CodeQL

Mostafa Sattari

Agenda

- Introduction
- Installation
- Writing Queries
- Example

Intro

```
void fire_thrusters(double
vectors[12]) {
  for (int i = 0; i < 12 i++) {
    ... vectors[i] ...
  }
}

double thruster[3] = ... ;
fire_thrusters(thruster);
```

- In C, array types of parameters degrade to pointer types.
- The size is ignored!
- No protection from passing a mismatched array.

Intro

- ...to find all instances of the problem.

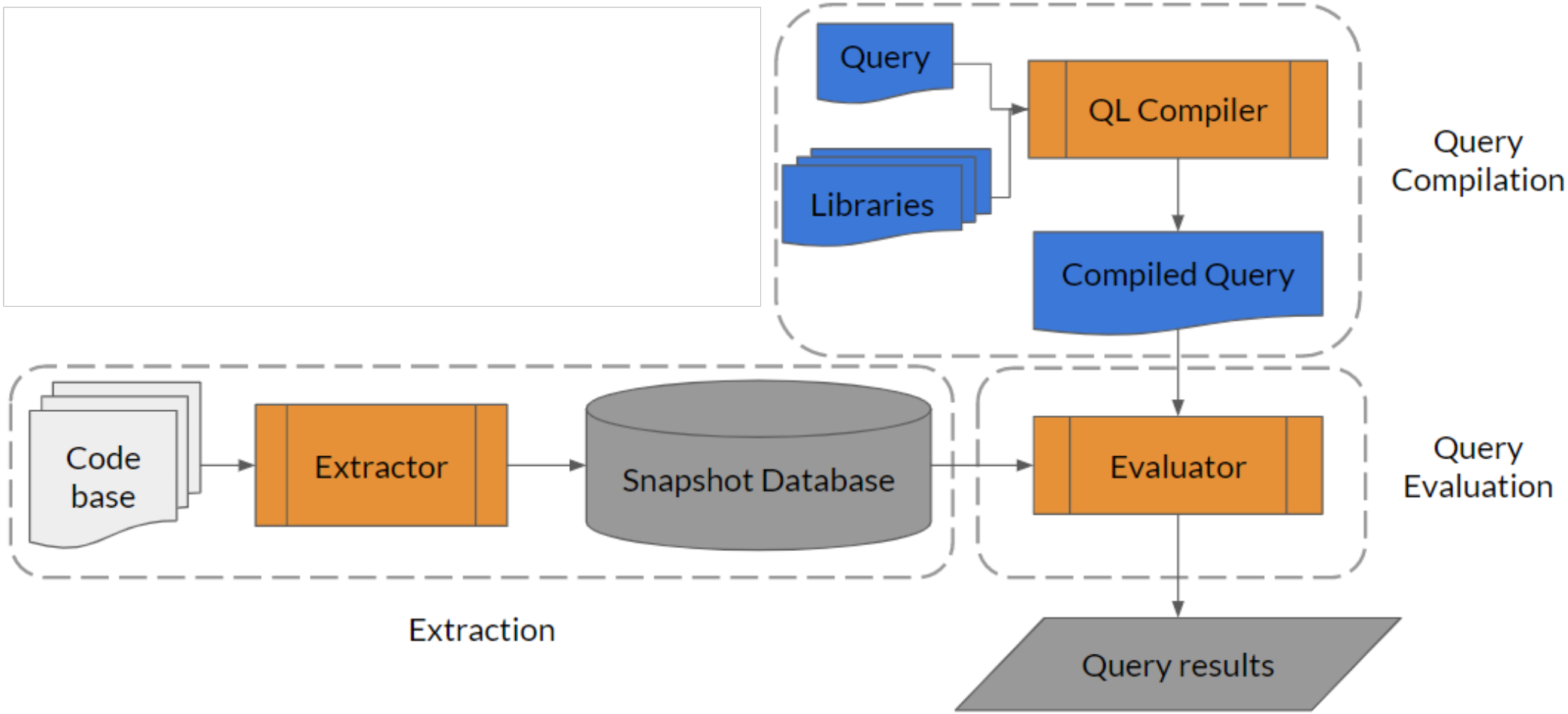
```
import cpp

from Function f, FunctionCall c, int i, int a, int b
where f = c.getTarget()
  and a = c.getArgument(i).getType().(ArrayType).getArraySize()
  and b = f.getParameter(i).getType().(ArrayType).getArraySize()
  and a < b
select c.getArgument(i), "Array of size " + a
       + " passed to $@, which expects an array of size " + b +
".",
       f, f.getName()
```

Intro

- **CodeQL Consists of:**
  - **QL**: the programming language for CodeQL code analysis platform.
  - **CLI**: run queries
  - **Libraries**: QL libraries
  - **Databases**: contains all the things needed to run the queries
- **Used for Variant Analysis**

Analysis overview



Intro - Tools

- **Standalone CodeQL CLI**
- **Interactive Query Console (lgtm.com)**
- **IDE extensions**
  - Eclipse
  - VSCode

Intro - CLI

```
mms@think: ~
mms@think: ~ 89x26
mms@think  ~  codeql
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 amount of progress
                               messages printed.
  -q, --quiet                Incrementally decrease the amount 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.
  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.
✘ mms@think  ~  |
```

Intro – Interactive Query Console

lgtm

Blog Help Query console Community Log in

Query console

Language: C/C++ Project: u-boot/u-boot Help

Semmle is joining GitHub

Query Query

```
import cpp

class Ferdowsi_CERT extends Comment {
    override string toString() { result = "Greetings from CERT of Ferdowsi University" }
}

from Ferdowsi_CERT apa
select apa.toString()
```

View results

View some example queries

Query ran on 1 project Run a minute ago Order query runs by... Share

u-boot/u-boot 643366b 1 result

col0
Greetings from CERT of Ferdowsi University

Intro – VSCode Extension

CodeQL

GitHub | 1,516 installs | ★★★★★ (0) | Free

CodeQL for Visual Studio Code

Installation

Launch VS Code Quick Open (`Ctrl+P`), paste the following command, and press enter.

```
ext install GitHub.vscode-codeql
```

Copy | More Info

Overview | Q & A | Rating & Review

CodeQL extension for Visual Studio Code

This project is an extension for Visual Studio Code that adds rich language support for CodeQL and allows you to easily find problems in codebases. In particular, the extension:

- Enables you to use CodeQL to query databases generated from source code.
- Shows the flow of data through the results of path queries, which is essential for triaging security results.
- Provides an easy way to run queries from the large, open source repository of CodeQL security queries.
- Adds IntelliSense to support you writing and editing your own CodeQL query and library files.

To see what has changed in the last few versions of the extension, see the Changelog.

Quick start overview

The information in this `README` file describes the quickest way to start using CodeQL. For information about other configurations, see the separate CodeQL help.

Quick start: Installing and configuring the extension

1. Install the extension.
2. Check access to the CodeQL CLI.
3. Clone the CodeQL starter workspace.

Quick start: Using CodeQL

Categories

Programming Languages

Tags

CodeQL | dbscheme | ql

Resources

Repository
Changelog
Download Extension

Project Details

github/vscode-codeql
Last Commit: 5 days ago
5 Pull Requests
4 Open Issues

More Info

Version	1.0.2
Released on	11/14/2019, 8:48:02 PM
Last updated	12/13/2019, 6:22:37 PM

Installation

- **What you need to run queries**
  - CodeQL CLI tool
  - Query libraries
  - A database

- **Installing VSCode and CodeQL extension**
  - Alternatively Eclipse + CodeQL extension
  - Add CodeQL cli to your env
    - ~/.config/Code/User/globalStorage/github.vscode-codeql/distribution1/codeql/codeql
    - It might vary on your machine

Installation

- **Install CodeQL CLI**

  or

- **Install VSCode and CodeQL extension**
  - Alternatively Eclipse + CodeQL extension
  - Add CodeQL cli to your env
  - ~/.config/Code/User/globalStorage/github.vscode-codeql/distribution1/codeql/codeql
    - It might vary on your machine

Installation

- **Running codeql database create**
- **Importing a database from lgtm.com**

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

--language: cpp/csharp/go/java/python/javascript
--source-root: the root folder for the primary source files (default = current directory).
--command: for compiled languages only, the build commands that invoke the compiler.
```

Writing QL Queries

- **QL**
  - **logic** programming language
  - built up of **logical** formulas
  - Object oriented
- **Basic syntax**

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

// Example:
from int x, int y
where x = 6 and y = 7
select x * y
```

Writing QL Queries

- **Python**

```
import python

from Function f
where count(f.getAnArg()) > 7
select f
```

- **Java**

```
import java

from Parameter p
where not exists(p.getAnAccess())
select p
```

- **JavaScript**

```
import javascript

from Comment c
where c.getText().regexpMatch("(?si).*\\bTODO\\b.*")
select c
```

Writing QL Queries

- ## Formulas

```
<expression> <operator> <expression> // Comparison
<expression> instanceof <type>       // Type check
<expression> in <range>              // Range check

exists(<variable declarations> | <formula>)
forex(<variable declarations> | <formula 1> | <formula 2>)
forall(<vars> | <formula 1> | <formula 2>) and
exists(<vars> | <formula 1> | <formula 2>)
```

Two formulas in the body: It holds if <formula 2> holds for all values that <formula 1> holds for.

- ## Aggregates

- *Common aggregates are **count**, **max**, **min**, **avg** (average) and **sum**.*

```
from Person t
where t.getAge() = max(int i | exists(Person p | p.getAge() = i) | i)
select t
```

Writing QL Queries

- **Predicates**

```
predicate southern(Person p) {
    p.getLocation() = "south"
}

from Person p
where southern(p)
select p
```

- The name of a predicate always starts with a **lowercase** letter.
- You can also define predicates with a result. In that case, the keyword predicate is replaced with the type of the result. This is like introducing a new argument, the special variable result. For example, ***int getAge() {result = ...} returns an int***.

Writing QL Queries

- **Classes**
  - instanceof

```
class Southerner extends Person {
    Southerner() { southern(this) }
}

from Southerner s
select s
```

```
class Child extends Person{

    /* the characteristic predicate */
    Child() { this.getAge() < 10 }

    /* a member predicate */
    override predicate isAllowedIn(string region){
        region = this.getLocation()
    }
}
```

- ***You might be tempted to think of the characteristic predicate as a constructor. However, this is not the case - it is a logical property which does not create any objects.***

Writing QL Queries

- **Annotations**
  - abstract, finaal, overrise, private, ...
- **Recursion**
  - Transitive closures +
  - Reflexive transitive closure *
- **Name Resolution**
  - Qualified references *(import examples.security.MyLibrary)*
  - Selections *(<module_expression>::<name>)*

Variant analysis

- **Control flow analysis (CFA)** allows you to inspect how the different parts of the source code are executed and in which order. Control flow analysis is useful for finding vulnerable code paths that are only executed under unlikely circumstances.
- **Data flow analysis (DFA)** is the process of tracking data from a source, where it enters an application, to a sink, where the data is used in a potentially harmful way if it's not sanitized along the way.
- **Taint tracking** typically refers to untrusted – or tainted – data that is under partial or full control of a user. Using data flow analysis, tainted data is tracked from the source through method calls and variable assignments – including containers and class members – to a sink.
- **Range analysis (or bounds analysis)** is used to investigate which possible values a variable can hold, and which values it will never hold. This is useful information in various lines of investigation.
- **Semantic code** search allows you to quickly interrogate a codebase and identify areas of interest for further investigation. This is valuable to identify methods having a particular signature, or variables that may contain credentials.

Variant analysis - Modules

- **semmle.code.cpp.dataflow.DataFlow**
  - IsSource : *defines where data may flow from*
  - IsSink : *defines where data may flow to*
  - HasFlow : *performs the analysis*
- **semmle.code.cpp.dataflow.TaintTracking**
  - IsSanitizerGuard : *optional, restricts the taint flow*

Variant analysis

- **Analyzing data flow in C/C++**

```
import cpp
import semmle.code.cpp.dataflow.TaintTracking

class MyTaintTrackingConfiguration extends TaintTracking::Configuration {
  MyTaintTrackingConfiguration() { this = "MyTaintTrackingConfiguration" }

  override predicate isSource(DataFlow::Node source) {
    ...
  }

  override predicate isSink(DataFlow::Node sink) {
    ...
  }
}
```

Variant analysis - Example

```
import semmle.code.cpp.dataflow.DataFlow

class EnvironmentToFileConfiguration extends DataFlow::Configuration {
  EnvironmentToFileConfiguration() { this =
"EnvironmentToFileConfiguration" }
  override predicate isSource(DataFlow::Node source) {
    exists (Function getenv |
      source.asExpr().(FunctionCall).getTarget() = getenv and
      getenv.hasQualifiedName("getenv")
    )
  }
  override predicate isSink(DataFlow::Node sink) {
    exists (FunctionCall fc |
      sink.asExpr() = fc.getArgument(0) and
      fc.getTarget().hasQualifiedName("fopen")
    )
  }
}
from Expr getenv, Expr fopen, EnvironmentToFileConfiguration config
where config.hasFlow(DataFlow::exprNode(getenv),
DataFlow::exprNode(fopen))
select fopen, "This 'fopen' uses data from $@.",
  getenv, "call to 'getenv'"
```

Recaps

- **Almost all materials are burrowed from *Semmle.com***
  - https://help.semmle.com/QL/learn-ql/index.html
  - https://help.semmle.com/QL/ql-training/cpp/intro-ql-cpp.html
  - https://marketplace.visualstudio.com/items?itemName=github.vscode-codeql

- **Get help**
  - https://discuss.lgtm.com/latest
  - https://stackoverflow.com/questions/tagged/semmle-ql

Thank you