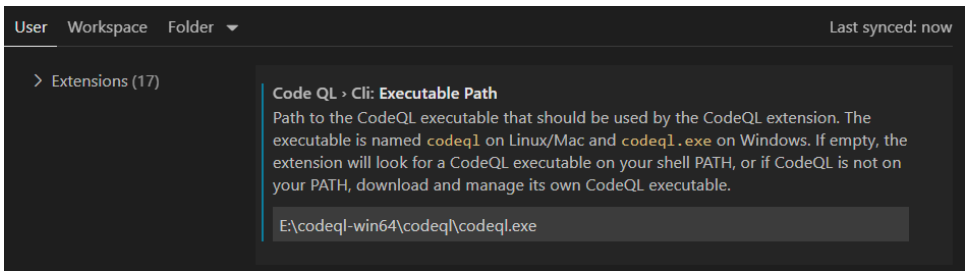
B1: Tải Extension code QL trên VS code



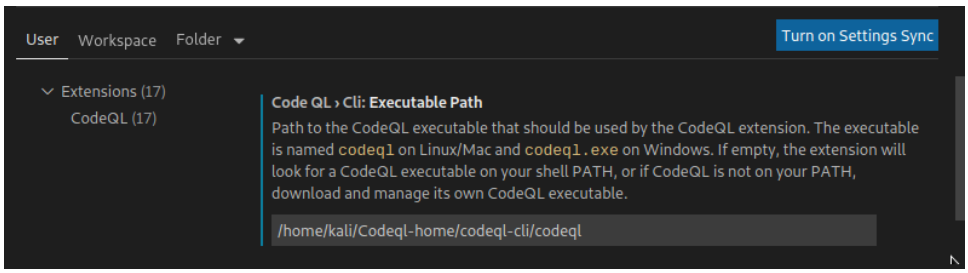
B2: Tải Codeql-cli và QL library

Tạo thư mục chứ codeql CLI và thư viện QL và giải nén

Để run queries trên VS code ta vào cài đặt extension và thêm path dẫn tới file codeql. Window sẽ là **codeql.exe** còn linux là **codeql**



Linux:



B3: Tạo Database

Để có thể Run ta phải chọn database là codebase là nơi để truy vấn.

Khi tạo database đẻ truy vấn, code ql sẽ phân tích source code và tạo 1 bản snapshot trên source. Để tạo database trên terminal ta sử dụng câu lệnh:

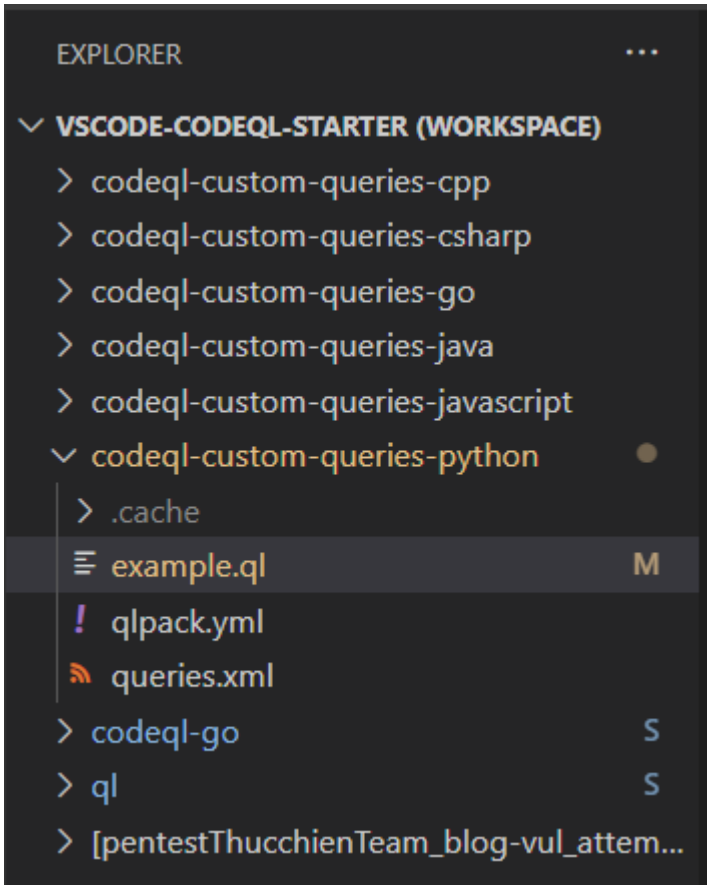
```
codeql database create <database> --language=<language-identifier>
```

- <database>: đường dẫn tới thư mục database
- --language: chỉ định ngôn ngữ để tạo database

B4: Viết truy vấn

Clone Work Space Starter : https://github.com/github/vscode-codeql-starter/

- **Codeql-custom-queries-xxx : xxx** là ngôn ngữ ta muốn dùng để truy vấn.
- Các câu truy vấn được viết trong file **.ql** và đặt trong thư mục của ngôn ngữ mình muốn viết.



Chi tiết cách set up codeQL trên local :
https://codeql.github.com/docs/codeql-for-visual-studio-code/setting-up-codeql-in-visual-studio-code/#starter-workspace

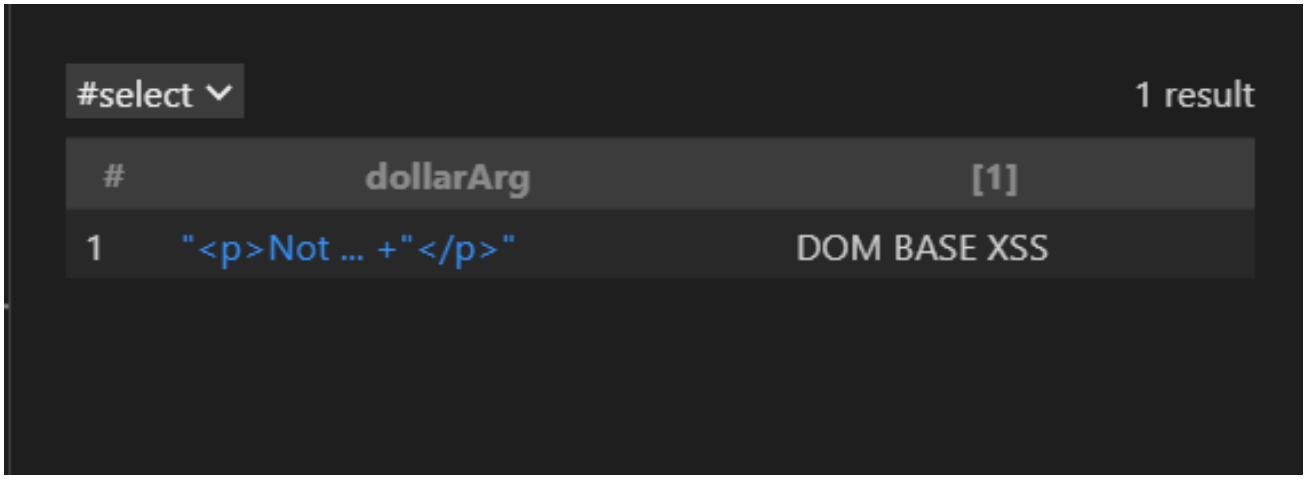
Viết truy vấn

Ta sẽ tạo một file như yêu cầu ở trên để bắt đầu viết truy vấn, ví dụ ta viết một truy vấn tìm lỗi **DOM base XSS:**

- **Ngôn ngữ**: Javascript
- **Database** với ngôn ngữ JS

```
codeql-custom-queries-javascript > example.ql > {} example
/**
 * @name Empty block
 * @kind problem
 * @problem.severity warning
 * @id javascript/example/empty-block
 */

import javascript
from Expr dollarArg,CallExpr dollarCall // tạo 2 đối tượng của lớp Expr và CallExpr
where dollarCall.getCalleeName() = "write" and  // Đối tượng có tên method là "Write"
    dollarCall.getReceiver().toString() = "document" and // BIểu thức nơi nhận hàm write
    dollarArg = dollarCall.getArgument(0) // dollarArg lấy giá trị bằng đối số [0] của hàm document.write
select dollarArg,"DOM BASE XSS" // Lấy kết quả
```

Và ta được kết quả:



Click vào liên kết và ta thấy được đoạn mã dẫn đến lỗi XSS mà ta truy vấn

```
<script>
function trackSearch(quer){
    document.write("<p>Not Found: "+query+"</p>" );
}
    var query = (new URLSearchParams(window.location.search)).get('searc
        if(query){
        trackSearch(query)
    }
    </script>
```

Vậy là đã tìm được **Sink**, điểm kết thúc và trực tiếp gây ra lỗi **DOM XSS**

Giờ ta sẽ đi tìm **Source**, những nơi mà dữ liệu đi vào lấy trực tiếp từ user thông qua biến **query** :

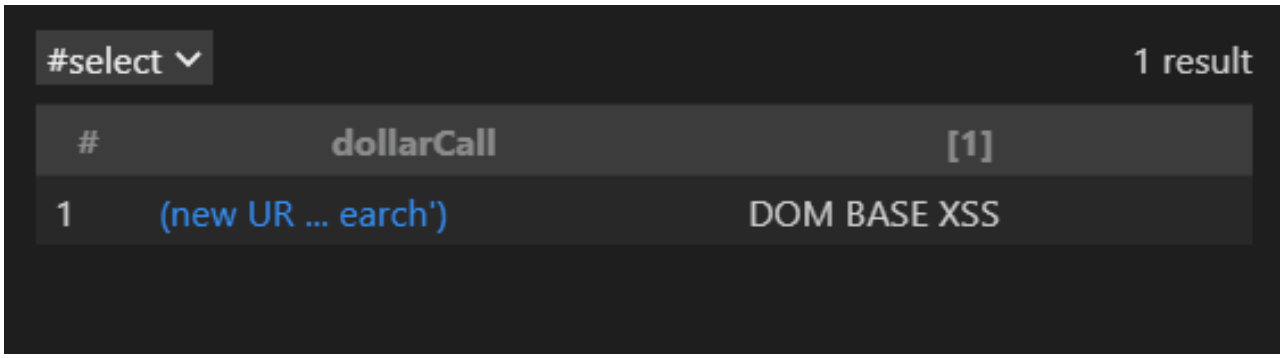
Ở đây ta tìm nơi input được lấy từ parameter **“search”** trên URL:

(Câu query này chưa chuẩn nhưng chưa đủ thời gian nghiên cứu nên lấy tạm , nhưng kết quả nhận được nó cũng giống câu query chuẩn hehe:v)

codeql-custom-queries-javascript > flowabc.ql > {} flowabc

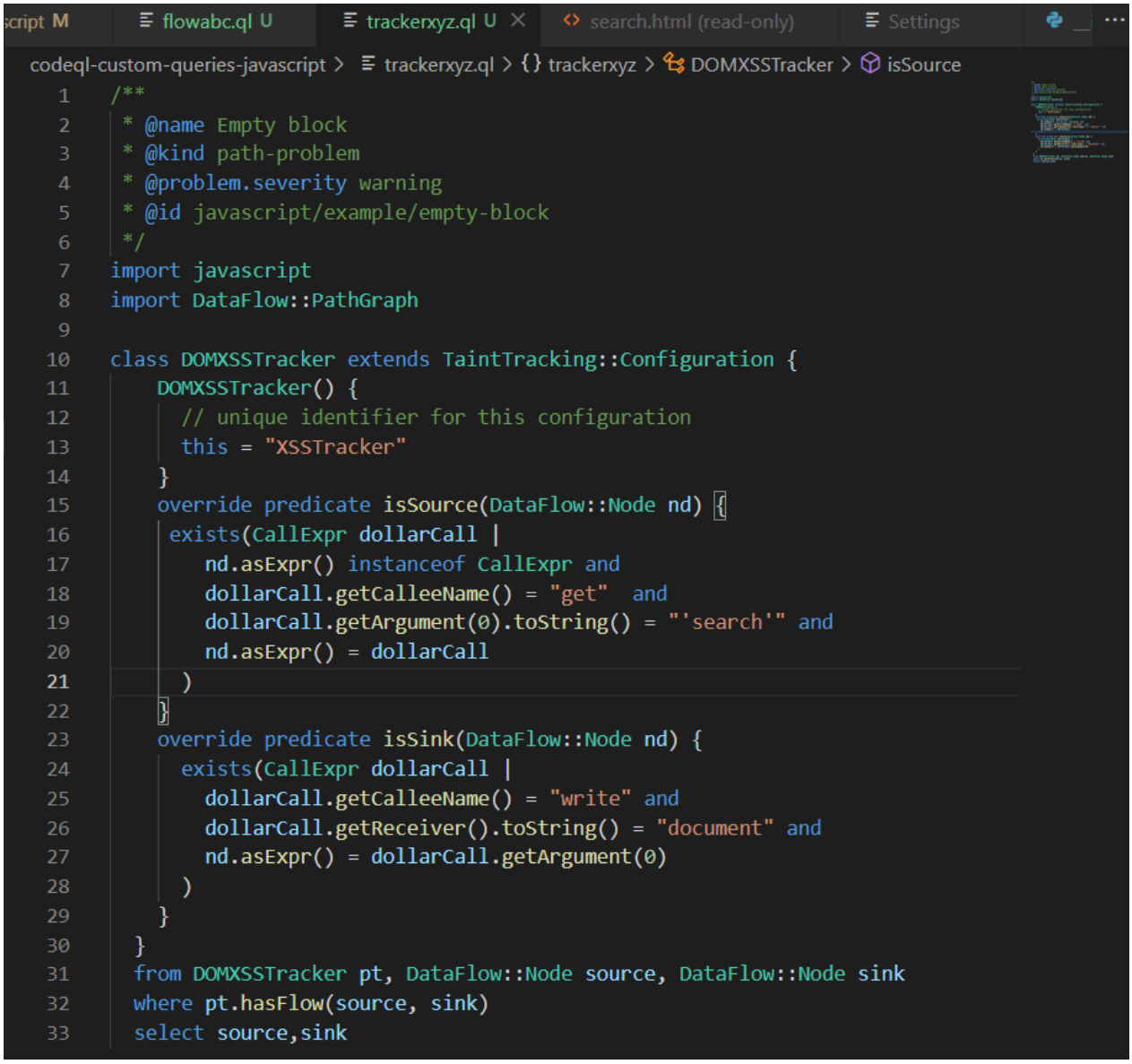
```
/**
 * @name Empty block
 * @kind problem
 * @problem.severity warning
 * @id javascript/example/empty-block
 */

import javascript
from CallExpr dollarCall // tạo 2 đối tượng của lớp Expr và CallExpr
where dollarCall.getCalleeName() = "get"  and

    dollarCall.getArgument(0).toString() = "'search'"

select dollarCall,"DOM BASE XSS" // Lấy kết quả

```

Kết quả:




```
var query = (new URLSearchParams(window.location.search)).get('search');
```

Vậy là ta đã tìm được **Source** và **Sink**, tiếp theo ta sẽ sử dụng **Taint Tracking** trong codeql để tìm tất cả những đoạn code trong mã nguồn của chúng ta mà có luồng dữ liệu đi từ **Source -> Sink**:



Kết quả:

Chỉ có 1 flow từ source -> sink gây ra lỗi **DOM XSS**

Reference

CodeQL Docs: https://codeql.github.com/docs/codeql-overview/

Blog: https://viblo.asia/p/tim-hieu-codeql-Qpmlep7MZrd

https://sec.vnpt.vn/2020/05/codeql-for-beginner-part-1/