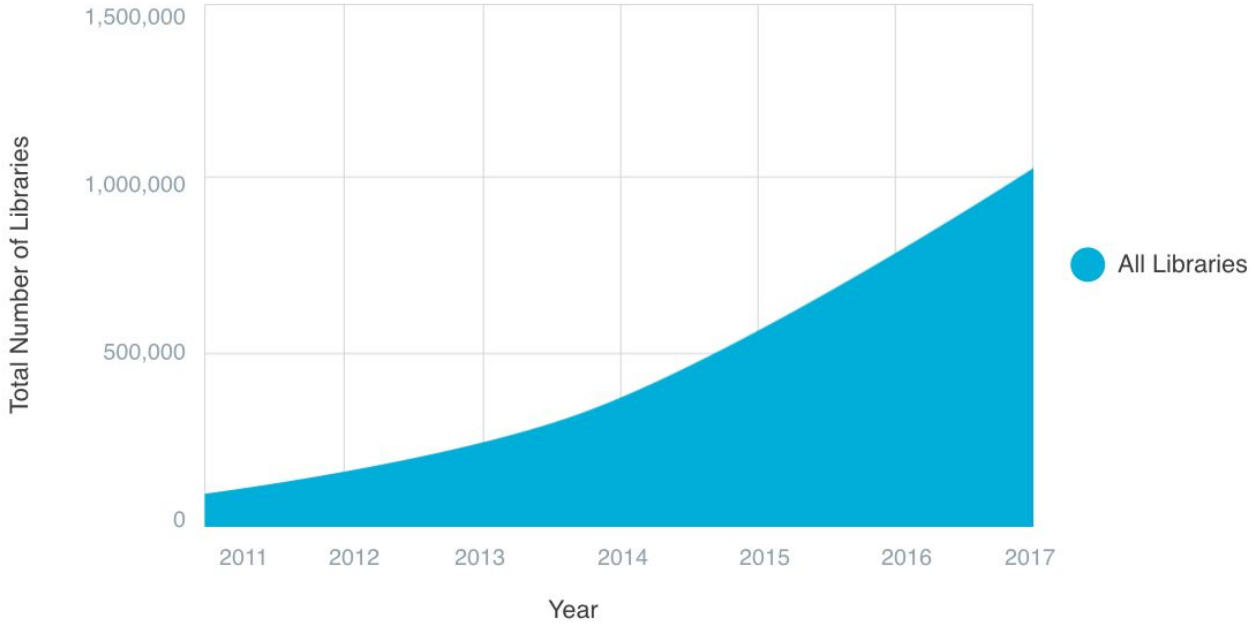
Design and Implementation of Security Graph Language (SGL)

Darius Foo, Ang Ming Yi,
Jason Yeo, Asankhaya Sharma

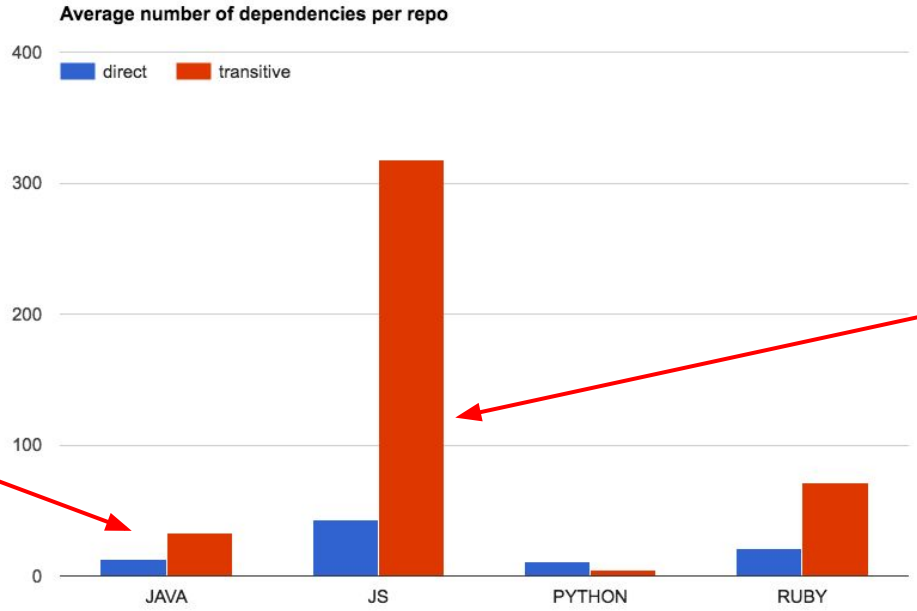
Motivation

- Software is built using large amounts of third-party code (up to 90%)



Motivation

- Software is built using large amounts of third-party code (up to 90%)



For each Java library depended on, 4 others are added

For each JS library depended on, 9 others are added

Motivation

- Unaudited third-party code is a liability
  - Apache Struts (2018)
    - CVE-2018-11776: RCE via URL
    - CVE-2017-5638: RCE via HTTP headers (Equifax breach)
  - Malicious libraries (eslint-scope, crossenv, 2018)
  - Heartbleed (OpenSSL, 2017)
  - GHOST (glibc, 2015)
  - Apache Commons Collections deserialization RCE (2015)

Motivation

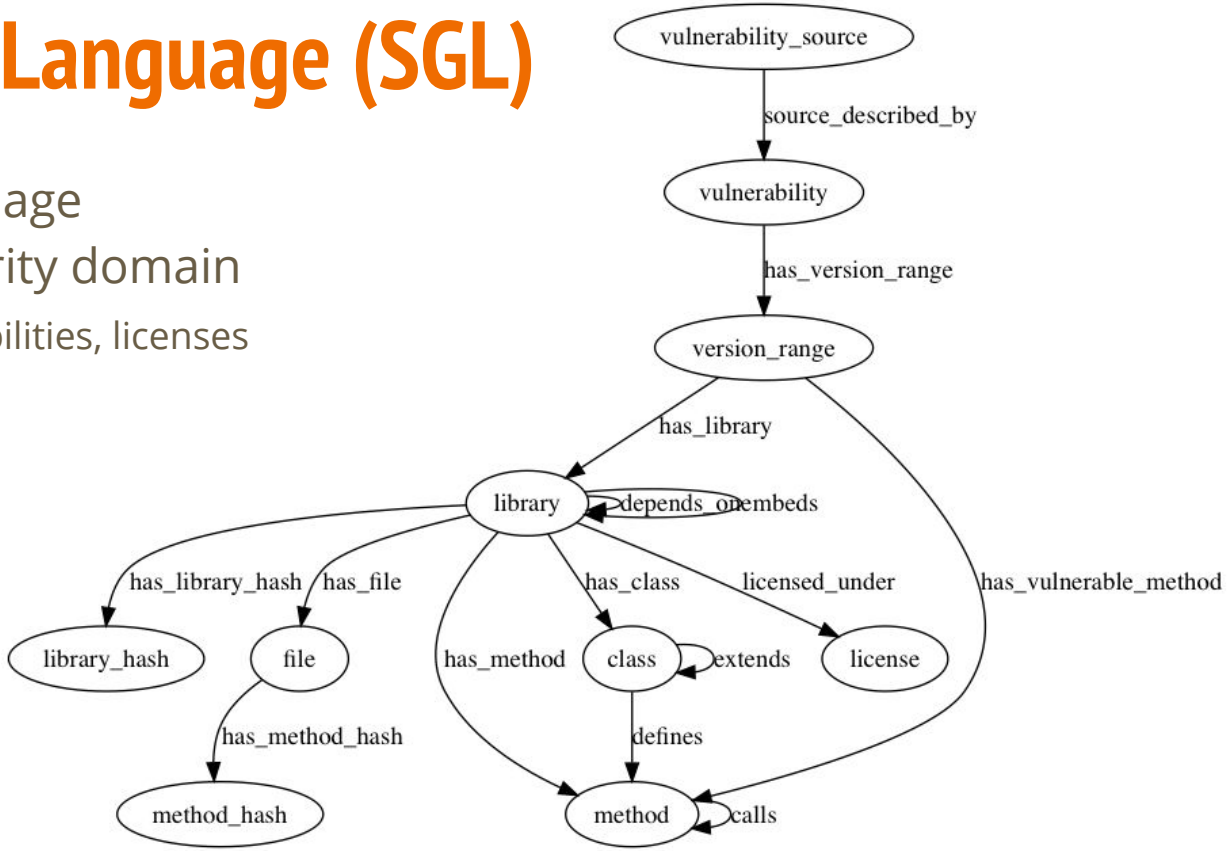
- Manual auditing is infeasible
  - Hundreds of dependencies
  - Constantly changing
- Automated audits
  - Dependency-level
    - Ensure you're not using a vulnerable version
  - Source-level
    - Ensure you're not vulnerable, *despite* using a vulnerable version
    - Ensure you won't be vulnerable as things change
      - *Potential* vulnerabilities, anti-patterns

What we want

- Capture the space in some abstract form
- Be able to interrogate it using flexible queries
- Automate and share these queries to rule out classes of issues

Security Graph Language (SGL)

- Graph query language
- Open source security domain
  - Libraries, vulnerabilities, licenses
  - Methods, classes



Use cases

- Program analysis
  - Syntax trees, call graphs, dataflow graphs
  - Dependency graphs
- Vulnerability description
  - Structured alternative to CVEs

Related work

- Code analysis + graph databases
  - Yamaguchi, Fabian, et al. "Modeling and discovering vulnerabilities with code property graphs." Security and Privacy (SP), 2014 IEEE Symposium on. IEEE, 2014.
- Graph query languages
  - Gremlin
  - Cypher
- Vulnerability description languages
  - OVAL

SGL: implementation

- Typed, declarative Gremlin subset
- Compiles to Gremlin
- Query planning

SGL: reachability

```
library(coord1: ‘commons-collections’, version: ‘3.2.2’)

has_method method(name: ‘readObject’)
```

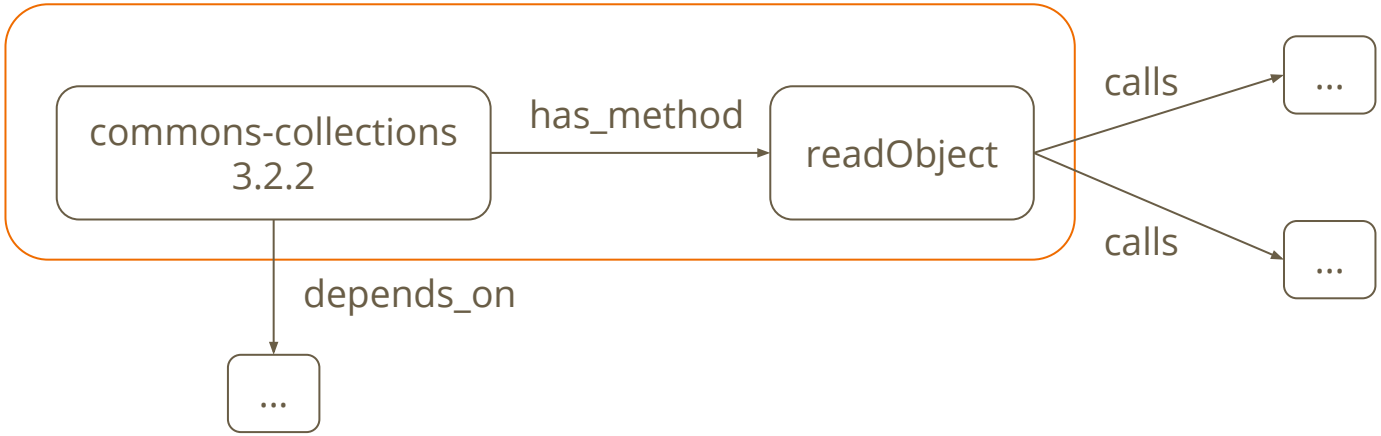
“Does this version of Apache Commons Collections contain a method named readObject?”

SGL: reachability

path

```
library(coord1: ‘commons-collections’, version: ‘3.2.2’)

has_method method(name: ‘readObject’)
```



SGL: results

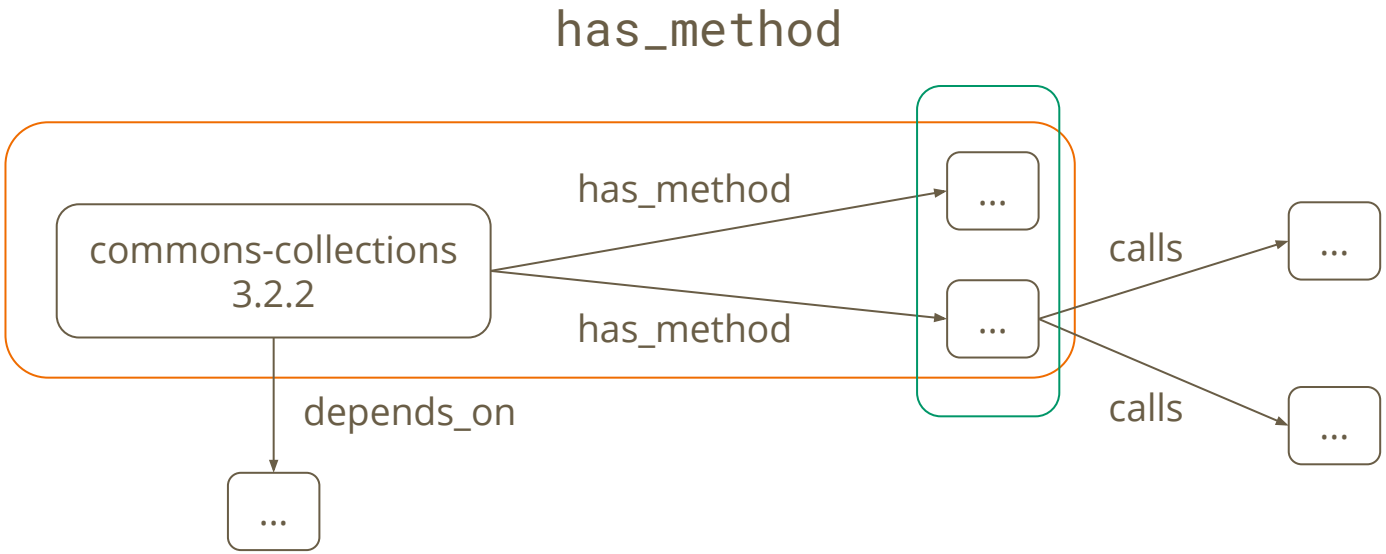
```
library(coord1: ‘commons-collections’, version: ‘3.2.2’)

has_method
```

“What methods does this version of Apache Commons Collections contain?”

SGL: results

```
library(coord1: ‘commons-collections’, version: ‘3.2.2’)
```



SGL: results

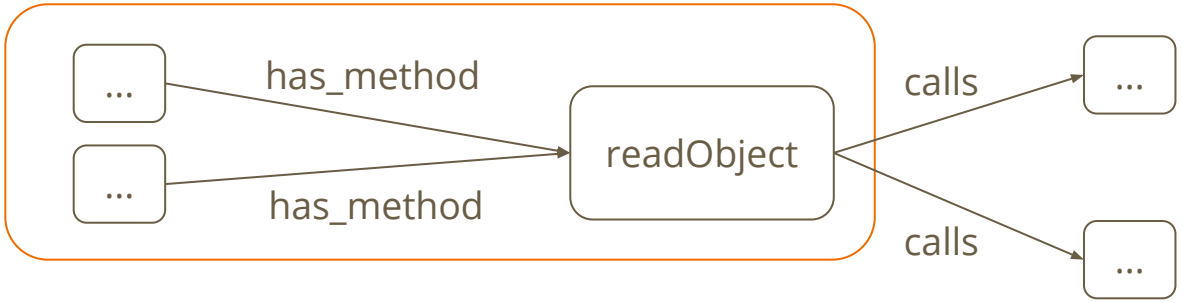
```
library(coord1: ‘commons-collections’, version: ‘3.2.2’)

has_method

method(name: ‘readObject’)
method(name: ‘readExternal’)
method(name: ‘readResolve’)
```

SGL: projection

```
library(_) where(has_method method(name: ‘readObject’))
```



SGL: projection

```
library(_) where(has_method method(name: ‘readObject’))
```

```
library(version: ‘3.2.2’)
library(version: ‘3.2.3’)
library(version: ‘3.2.4’)
```

SGL: transitive closure

```
library(coord1: ‘commons-collections’, version: ‘3.2.2’)
```

```
depends_on
```

“What are the direct dependencies of Apache Commons Collections?”

SGL: transitive closure

```
library(coord1: ‘commons-collections’, version: ‘3.2.2’)

depends_on*
```

“What are **all** the dependencies of Apache Commons Collections?”

SGL: aggregations

```
library(coord1: ‘commons-collections’, version: ‘3.2.2’)

depends_on* limit(5)
```

aggregation

“What are 5 dependencies of Apache Commons Collections?”

SGL: aggregations

```
library(coord1: ‘commons-collections’, version: ‘3.2.2’)

depends_on* count
```

aggregation

“How many dependencies of Apache Commons Collections are there?”

SGL: bindings, abstraction

```
let spring = library(
  'java',
  'org.springframework',
  'spring-webmvc',
  '4.3.8.RELEASE'
) in
spring depends_on*
```

```
let depends_on_method =
  depends_on has_method in
spring depends_on_method
```

Compilation

```
let spring = library(
  'java',
  'org.springframework',
  'spring-webmvc',
  '4.3.8.RELEASE'
) in
spring depends_on*
```

```
g.V()
  .hasLabel('library')
  .has('language', 'java')
  .has('group', 'org.springframework')
  .has('artifact', 'spring-webmvc')
  .has('version', '4.3.8.RELEASE')
.emit().repeat(out('depends_on').dedup())
```

Demo

- General features
- Struts
  - CVE-2018-11776, Apache Struts
  - URL payload, leads to RCE via OGNL execution
  - Source: ActionProxy#getMethod
  - Sink: OgnlUtil#compileAndExecute

Semantics

- Not Turing-complete
  - Programs always terminate
- No side effects
  - Every expression is referentially transparent
- Easier to rewrite and analyze

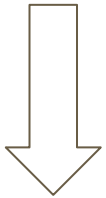
Optimizations

- Motivation
  - Keep SGL declarative
  - Free users from having to worry about query performance
- Reduction to relational algebra
  - (Inner) join: edge traversal
  - Project: where
  - Select: vertex predicates
  - Treat transitive closure as a view/intensional relation

Optimizations

- Join ordering (i.e. query planning)
  - Given n relations, n! possible orderings
  - Essential problems: query equivalence, cost
  - Enumerate equivalent queries, ordered by cost

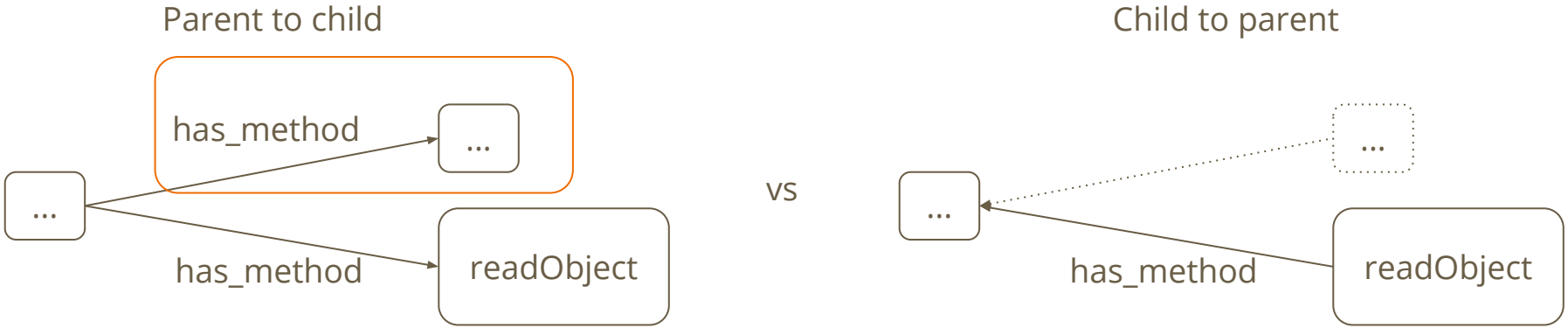
```
library(_) where(has_method method(name: ‘readObject’))
```



```
method(name: ‘readObject’) method_in_library
```

Optimizations

- Join ordering
  - Query cost
    - Observation: certain join orderings are *known* to be more efficient
    - e.g. many-to-one relations
    - Notion of redundancy: vertices traversed which don't contribute to result



Optimizations

- Join ordering
  - Query cost
    - Redundancy for many-to-one relations
    - For the others, statistics of large dataset
      - Maven Central
      - 79M vertices, 582M edges, 76GB
      - Product of cardinalities

Edge	Avg out-deg	Avg in-deg
depends_on	4.0	4.1
has_file	43.5	1.0
has_method	1508.2	8.9
calls	27.2	30.6
embeds	54.9	22.0
defines	14.4	1.8
has_library_hash	1.0	2.6
has_method_hash	4.9	18.6
has_library	16.4	1.9
has_vulnerable_method	1.8	2.1
has_version_range	2.9	1.2
has_class	217.0	11.1
extends	1.0	1.0

Optimizations

- Join ordering benchmarks
- GlassFish zero-day; refer to paper for details

```
let glassfish_class =
  class(regex 'org.glassfish.*') in
let read_object =
  method(method_name:'readObject') in
let get_path = method(
    class_name:'java/io/File',
    method_name:'getPath') in
glassfish_class defines
  read_object where(calls get_path)
```

Optimizations

- Join ordering benchmarks

	Query	Redundancy	Runtime
Original	`glassfish_class defines read_object where(calls get_path)`	391.2	105.8s
Reversed	`get_path called_by read_object where(defined_by glassfish_class)`	55.7	0.6s

Use cases

- Program analysis
- Vulnerability description
  - Structured alternative to CVEs

Describing vulnerabilities

- CVEs
  - Useful canonical identifiers for vulnerabilities
  - Not machine-readable
  - When applied to a real-world system:
    - Manual matching of CPEs with whatever is actually being used
    - Vulnerability described in unstructured text; manual check
    - False positives, inconsistency

Describing vulnerabilities

- Idea: represent vulnerabilities as SGL *queries*
  - Structured and can be processed by tools
  - Trivially check by executing
  - Generalize vulnerabilities by removing query predicates
  - Run regularly in CI, etc.
- Deduplication
  - Researchers often must check if a vulnerability that comes in is something they have dealt with before
  - Relies on query equivalence; difficult for arbitrary queries
  - Idea: define a subset that can be checked for equivalence

Constant queries

- Constant queries that can be compared, i.e. a data structure
- Subset of language features
  - Bindings
  - Vertex predicates
  - No edge steps
  - Must begin at vulnerability
  - Expand syntactic sugar
  - Sort
- Normalize
- Structural type

```
vulnerability(cwe: 1)
  has_version_range union(
    version_range(from: '1.0', to: '1.1'))
  union(
    has_library union(
      library('java', 'web', 'core', '1.0'),
      library('java', 'web', 'core', '1.1')),
    has_vulnerable_method union(
      method('com/example/Controller',
        'config', '()')))
```

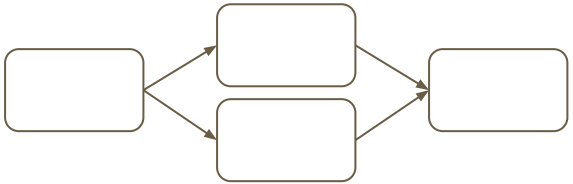
Reification

- We'd also like to use vulnerabilities in queries
  - "Find all vulnerable libraries"
- Reify vulnerabilities as vertices
- Link to other data, like libraries and vulnerable methods
- Distinguish by storing normalized query in a property

Future work

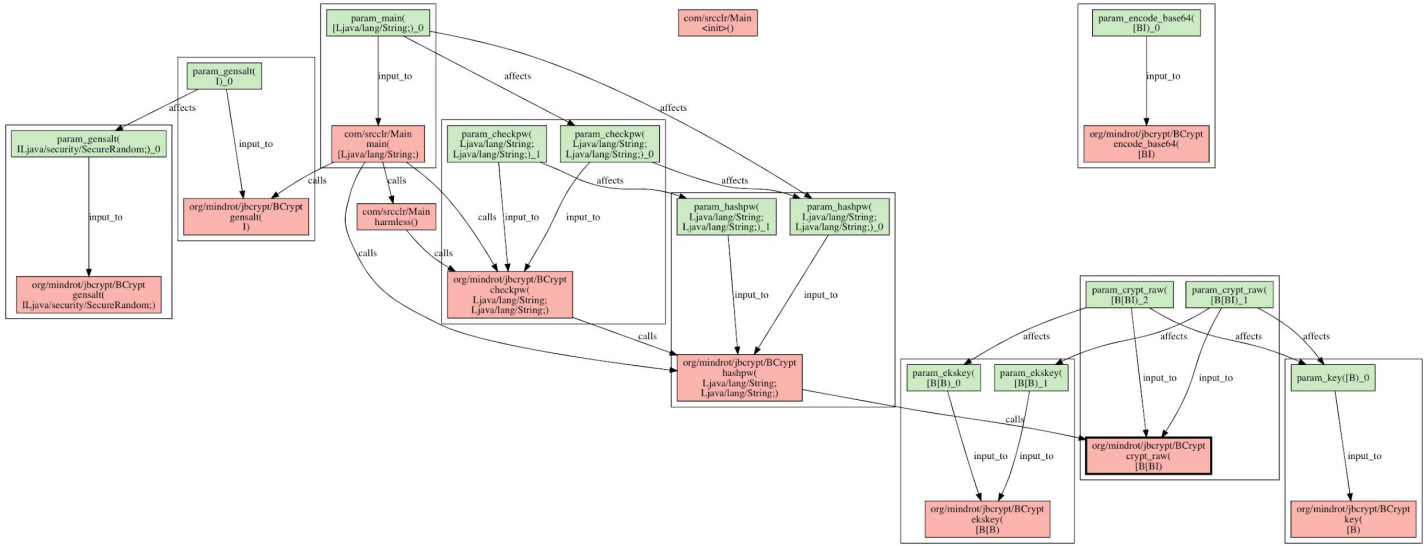
- Expressiveness
  - Datalog without user-defined rules
    - Computation?
  - Arbitrary “diamond” joins

```
library(…) ?a depends_on library(…) ?b,
?a has_method method(…) method_in_library ?b
```



Future work

- More domains
  - Dataflow graphs



Try it out

- www.sourceclear.com
- Free trial
- `SRCCLR_ENABLE_SGL=true srcclr scan --url https://github.com/srcclr/example-java-maven --sgl`

Thank you!

Q&A