GitHub

Code scanning with CodeQL

Presented by Rasmus Lerchedahl Petersen (@yoff)

Plan for today

Lecture 1: Understand the potential of CodeQL from a hacker's perspective
Lecture 2: Live coding, see it in action (and see the syntax).

Exercises: Try it yourself!

Interrogating a code base

A two-step process:

1. Extraction: Convert the code base to a database
   Language specific and requires the source code

2. Querying: Evaluate a CodeQL query against the database
   The query codifies your security question

CodeQL

It is a programming language.
It facilitates object-oriented logic programming.

CodeQL programs are

- evaluated *in the context of* a code base.
- logical *predicates*, statements about (parts of) that code base.

Evaluating a CodeQL program results in all the parts that makes the statement true.

Static analysis

But not really defining a semantics

- Static “bug finding”
- Search a code base for patterns (these have a semantics)
- Not complete (programs are not verified)
- Not sound (there will be false positives)
- But useful
- Patterns can be quite advanced
  (to the point where they basically define a semantics)

Common predicates

An extracted database will typically contain mostly syntactic predicates.

We offer libraries to build upon those and construct

- A control flow graph (sometimes already included)
- SSA analysis
- A data flow graph
- Several security specific concepts
  (remote flow source, code injection sink, request handler, etc...)

Writing predicates

We can refer to

- the AST, e.g. find me a call
- the CFG, e.g. find me a successor, check dominance
- dataflow, e.g. where can this value come from?
- range analysis (only some languages), e.g. what numerical values can this expression take?

Tables and tuples

Predicates are n-ary relations.
They each have a table where each row is a tuple.
A tuple contains related values.

Example relation < (“less than”):
Rows would be
(1, 2)
(2, 3)
(3, 4)
…
(but we prefer finite relations)

Example relations

AST (given by the extractor)

Table for elements (*id*, *location*, *element*):
(1, loc1, operator +=)
(2, loc2, name “x”)
(3, loc3, int 1)

Table for children (*parent*, *child*):
(1, 2)
(1, 3)

Example relations

We can encode graphs by encoding the edge relation.
(Either directly or as successor/childOf)

This means that can encode

- the control flow graph
- the dataflow graph
- the call graph

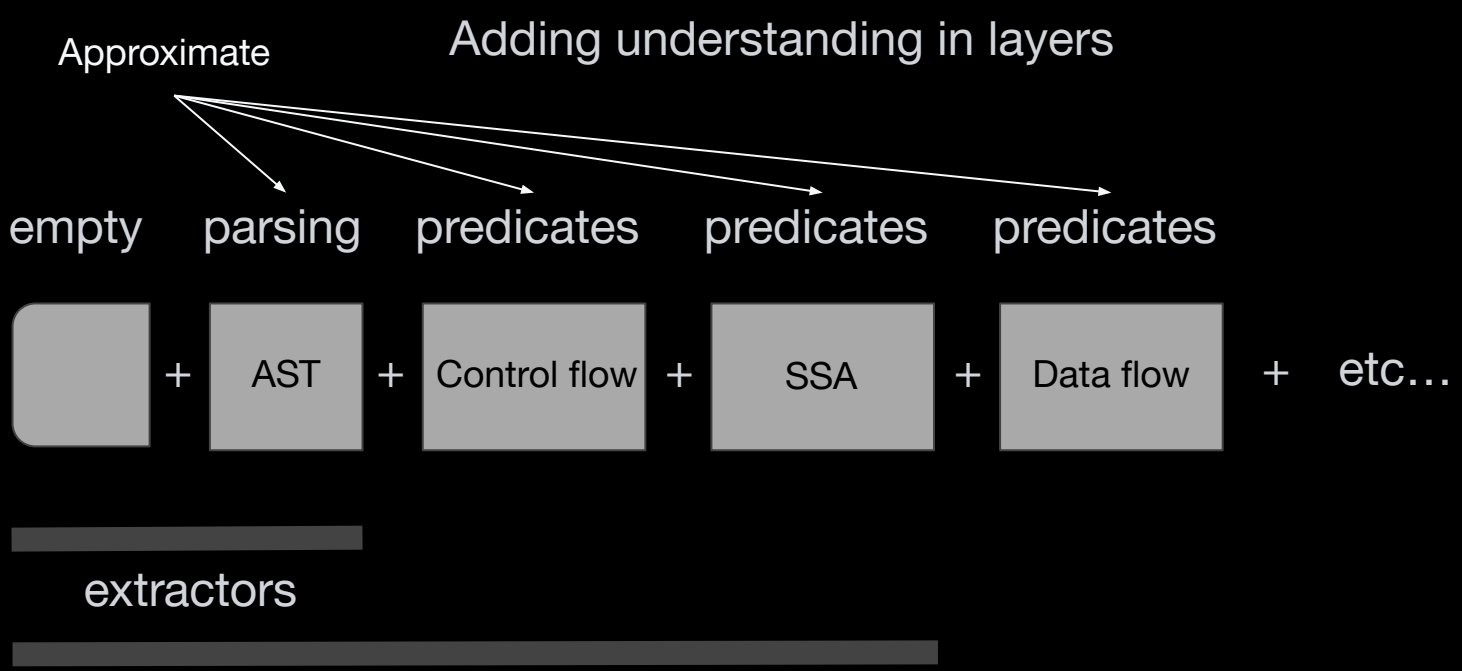
Example relations

Reachability

We can ask for the transitive closure of a predicate.
That means that we can ask for reachability in a graph.

Control flow: Can we hit a use before an allocation?
Dataflow: Can a user controlled value end up here?

Building a semantics



Security research using CodeQL

Security researchers write *queries* in CodeQL and run them against repositories.

Commonly, they target either

- a single interesting code base (perhaps as part of an audit)
- or a single vulnerability (wish to look at many repositories)

Targeting a code base

Common steps:

- Build a database (or download a pre-built one)
- Load it into VSCode using the CodeQL extension
- Write exploratory queries

Targeting a vulnerability

Common steps:

- Formulate the vulnerability as a query
- Use MRVA to run it on several interesting code bases

- Review results and iterate

Exercises

- Install VSCode *or* open a code space
- Install CodeQL extension
- Download a database
- Run a quick-query
- Check out our libraries and run a standard query

(clone https://github.com/github/codeql)

Exercises

Attack your favourite code base

- Get hold of a data base
- Go explore!

Implement your favourite vulnerability

- Encode it as a query
- Run it using MRVA

Try a “capture the flag”
https://securitylab.github.com/ctf/