Supporting Analysis of SQL Queries in PHP AiR

Mark Hills (Appalachian State University, NC, USA)

PEM Presentation
Centrum Wiskunde & Informatica
June 14, 2023

NOTE: Based on material from SANER 2017, SCAM 2017, and SERPL 2019

Based on work with David Anderson, MS student at ECU

PHP Analysis in Rascal (PHP AiR)

- PHP AiR: a framework for PHP source code analysis
- Domains:
  - Program analysis (static/dynamic)
  - Software metrics
  - Empirical software engineering

Motivation

- We want to transform uses of older database APIs into equivalent uses of newer, safer APIs

- We want to aid developers in understanding how queries are built in their (maybe unfamiliar to them!) programs

- (For us and other researchers/tool builders) We want to better understand how PHP, database APIs, and query languages are used in existing code to help us build better tools for program analysis, comprehension, and transformation

Context: PHP and MySQL

- MySQL API uses query functions (originally mysql_query, that is the one we focus on here) to execute queries

- Queries given as strings, formed using string building operations

- Queries often have a mixture of static and dynamic pieces

```
$query = mysql_query("
SELECT title
FROM semesters
WHERE semesterid = $_POST[semester]
");
```

Note: Real but horrible query, this has a major security vulnerability…

Query construction patterns (earlier work)

- NOTE: From SANER 2017
- How are queries typically built in PHP scripts?
- What parts of a query tend to be dynamic?
- What features are used to build these dynamic query parts?
- Note: This work focused on identifying *patterns*, not building *models* of queries.

Results of SANER'17 paper

- Queries appear to be built in predictable patterns

- Dynamic parts are mainly in the "right" places, making a transformation to prepared statements possible

- We need a more extensive analysis with more systems and more precise and sound analysis algorithms

- We need better models of the queries themselves (current work) for more precise pattern identification

- We need to build the transformation!

Research questions

- R1: How can we model how queries are built?

- R2: Using these models, how can we generate the queries or query templates (with placeholders for dynamic bits) that could actually be executed?

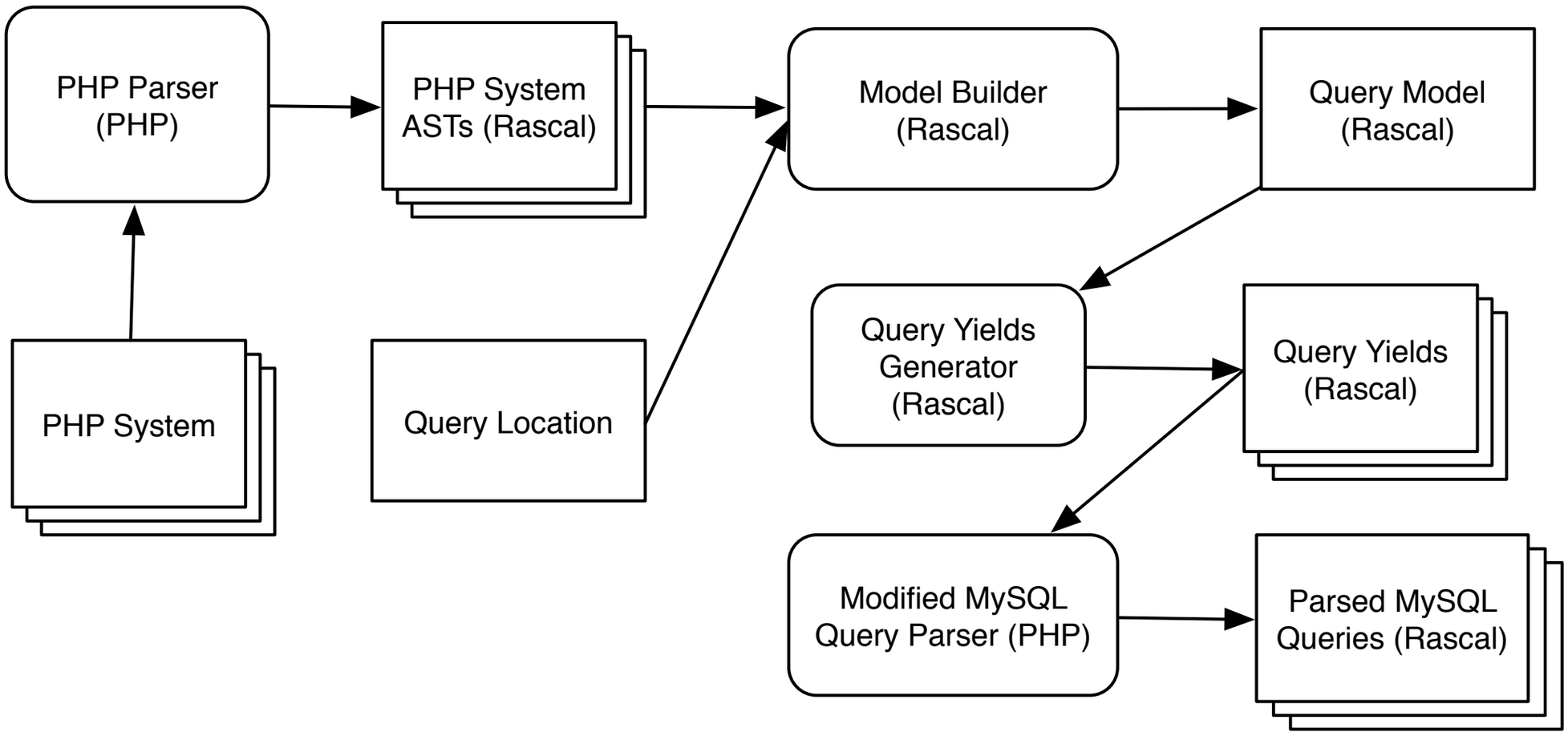
- R3: What parts of the query language (here, SQL) are used in these queries? Which of these parts are static, and which are dynamic?

- R4 (suggested by reviewers): Can we use models to find potential security vulnerabilities in PHP code? We haven't looked at this yet, but it sounds promising, especially with interprocedural support…

Analyzing and parsing queries: methodology

- All analysis code is written using Rascal
  - https://github.com/PLSE-Lab/mysql-query-construction-analysis
- PHP is parsed using a PHP parser written in PHP, parser yields Rascal terms
- MySQL queries are parsed using a fork of the parser found in phpMyAdmin, a web frontend for administering MySQL
- A quick note: this is all part of the PHP AiR project

Analyzing and parsing queries: “The Big Picture”



R1: Building models

Input : *sys*, a PHP system, mapping from file locations to abstract syntax trees
Input : *callLoc*, a location indicating the query call to be analyzed
Output : *res*, a query model

1 *inputCFG* ← `buildCFG4Loc` (*callLoc*)
2 *inputNode* ← the CFG node from *inputCFG* representing the call at *callLoc*
3 *d* ← `definitions` (*inputCFG*)
4 *u* ← `uses` (*inputCFG*, *d*)
5 *slicedCFG* ← `basicSlice` (*inputCFG*, *inputNode*, `usedNames` (*u*, *inputNode*), *d*, *u*)
6 *startingFragment* ← `expr2qf` (*inputNode*)
7 *fragmentRel* ← {}
8 **while** *fragmentRel* `continues to change` **do**
9 | *nodesToExpand* ← (*inputNode.l* × *startingFragment*) ∪ *fragmentRel*⟨3, 4⟩
10 | **foreach** (*nodeLabel* × *nodeFragment*) ∈ *nodesToExpand* **do**
11 | | add `expandFragment` (*nodeLabel*, *nodeFragment*, *d*, *u*) to *fragmentRel*
12 | **end**
13 **end**
14 *fragmentRel* ← `addEdgeInfo` (*fragmentRel*, *slicedCFG*)
15 *res* ← the model, including *fragmentRel*, *startingFragment*, and *callLoc*

Algorithm 1: Extracting a Model of a SQL Query.

See the SCAM 2017 paper for all the details, here comes a summary…

R1: Building models

- Models are (possibly cyclic) graphs of *query fragments* (with a bit of bookkeeping info, like the location of the call)

- A query fragment is a static or dynamic piece of the query

- Intraprocedural, backwards slice throws away code that does not impact query

- CFG and def/use info link names in fragments to defs of those names

- Reachability conditions used to decorate graph edges

R2: Extract yields

- Yields are lists of “pieces”:
  - Static pieces for query text
  - Dynamic pieces for arbitrary expressions
    - Name pieces for names (useful to track separately)
- Generated by traversing the graph, currently cuts off when cycles detected
- Can use edge labels to filter infeasible yields, improving to work in more situations (loops are problematic)

R3: Parsing partial queries

- First, yields are converted to strings: static pieces yield strings directly, dynamic and name pieces are turned into query holes (e.g., ?1, ?2, generally ?n)

- Second, string is parsed by our modified MySQL parser, yielding PHP objects representing MySQL AST (limitation: we assume holes are expressions and do not cross clause boundaries, supporting this is ongoing work, a dedicated parser would be quite helpful here since we depend on the MySQL parser but don’t control this at all!)

- Third, AST pretty-printed to a Rascal term representing AST, similarly to current PHP parser

And now for some controversy

- We want to extend this to be interprocedural, but: for a really dynamic language, where even the decision of what code to include is deferred until runtime, is this even useful? How close can we get to the actual system? (see, e.g., work on JavaScript call graph construction for how challenging this can be!)

- Do we even need to support the entire language for this to be useful for developers? Will keeping this simple work? How precise does this need to be?

Future Directions: Expand the Corpus!

- Earlier corpus was somewhat ad-hoc, based on past work, mix of current and no longer maintained systems

- New corpus based on 1000 most starred repos on GitHub as of April 2018 (based on GHTorrent and BigQuery), identified 78 of these systems that use MySQL or MySQLi libraries

- One question: how representative are these of the systems people actually write themselves? are these good exemplars, or overly general frameworks and libraries?

Future Directions: More APIs!

- Current work has focused on MySQL API
- Already (mostly) expanded to include MySQLi API
  - Challenge — type inference for dynamic languages…
- Planning to add support for PDO, potentially other DB-specific APIs
- May look at ORMs such as Doctrine with custom query languages

PHP Analysis in Rascal: Updates

- Lots of changes so far in 2023!
  - We now support PHP 8
  - We no longer use annotations!
  - PHP AiR is now available as a Maven plugin, so you don’t need to work directly in the project or use it as a project dependency
  - Examples are being moved out to keep the core smaller and easier to maintain
  - Support for working with Git repos is being added because of the amount of code we are now processing (e.g., about 700 million for a recent paper under review)

Discussion

Thank you!
Any Questions?

- Rascal: https://www.rascal-mpl.org/
- Me: https://cs.appstate.edu/hillsma