

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.q1 > {} 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:

#select ▾		1 result
#	[0]	
1	pentestcommunity	

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 Định 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 nơi 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 cấu trúc của chương trình.

4. Set up

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

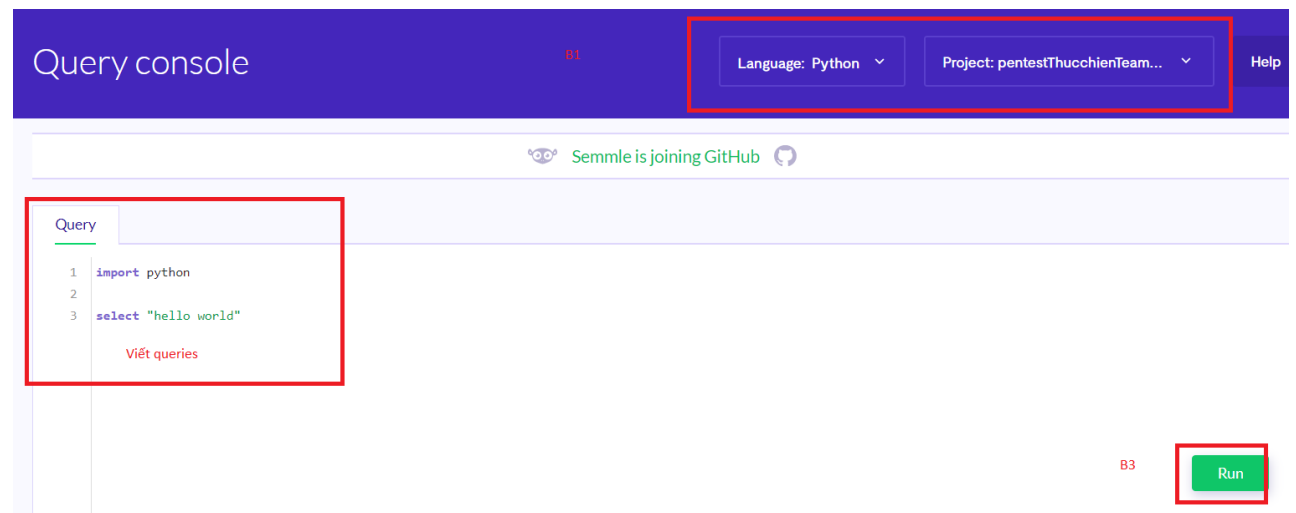
- *Queries*
- *Database* để chạy câu truy vấn trong đó

CodeQL có thể thực hiện thông qua nền tảng online trên [LGTM.com](https://lgtm.com).

B1: Truy cập <https://lgtm.com/query>

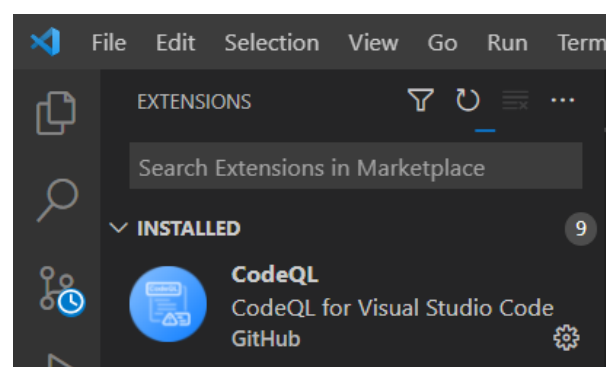
B2: Chọn ngôn ngữ và database

B3: Run queries



Chạy trên local: VS code extension

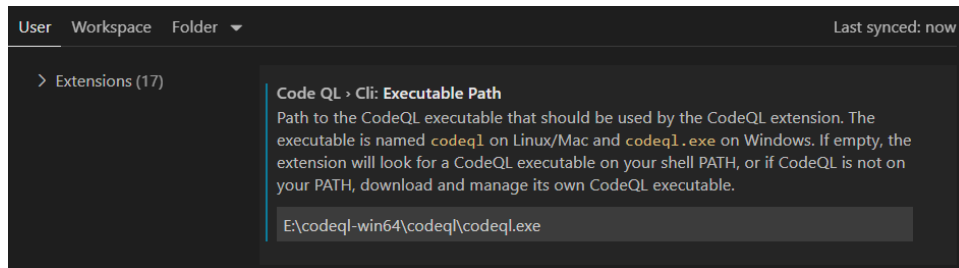
B1: Tải Extension code QL trên VS code



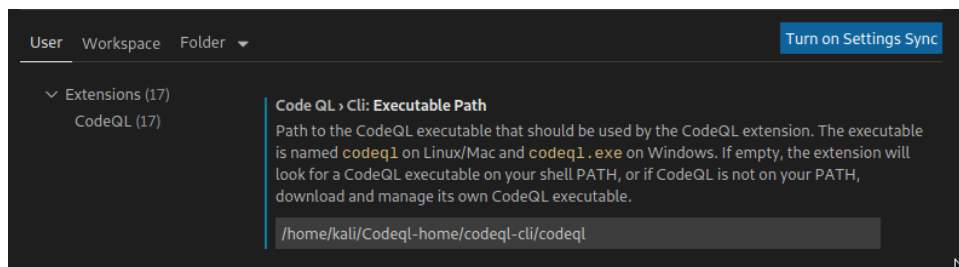
B2: Tải [Codeql-cli](#) và [QL library](#)

Tạo thư mục chứa 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.

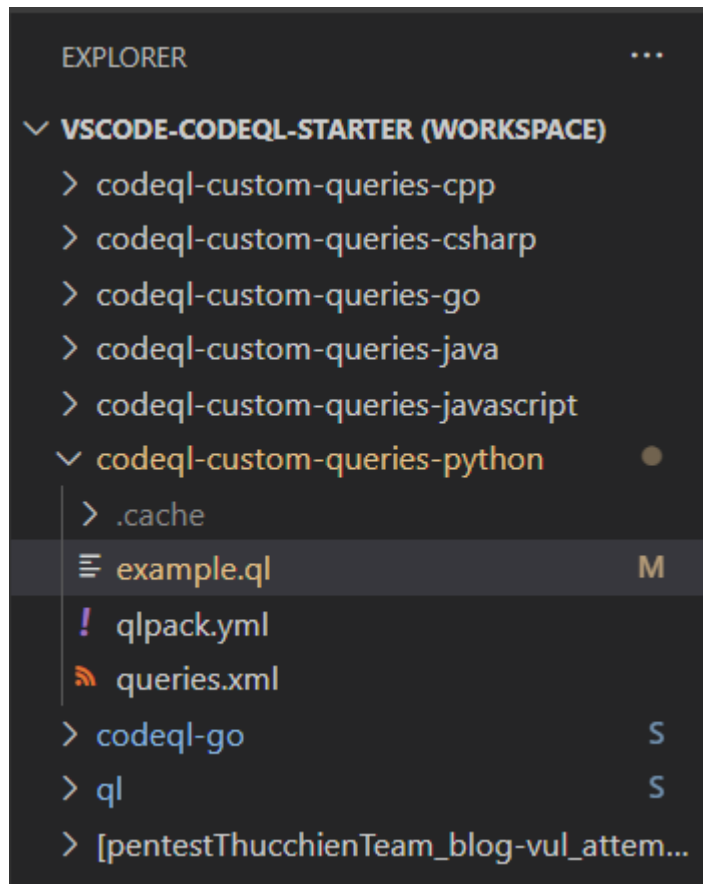
Khi tạo database để truy vấn, code ql sẽ phân tích source code và tạo 1 bản snapshot trên source. Để tạo database trên terminal ta sử dụng câu lệnh:

```
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

- **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.q1 > {} example
1  /**
2   * @name Empty block
3   * @kind problem
4   * @problem.severity warning
5   * @id javascript/example/empty-block
6   */
7
8  import javascript
9  from Expr dollarArg, CallExpr dollarCall // tạo 2 đối tượng của lớp Expr và CallExpr
10 where dollarCall.getCalleeName() = "write" and // Đối tượng có tên method là "write"
11     dollarCall.getReceiver().toString() = "document" and // Biểu thức nơi nhận hàm write
12     dollarArg = dollarCall.getArgument(0) // dollarArg lấy giá trị bằng đối số [0] của hàm document.write
13 select dollarArg, "DOM BASE XSS" // Lấy kết quả
14
```

Và ta được kết quả:

#select ▾			1 result
#	dollarArg		[1]
1	"<p>Not ... + "</p>"		DOM BASE XSS

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

```
31  <script>
32  function trackSearch(quer){
33      document.write("<p>Not Found: "+query+"</p>" );
34  }
35      var query = (new URLSearchParams(window.location.search)).get('search')
36      if(query){
37          trackSearch(query)
38      }
39  </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.q1 > {} flowabc
1  /**
2   * @name Empty block
3   * @kind problem
4   * @problem.severity warning
5   * @id javascript/example/empty-block
6   */
7
8  import javascript
9  from CallExpr dollarCall // tạo 2 đối tượng của lớp Expr và CallExpr
10 where dollarCall.getCallleeName() = "get" and
11
12     dollarCall.getArgument(0).toString() = "'search'"
13
14 select dollarCall,"DOM BASE XSS" // Lấy kết quả
15
```

Kết quả:

#select ▾		1 result
#	dollarCall	[1]
1	(new UR ... earch')	DOM BASE XSS

```
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**:


```
script M  flowabc.q1 U  trackerxyz.q1 U X  search.html (read-only)  Settings  ...

codeql-custom-queries-javascript >  trackerxyz.q1 > {} trackerxyz > DOMXSSTracker > isSource

1  /**
2   * @name Empty block
3   * @kind path-problem
4   * @problem.severity warning
5   * @id javascript/example/empty-block
6   */
7  import javascript
8  import DataFlow::PathGraph
9
10 class DOMXSSTracker extends TaintTracking::Configuration {
11     DOMXSSTracker() {
12         // unique identifier for this configuration
13         this = "XSSTracker"
14     }
15     override predicate isSource(DataFlow::Node nd) {
16         exists(CallExpr dollarCall |
17             nd.asExpr() instanceof CallExpr and
18             dollarCall.getCalleeName() = "get" and
19             dollarCall.getArgument(0).toString() = "'search'" and
20             nd.asExpr() = dollarCall
21         )
22     }
23     override predicate isSink(DataFlow::Node nd) {
24         exists(CallExpr dollarCall |
25             dollarCall.getCalleeName() = "write" and
26             dollarCall.getReceiver().toString() = "document" and
27             nd.asExpr() = dollarCall.getArgument(0)
28         )
29     }
30 }
31 from DOMXSSTracker pt, DataFlow::Node source, DataFlow::Node sink
32 where pt.hasFlow(source, sink)
33 select source,sink
```

Kết quả:

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

#select ▾		1 result
#	source	sink
1	(new UR ... earch')	"<p>Not ... + "</p>"

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/>