If you haven’t already, download all the files from the following link

http://tiny.cc/offensivecon

OffensiveCon

Can’t grep this

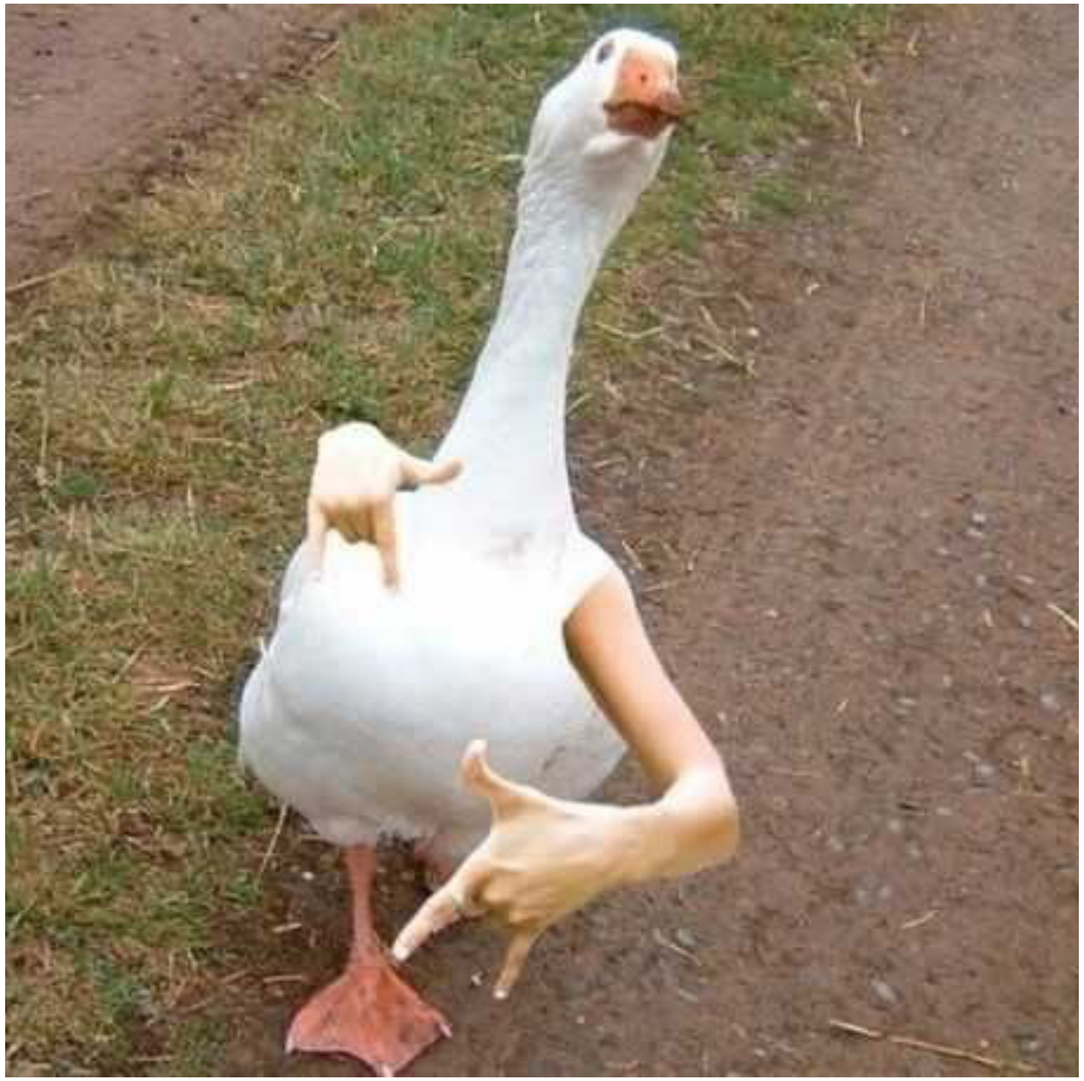
@agustingianni

About me

Goose

Security Researcher at GitHub Inc.

I like to use CodeQL.



BuT I CaN GrEp ThIs!

BuT I CaN GrEp ThIs!

- Whenever I’m talking about CodeQL this is the first thing I hear.
- Each tool has advantages and with this workshop I will try to convince you of CodeQL’s worth.
- Historically most auditors have used grep with great results.
- But can grep do the following things:
    - Match the absence of things.
    - Resolve macros.
    - Perform data flow analysis.

Workshop objectives

- Get you up and running.
- Introduce you to CodeQL.
- Avoid you some pitfalls.
- Give you some tricks I’ve developed while failing at CodeQL.

Disclaimer

- I tried to make this workshop as real life as possible.
- Run time of queries is an issue for a two hour workshop.
- Most of the examples will be reduced in scope.
- Complex things like global data flow will be performed in small projects that are guaranteed to terminate during the workshop.

Workshop Format

- Let's make this workshop as interactive as possible.
- If you are lost don’t hesitate to stop me and asking for help.
- Work in groups if you prefer so.
- The solution to the exercises are not unique, you may have expressed the query in a different way and that’s completely fine. Make sure the results match though.
- Feel free to peek into the solutions of the exercises but try to come up with your own.

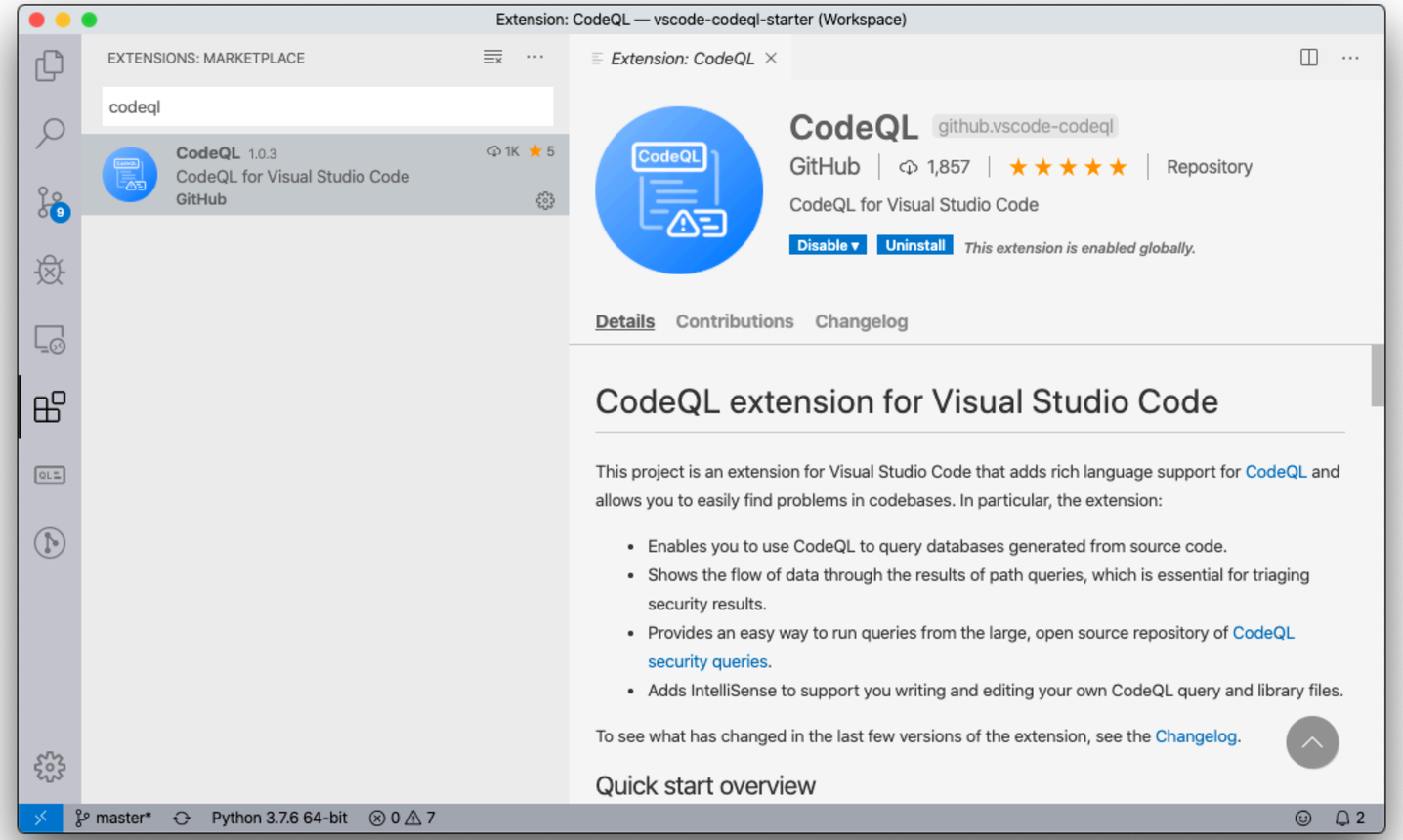
Setup

- We need the following things installed before starting
  - Visual Studio Code
  - CodeQL Plugin
  - CodeQL CLI
  - Workspace.

Visual Studio Code

- We will be using **VSCode** as our editor.
- Install it from https://code.visualstudio.com/ or via your package manager.
- Once you are done install the **CodeQL Plugin** from the marketplace as explained in the following slides.

CodeQL Plugin



CodeQL CLI + Standard Library

```
$ mkdir $HOME/semmle
$ cd $HOME/semmle
$ wget https://github.com/github/codeql-cli-binaries/releases/latest/download/codeql.zip
$ unzip codeql.zip
$ git clone https://github.com/Semmle/ql.git
$ mkdir databases
$ export PATH="$PATH:$HOME/semmle/codeql"
$ export DATABASES=“$HOME/semmle/databases"
```

CodeQL CLI + Standard Library

- The CLI will look into sibling directories for things to load like the standard libraries.
- For this reason make sure you create a directory and put everything under the same directory.
- Make sure you keep the standard library up to date since it will give you bug fixes and performance gains.

Test your installation

```
$ codeql resolve languages
 cpp (/Users/anon/semmle/codeql/cpp)
 csharp (/Users/anon/semmle/codeql/csharp)
 go (/Users/anon/semmle/codeql/go)
 java (/Users/anon/semmle/codeql/java)
 javascript (/Users/anon/semmle/codeql/javascript)
 python (/Users/anon/semmle/codeql/python)
```

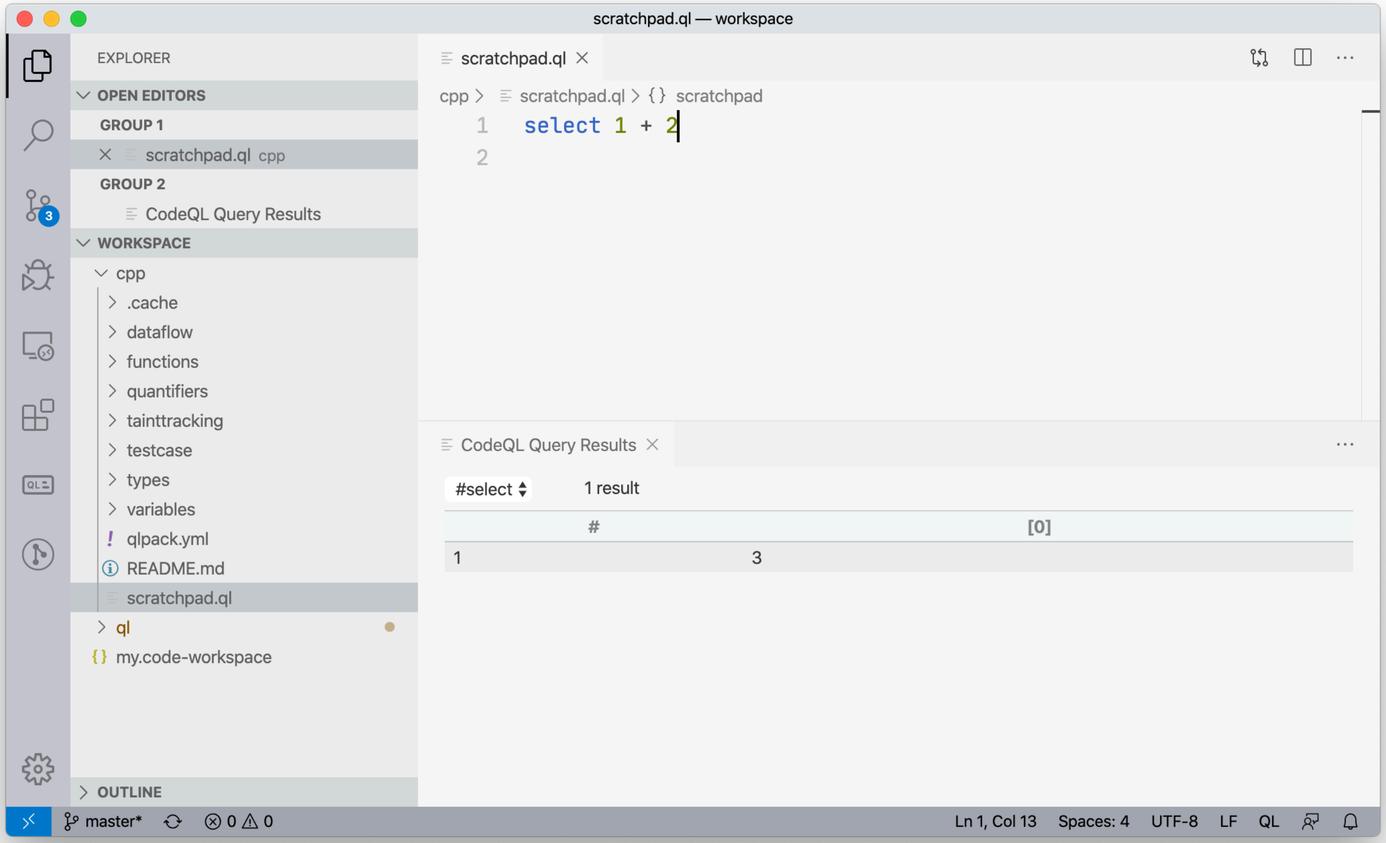
Workspace

- The workspace is where you put your CodeQL queries.
- It references the databases you will run your queries on.
- Contains your queries.
- And references the standard library you are currently using.
- Download it from the shared GDrive link as instructed in the next slides.

Workspace

- Download and uncompress all three zip files.
- Open **workspace** with vscode.
- Now add the linux-5.3-tinyconfig database to the workspace by using the CodeQL menu on the left.

Testing


scratchpad.ql — workspace
EXPLORER
OPEN EDITORS
GROUP 1
scratchpad.ql cpp
GROUP 2
CodeQL Query Results
WORKSPACE
cpp
.cache
dataflow
functions
quantifiers
tainttracking
testcase
types
variables
qlpack.yml
README.md
scratchpad.ql
ql
my.code-workspace
OUTLINE
scratchpad.ql
cpp > scratchpad.ql > { } scratchpad
1 select 1 + 2
2
CodeQL Query Results
#select
1 result
#
[0]
1
3
master*
0 0
Ln 1, Col 13
Spaces: 4
UTF-8
LF
QL

Checkpoint: are we all setup?

Introduction

What is CodeQL?

- A query language similar to **SQL** but with some subtle notes of **logic programming** languages.
- Gives you tools from logic to **reason about code**.
- Has some **object oriented** syntactic sugar to make things more familiar.
- The **CLI** allows you to turn a codebase into a database.

Your first CodeQL database

What is a CodeQL database?

- CodeQL allows you to reason about **code as if it was data**.
- For this it provides you with a **language** named QL and a **database**.
- The database is created by using the CLI tools.
- The size of a database depends on the size of your target code since QL will embed a copy of the compiled code into the database.
- This means that when a database is created it is independent of the original sources.

Methodology

- Check out the sources of your target.
- Setup the build system.
  - Run **configure** / **cmake** or whatever your project needs before hitting **build.**
- Make sure the project builds on your host.
- Clean build artifacts and rebuild using codeql.

Example: rabbitmq-c

- We will build **rabbitmq-c** from sources.
- This project requires **openssl**.
  - brew install openssl
- If you don’t feel like building this database you will find a prebuilt one inside the training materials.
  - The provided version contains a vulnerability that has been patched, so make sure not to mix them.

Example: rabbitmq-c

```
$ git clone https://github.com/alanxz/rabbitmq-c.git
$ cd rabbitmq-c
$ mkdir build
$ cd build
$ export PKG_CONFIG_PATH=/usr/local/opt/openssl/lib/
pkgconfig/
$ cmake ..
$ codeql database create $DATABASES/rabbitmq-c.db
    --language=cpp
    --command=make
    --threads=0
```

Other examples

```
# CSharp
$ codeql database create $DATABASE_NAME
    --language=csharp
    --command="dotnet build /t:rebuild"

# Java
$ codeql database create $DATABASE_NAME
    --language=java
    --command="gradle clean test"
```

CodeQL Queries

Structure of a CodeQL query

- A query has three parts
  - **from** — declaration of the main components of the query
  - **where** — the main logic of your query
  - **select** — representation of the results of your query
- A query file has the extension **.ql** and contains a query, and optionally imports, classes, and predicates.
- A query library has the extension **.qll** and does not contain a query clause, but may contain imports, classes, and predicates.
- import statements allow the classes and predicates defined in one file to be used in another.

Example

```
select "Hello OffensiveCon"
```

Example

```
select "OffensiveCon".length()
```

Example

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

Example

```
from string greeting, string who
where greeting = "Hello" and who = "OffensiveCon"
select greeting + " " + who + "!"
```

Example

```
from string greeting, string who
where
   (greeting = "Hello" and who = "OffensiveCon")
   or
   (greeting = "Goodbye" and who = "OffensiveCon")
select greeting + " " + who + "!"
```

Example

```
import cpp
from Function fun
where fun.getName() = "malloc"
select fun
```

Building Blocks

Variables

- One of the most fundamental pieces of code.
- CodeQL allows you to reason about them.
- There are different types of variables.
- Each type has a different type in CodeQL that allows you to query only the things that you are interested in.

Variables: kinds

- **Variable**
  - A regular variable.
  - Parameters are also considered variables.
  - Globals are also included.
- **LocalScopeVariable**
  - Local variable or a func parameter.

Variables: kinds

- **StackVariable**
  - Stack allocated variable.
  - Meant to exclude static and thread local variables.
- **LocalVariable**
  - Like a LocalScopeVariable but arguments are not considered LocalVariable.
- **GlobalVariable**
- **GlobalOrNamespaceVariable**

Variables

- Commonly used methods:
  - getAnAccess() — VariableAccess
  - getName() — string
  - getType() — Type
  - getAnAssignedValue() — Expr

Variables: exercises

- Explore **Variable.qll**.
  - Right click on something interesting and hit “Quick Evaluation”
- **ex0.ql** — Get all the variables declared in a project.
- **ex1.ql** — Filter those variable with a predicate based on the name.
  - Hint: Strings have a method called **matches** that allow you to use “_” and “%” wildcards.
- **ex2.ql** — Get all the local variables in the project. What changes from ex0.ql?
- **ex3.ql** — Get all the global variables.

Variables: accesses

- A Variable represents the **declaration** of a variable.
- In order to get **uses** of any of them we can ask QL to give us a **VariableAccess**.
- An access can be a read or write.

Variables: exercises

- **ex4.ql** — Get a reference to the global variable “**current_task**” and get all the accesses to it.
  - Why are we getting “repeated” accesses on the same line?
- **ex5.ql** — Get all the functions that use the variable “**current_task**”
  - Hint: access.getEnclosingFunction()
- **ex6.ql** — Get a list of all the global variables and with the number of times they are accessed.
  - Hint: count(expr)

Functions

- **Function** — Both **functions** and **methods**.
- **TopLevelFunction** — Regular functions.
- **MemberFunction** — Methods.
- **VirtualFunction** — Virtual methods.
- **Constructor** — Class constructors.

Functions: calls & accesses

- **FunctionCall** represents a call to a **Function**. It is an expression.
- **FunctionAccess** represents an access to a **Function** as a value.
- If you are interested on how the **Function** is used you need to reason about **FunctionCall** or **FunctionAccess**.

Functions: exercises

- Explore **Function.qll**.
  - Right click on something interesting and hit “Quick Evaluation”
- **ex0.ql** — get all the functions that have **ioctl** in its name.
  - We are getting all the prototypes too. Filter them by adding the **hasDefinition** predicate.
- **ex1.ql** — explore function calls to the functions in ex0.ql
  - Use **Function::getACallToThisFunction** to get calls to the target function.
  - Use **Expr::getEnclosingFunction** to see who makes the call.
- **ex2.ql** — Build a query that finds all the functions that have ioctl in their name that are accessed as a function pointer.
  - Use **Function::getAnAccess**.

Checkpoint

- If you got this far, now you have fundamental tools you need to move around a codebase by using CodeQL.

- In the next slides we will incrementally increase the complexity of the queries to get even more interesting information out of the database.

Predicates

Predicates

- They establish a relationship between its parameters by means of a formula.
- A predicate represents the **set of tuples** that satisfy its formula.
    - A database table if you will.
- They are similar to functions but the analogy will become weird if you push it too much.

Predicates

- Since there is no state, just a formula, you can evaluate anything in QL and get the results.
    - This is an incredible feature and should be used extensively to understand the bits and bolts of more complex queries.
- In predicates with a return type, the return value is implicit.
    - You can access it with the implicit variable **result.**

Predicates

```
// (Block): is `block` an empty block?
predicate isEmpty(Block block) {
  block.getNumStmt() = 0
}

// (Expr, Call): get the first argument of a function call.
Expr getFirstArg(Call call) {
  result = call.getArgument(0)
}

// (Expr, Call): same as before but expressed differently.
predicate getFirstArg(Expr expr, Call call) {
  expr = call.getArgument(0)
}
```

Predicates != Functions

```
// (Call, Expr): Get all Call's that have `expr` as its first argument.
Call getCallFor(Expr expr) {
    expr = result.getArgument(0)
}

// (Call, Expr): Get all Call's that have `expr` as its first argument.
predicate getCallFor(Call call, Expr firstArg) {
    firstArg = call.getArgument(0)
}

// (int, Call, Expr): Get the index of `arg` when called as the first argument
// of `call`.
int getArgIndex(Call call, Expr arg) {
    arg = call.getArgument(result)
}
```

Useful predicates

- none() — represents the empty set also used to represent “false”.
- any() — used to say “true”
- any(variable | condition) — represents the set of values that satisfy “condition”.

Type Checks

- Checking a type in CodeQL is akin to asking if an element belongs to a set.
- Its use is not discouraged.
- Specially powerful when used with user defined types.

Type Checks: exercises

- ex0.ql — create a query that finds all **FunctionCall** that have a **SizeOfOperator** as any of its parameters.
- ex1.ql — improve the previous query by including expressions that contain the **SizeOfOperator** expression as a child.
    - Hints: use **getAChild**() and a **transitive closure** to inspect all the children.
- ex2.ql — make the previous query more specific by only displaying calls to **kmalloc**.
- ex3.ql — display only those calls with a sizeof that has a **PointerType**
- ex4.ql — display those calls to **kmalloc** that have a compile time allocation size.
    - Hint: use **getValue()** on the size expression.
- ex5.ql — display calls to **kmalloc** that contain a **MulExpr** in its size argument.
- Explore the following queries from the standard library. Can you extend them to handle **kmalloc**?
    - cpp/ql/src/Critical/SizeCheck.ql
    - cpp/ql/src/Critical/SizeCheck2.ql

Equality

- This is annoying but more likely than not, it will bite you.
- **A** = **B** holds if there is an **a** ∈ **A** and **b** ∈ **B** such as **a** = **b**
  - True if there is a pair (a, b) such as a = b
- **A != B** holds if there is an **a** ∈ **A** and **b** ∈ **B** such as **a != b**
  - True if there is a pair (a, b) such as a !=b
- **not A = B** holds if there is not an **a** ∈ **A** and **b** ∈ **B** such as **a = b**
  - True if there is not a pair (a, b) such as a = b
- https://help.semmle.com/QL/ql-handbook/formulas.html#equality

if … then … else

- More syntactic sugar!
  - if **A** then **B** else **C**
  - (**A** and **B**) or (not **A** and **C**)

Classes

- A way of grouping things that share a certain characteristic.
- If your query contain very clear “things” you want to operate then most likely you should define a class for them.
- A predicate used to define those characteristics is called a, surprise, surprise, a “characteristic predicate”.
- They can have “methods” (predicates).

Classes: example

```
class EmptyBlock extends Block {
  EmptyBlock() {
    this.getNumStmt() = 0
  }

  // Return the IfStmt containing this block (if any).
  // (EmptyBlock, IfStmt)
  IfStmt getParentIf() {
    result.getThen() = this or result.getElse() = this
  }
}
```

Classes: example

```
class MemoryAllocFunction extends Function {
  MemoryAllocFunction() {
    getName() = "malloc" or
    getName() = "calloc" or
    getName() = "VirtualAlloc"
  }
}
```

Classes: exercise

- Take a look at https://docs.microsoft.com/en-us/previous-versions/bb288454(v=msdn.10)?redirectedfrom=MSDN
- It lists functions considered dangerous and that should be avoided.
- Write a class with as many banned functions as you like and give it a spin on a database of your choice.

Banned functions

```
class BannedStringCopyFunctions extends Function {
  BannedStringCopyFunctions() {
    exists(string name |
      hasGlobalOrStdName(name) and
      (name = "strcpy" or name = "strcpyA" or name = "strcpyW" or name = "wcscpy" or
      name = "_tcscpy" or name = "_mbscpy" or name = "StrCpy" or name = "StrCpyA" or
      name = "StrCpyW" or name = "lstrcpy" or name = "lstrcpyA" or name = "lstrcpyW" or
      name = "_tccpy" or name = "_mbccpy" or name = "_ftcscpy" or name = "strncpy" or
      name = "wcsncpy" or name = "_tcsncpy" or name = "_mbsncpy" or name = "_mbsnbcpy" or
      name = "StrCpyN" or name = "StrCpyNA" or name = "StrCpyNW" or name = "StrNCpy" or
      name = "strcpynA" or name = "StrNCpyA" or name = "StrNCpyW" or name = "lstrcpyn" or
      name = "lstrcpynA" or name = "lstrcpynW")
    )
  }
}
```

Quantifiers

- There are several of them but the most used is **exists**.
  - **exists**(<variable declarations> | <formula>)
    - Reads as “there is an X such as”.
    - Holds if the variables declared satisfy the formula.
    - Used mostly for readability.
    - Think about them like small auxiliary predicates.
    - You can use the **not** operator to negate the predicate.
  - https://help.semmle.com/QL/ql-handbook/formulas.html#quantified-formulas

Quantifiers: example

```
class JumpTarget extends Stmt {
  JumpTarget() {
        exists(GotoStmt g | g.getTarget() = this)
    }
}
```

Quantifiers: exercise

- ex0.ql — write a class that represents functions that are never used (nor called nor referenced).
  - Hints: use **FunctionCall** and **FunctionUse**
- ex1.ql — write a class that represents variables without accesses.
  - Hints: use **LocalVariable** and **VariableAccess**.

Recursion

- Useful technique to represent transitive properties.
- Helps to model iteration.
- Evaluation continues until it reaches a fixed point where there are no more new tuples.
- Initially may be hard to grasp.

Recursion: strategy

- Define the initial base case as the first condition.
- Make sure the expression is not infinite.
- Recurse!

Recursion: example

```
Function getFunctionWrapper(Function target) {
  exists(FunctionCall call, ReturnStmt return |
    call = target.getACallToThisFunction() and
    call.getEnclosingFunction() = result and
    return.getEnclosingFunction() = result and
    DataFlow::localExprFlow(call, return.getExpr())
  )
}

Function getFunctionWrappers(Function function) {
  result = function
  or
  exists(Function wrapper |
    wrapper = getFunctionWrapper(function) and
    result = getFunctionWrappers(wrapper)
  )
}

from Function target, Function wrapper
where
  wrapper = getFunctionWrappers(target) and wrapper ≠ target
select wrapper, target
```

Transitive Closures

- Simply put it is a way to apply the same predicate zero or more times in a transitive way.
- It allows you to succinctly express relationships between elements of the same set that are “connected” between them by means of a relationship / predicate.
- Example: get the set of all the superclasses of “class”
  - class.getASupertype()
- Example: get the set of all the superclasses of “class” “recursively”
  - class.getASupertype+()

Dataflow Analysis

Dataflow Analysis

- Allows us to elaborate queries regarding the **propagation** of data.
- Allows us to answer questions like these
  - Does this expression reach X?
  - Where does this expression reach in the program?
- Fundamental in more complex queries.
- https://help.semmle.com/QL/learn-ql/cpp/dataflow.html

Dataflow Analysis: flavors

- **DataFlow**
  - **Local**
    - DataFlow::localFlow
    - TaintTracking::localTaint
  - **Global**
    - DataFlow::Configuration
    - TaintTracking::Configuration

Dataflow Analysis

- **Global**
  - Follows propagation **across functions**.
  - Computationally **expensive**.
- **Local**
  - Follows propagation at the **function** level.
  - Relatively **cheap**.

Dataflow Analysis

- **Regular data flow**
  - Data propagation is value preserving.
    - Y = X; — There is data flow into Y.
    - Y = X + Z; — There is no data flow into Y.
- **Taint analysis**
  - Extends data flow to non value preserving expressions.
    - Y = X + Z; — There is propagation into Y.

Local Dataflow

- It tracks the value of variables.
  - **int number** — tracks the integer in the **variable**.
  - **void *buffer** — tracks the value of the **pointer**.
- To use it just **import semmle.code.cpp.dataflow.DataFlow**

Local Dataflow: limitations

- Cannot propagate flow if there is no sources
  - Think about standard library functions.
  - Can be avoided by modeling missing functions.
    - cpp/ql/src/semmle/code/cpp/models/implementations/Strcpy.qll
    - cpp/ql/src/semmle/code/cpp/models/implementations/Memcpy.qll
- It is not global so results may be incomplete.

Local Dataflow: example

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

from Function fopen, FunctionCall fc, Expr src
where fopen.hasQualifiedName("fopen")
  and fc.getTarget() = fopen
  and DataFlow::localExprFlow(
        src,
        fc.getArgument(0)
  )
select src
```

Local Dataflow: example

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

from FormattingFunction format, FunctionCall call, Expr formatString
where call.getTarget() = format
  and call.getArgument(format.getFormatParameterIndex()) = formatString
  and not exists(DataFlow::Node source, DataFlow::Node sink |
    DataFlow::localFlow(source, sink) and
    source.asExpr() instanceof StringLiteral and
    sink.asExpr() = formatString
  )
select call, "Argument to " + format.getQualifiedName() + " isn't hard-
coded."
```

Local Dataflow: exercises

- ex0.ql — given a call to **kmalloc** build a query that shows where does its return value flows.
  - Hint: use DataFlow::localExprFlow
- ex1.ql — extend the previous query to show only the calls that do not flow into an if statement.
  - Hint: not exists | IfStmt | DataFlow::localExprFlow | getControllingExpr
  - Explore the results. Do you notice something weird? Can you explain why the query “fails”?
- ex2.ql — Improve the results of the previous query by using a transitive closure.
  - Hint: **sink.getControllingExpr().getAChild*()**
  - Explore the results. It seems like most false positives are related with wrappers, how can you solve this?

Taint Analysis

- What happens when a variable A encounters
  - B = A + C
  - B = A > 0
- Is flow propagated to B?
- Since data flow is value preserving there is no flow.
- In order to track flow in similar situations we need to perform taint analysis.

Taint Analysis: example

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

from FunctionCall call, DataFlow::Node source, DataFlow::Node sink
where
  call.getTarget().getName() = "snprintf" and
  call.getArgument(2).getValue().regexpMatch("(?s).*%s.*") and
  TaintTracking::localTaint(source, sink) and
  source.asExpr() = call and
  sink.asExpr() = call.getArgument(1)
select call, upperBound(call.getArgument(1).getFullyConverted())
```

Global DataFlow

- Models propagation across functions.
- To make the problem tractable we need to define a source and a sink.
- This is done by extending the DataFlow::Configuration class.
  - predicate **isSource**(DataFlow::Node source)
  - predicate **isSink**(DataFlow::Node sink)

Global DataFlow: example

```
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'"
```

Global TaintTracking: example

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

  override predicate isSource(DataFlow::Node node) {
    node.asExpr().(FunctionCall).getTarget().hasGlobalName("ntohl")
  }

  override predicate isSink(DataFlow::Node node) {
    exists(ArrayExpr ae | node.asExpr() = ae.getArrayOffset())
  }
}

from DataFlow::Node ntohl, DataFlow::Node offset, NetworkToBufferSizeConfiguration conf
where conf.hasFlow(ntohl, offset)
select offset, "This array offset may be influenced by $@.", ntohl,
  "converted data from the network"
```

Real world bug!

rabbitmq-c

- Client library to talk to RabbitMQ.
- Let’s rediscover this vulnerability I found by using CodeQL.
- https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-18609
- Solutions to the exercises can be found in the **testcase** folder.

rabbitmq-c: where do we begin?

- Since this is a network client library we can start our research by looking for data read from the network.
- Build a query that shows you all the calls to **recv**.
    - Solution: testcase/ex0.ql
- There is only one result, a function called **amqp_tcp_socket_recv**.
- Build another query to investigate how is this function used.
    - Solution: testcase/ex1.ql
- Looks like this function is not called? What about function accesses?
- Build a query that shows you where is this function accessed.
    - Solution: testcase/ex2.ql

Sleazy amqp_tcp_socket_recv

- So this function is referenced via a function pointer in **amqp_socket_class_t**
- So we need to know what is the name of the function pointer.
- Build a query that gets you the definition of the struct **amqp_socket_class_t**.
  - Hint: use Struct.getName() = “amqp_socket_class_t”.
  - Solution: testcase/ex3.ql

Calls to recv

- We now that the target function is stored inside a struct.
- Now we need to see where it is used.
- Build a query that gets all the accesses to the structure field **recv**.
  - Hints
    - Get the struct.
    - Get a field of the struct.
    - Now get an access to the field.
  - Solution: testcase/ex4.ql

The end of the tunnel!

- So there is only one place that used the function pointer.
  - amqp_socket_recv
- Inspect what the function does by getting all its callers.
  - Solution: testcase/ex5.ql
- There is only one place and it populates an interesting structure called sock_inbound_buffer with network data.

Taint source

- Knowing that amqp_bytes_t::bytes contain network data create a class that encapsulates all accesses to this field.
  - Hints
    - Extend FieldAccess and check that the field name is **bytes**
    - Use **getQualifier** to get the structure that has this field.
    - Check that the structure type is **amqp_bytes_t**
  - Solution: testcase/ex6.ql

Taint sink

- A classic bug is reading a size from the network and allocating memory with it.
- Missing the proper checks can lead to vulnerabilities.
- rabbitmq-c uses **amqp_pool_alloc_bytes** to allocate memory.
- Create a class that represents this function.
  - Solution: testcase/ex7.ql

Taint Configuration

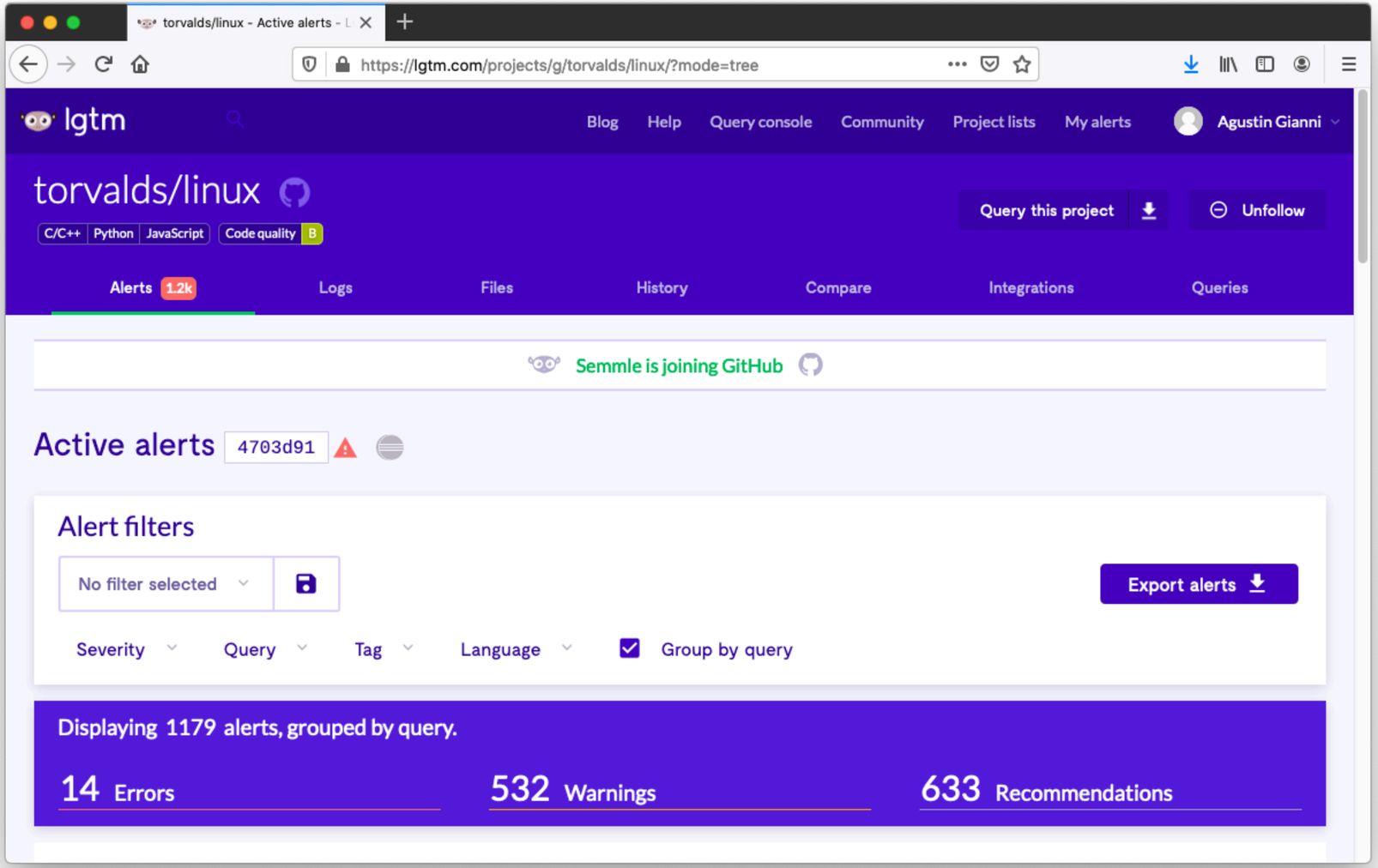
- Implement a taint configuration that
  - Defines as its source of taint the class defined in ex6.ql
    - Hint: check that the type of the source expression is the one defined in ex6.ql.
  - Defines as a sink the class defined in ex7.ql
    - Hint: Get a call and check that the sink is an argument to the call we are interested in.

lgtm.com

LGTM

- www.lgtm.com
- Continuous analysis of indexed open source projects.
- Has a web console that allows you to run queries.
- You can also download the databases for your personal use offline.
- **Warning:** projects will usually be compiled without much intervention. This leads to certain areas of the code to be not present in the database.
    - This is specially important for things like the linux kernel as many of the drivers will not be enabled.

https://lgtm.com/projects/g/torvalds/linux/

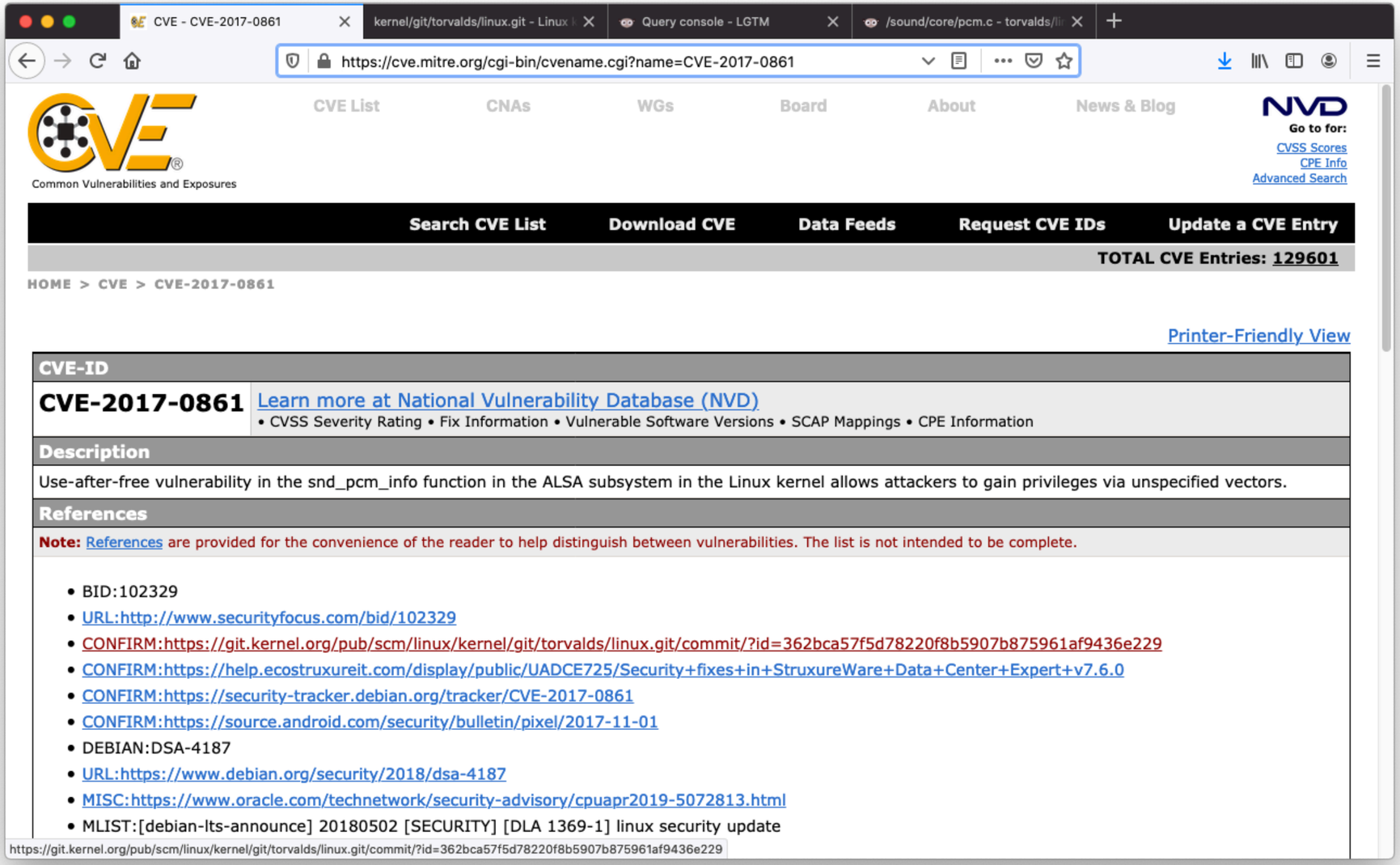


Exploring LGTM

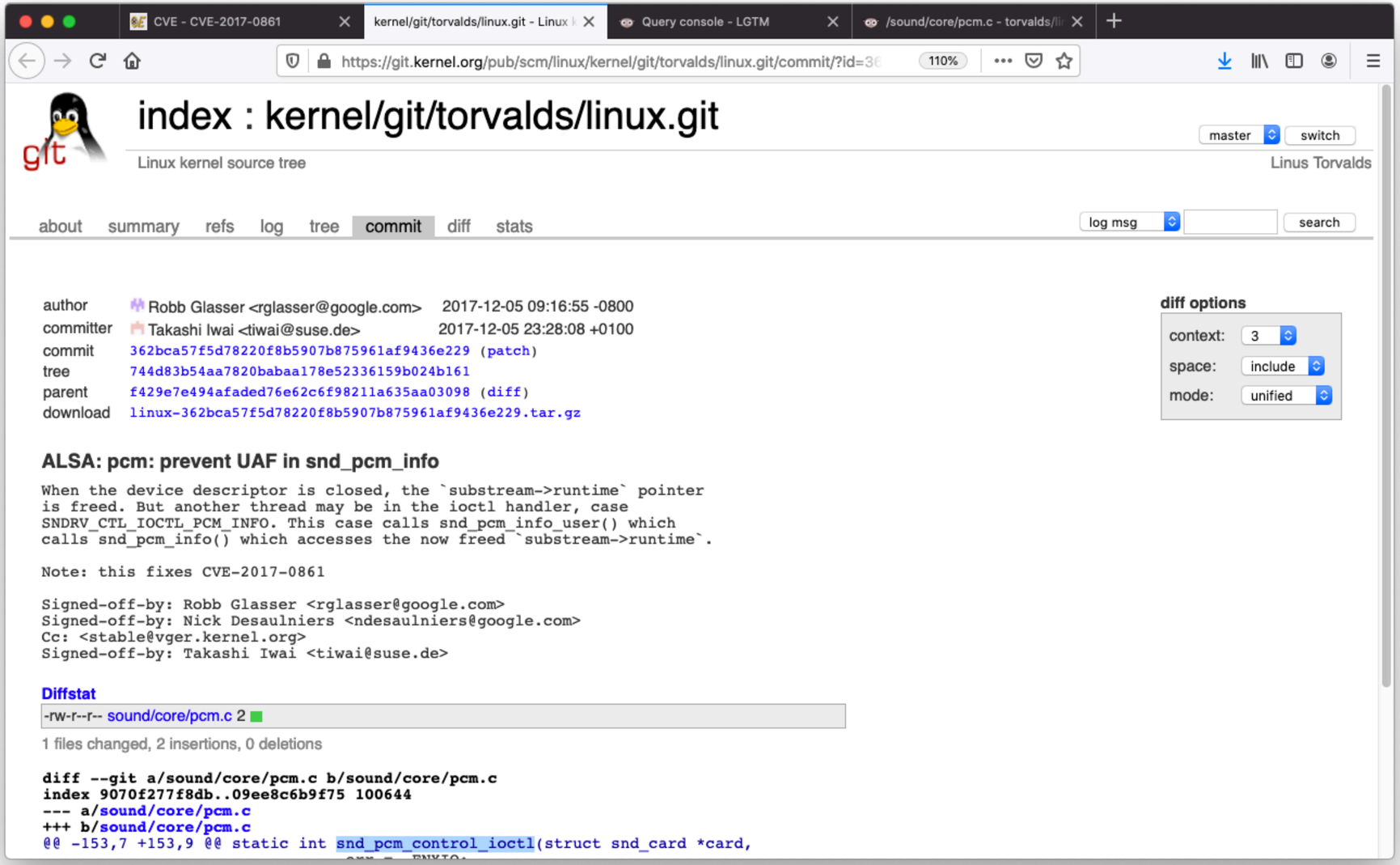
Exercise

- Find information about CVE-2017-0861
- Login to LGTM
- Query for the relevant functions

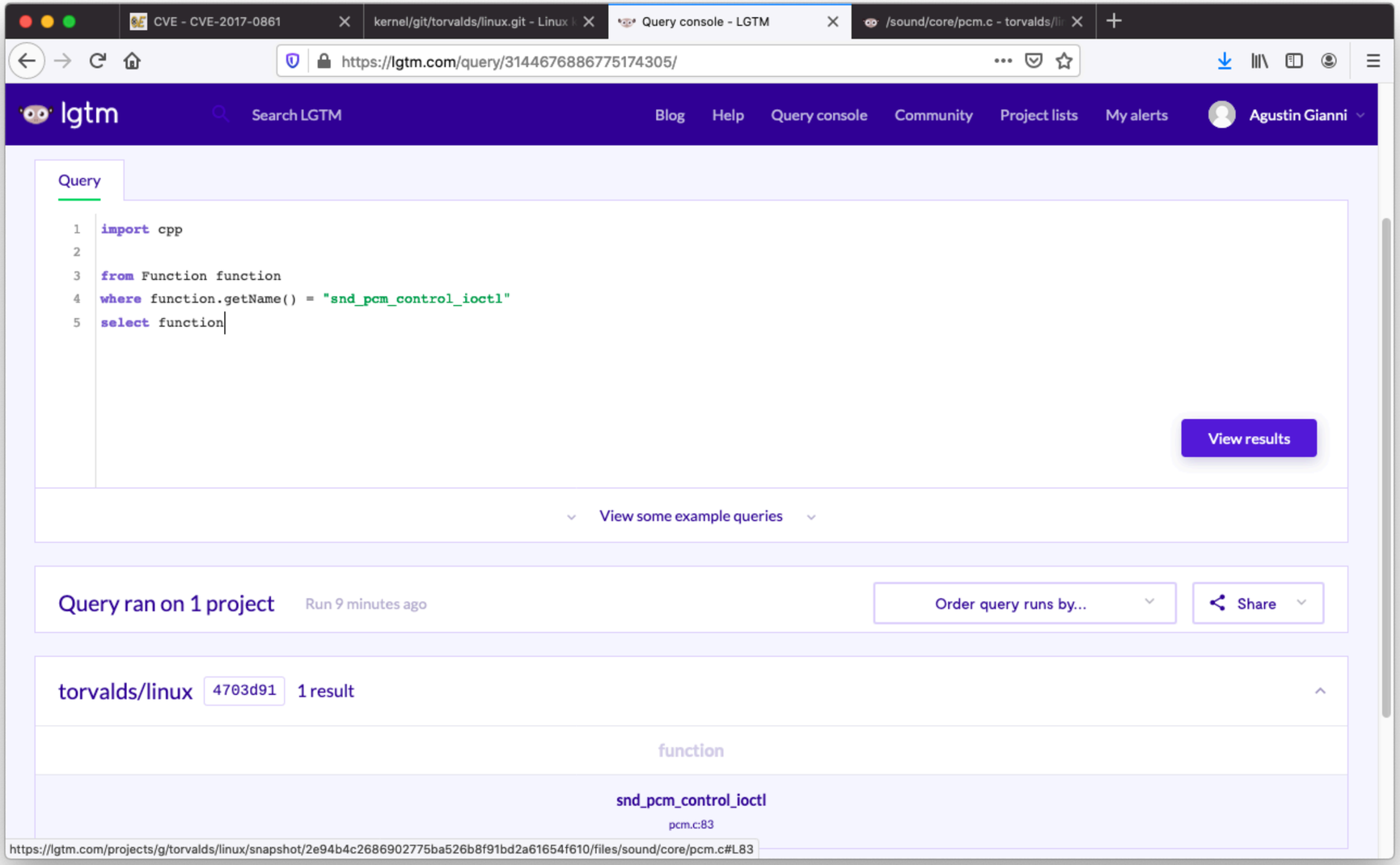
https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-0861



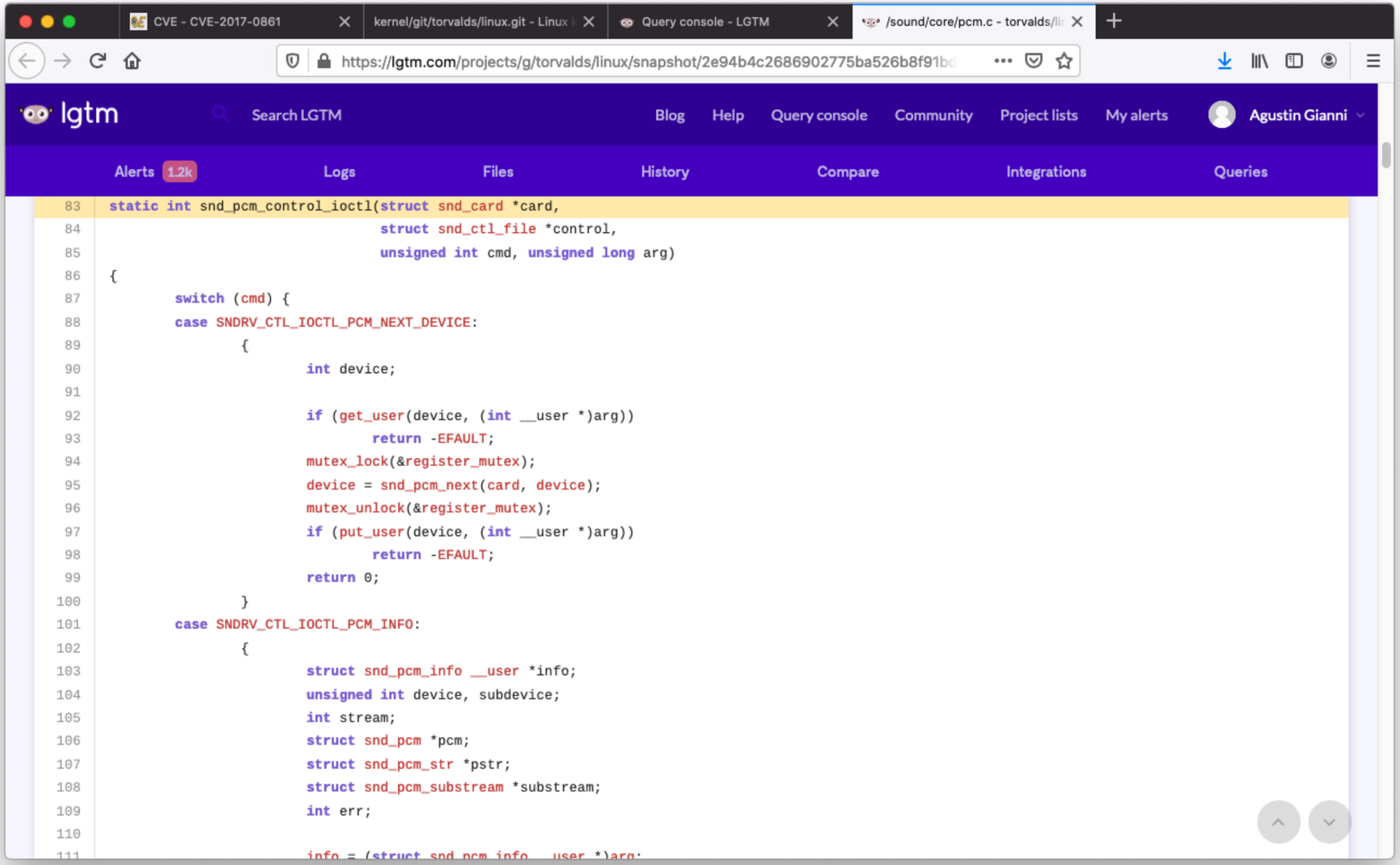
Git Commit


CVE - CVE-2017-0861
kernel/git/torvalds/linux.git - Linux
Query console - LGTM
/sound/core/pcm.c - torvalds/li
https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=3
110%
index : kernel/git/torvalds/linux.git
Linux kernel source tree
master
switch
Linus Torvalds
about
summary
refs
log
tree
commit
diff
stats
log msg
search
author Robb Glasser <rglasser@google.com> 2017-12-05 09:16:55 -0800
committer Takashi Iwai <tiwai@suse.de> 2017-12-05 23:28:08 +0100
commit 362bca57f5d78220f8b5907b875961af9436e229 (patch)
tree 744d83b54aa7820babaa178e52336159b024b161
parent f429e7e494afaded76e62c6f98211a635aa03098 (diff)
download linux-362bca57f5d78220f8b5907b875961af9436e229.tar.gz
diff options
context: 3
space: include
mode: unified
ALSA: pcm: prevent UAF in snd_pcm_info
When the device descriptor is closed, the `substream->runtime` pointer
is freed. But another thread may be in the ioctl handler, case
SNDRV_CTL_IOCTL_PCM_INFO. This case calls snd_pcm_info_user() which
calls snd_pcm_info() which accesses the now freed `substream->runtime`.
Note: this fixes CVE-2017-0861
Signed-off-by: Robb Glasser <rglasser@google.com>
Signed-off-by: Nick Desaulniers <ndesaulniers@google.com>
Cc: <stable@vger.kernel.org>
Signed-off-by: Takashi Iwai <tiwai@suse.de>
Diffstat
-rw-r--r-- sound/core/pcm.c 2
1 files changed, 2 insertions, 0 deletions
diff --git a/sound/core/pcm.c b/sound/core/pcm.c
index 9070f277f8db..09ee8c6b9f75 100644
--- a/sound/core/pcm.c
+++ b/sound/core/pcm.c
@@ -153,7 +153,9 @@ static int snd_pcm_control_ioctl(struct snd_card *card,

https://lgtm.com/query/3144676886775174305/



LGTM Result


CVE - CVE-2017-0861
kernel/git/torvalds/linux.git - Linux
Query console - LGTM
/sound/core/pcm.c - torvalds/li
https://lgtm.com/projects/g/torvalds/linux/snapshot/2e94b4c2686902775ba526b8f91bc
lgtm
Search LGTM
Blog
Help
Query console
Community
Project lists
My alerts
Agustin Gianni
Alerts 1.2k
Logs
Files
History
Compare
Integrations
Queries
83 static int snd_pcm_control_ioctl(struct snd_card *card,
84 struct snd_ctl_file *control,
85 unsigned int cmd, unsigned long arg)
86 {
87 switch (cmd) {
88 case SNDRV_CTL_IOCTL_PCM_NEXT_DEVICE:
89 {
90 int device;
91
92 if (get_user(device, (int __user *)arg))
93 return -EFAULT;
94 mutex_lock(&register_mutex);
95 device = snd_pcm_next(card, device);
96 mutex_unlock(&register_mutex);
97 if (put_user(device, (int __user *)arg))
98 return -EFAULT;
99 return 0;
100 }
101 case SNDRV_CTL_IOCTL_PCM_INFO:
102 {
103 struct snd_pcm_info __user *info;
104 unsigned int device, subdevice;
105 int stream;
106 struct snd_pcm *pcm;
107 struct snd_pcm_str *pstr;
108 struct snd_pcm_substream *substream;
109 int err;
110
111 info = (struct snd_pcm_info __user *)arg;

Query MetaData

- Expressed as a docstring comment.
- Changes the behavior of the engine.
- https://help.semmle.com/QL/learn-ql/writing-queries/query-metadata.html
- https://help.semmle.com/QL/learn-ql/writing-queries/introduction-to-queries.html

Query MetaData

Add header comment /** … */ at beginning of QL file to provide metadata:

- **@name:** a descriptive name of the query
- **@description:** a short description of what the query does
- **@id:** a unique identifier for the query (important for LGTM)
- **@kind:** how to display results, e.g. “@kind problem” for alert queries

Questions? Concerns? Comments?

Ping @octocat on Slack