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

- Graph query languages
  - Gremlin
  - Cypher

- Vulnerability description languages
  - OVAL
SGL: implementation

- Typed, declarative Gremlin subset
- Compiles to Gremlin
- Query planning
“Does this version of Apache Commons Collections contain a method named readObject?”
library(coord1: 'commons-collections', version: '3.2.2')

has_method method(name: 'readObject')
“What methods does this version of Apache Commons Collections contain?”
library(coord1: ‘commons-collections’, version: ‘3.2.2’)

has_method

calls

depends_on
SGL: results

library(coord1: 'commons-collections', version: '3.2.2')

has_method

method(name: 'readObject')
method(name: 'readExternal')
method(name: 'readResolve')
library(_) where(has_method method(name: ‘readObject’))
library(_) where(has_method method(name: 'readObject'))

library(version: '3.2.2')
library(version: '3.2.3')
library(version: '3.2.4')
library(coord1: 'commons-collections', version: '3.2.2')

depends_on

“What are the direct dependencies of Apache Commons Collections?”
library(coord1: 'commons-collections', version: '3.2.2')
depends_on*

“What are all the dependencies of Apache Commons Collections?”
library(coord1: 'commons-collections', version: '3.2.2')

depends_on* limit(5) aggregation

“What are 5 dependencies of Apache Commons Collections?”
library(coord1: 'commons-collections', version: '3.2.2')

depends_on* count

aggregation

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

![Diagram](image-url)

- Parent to child vs Child to parent
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

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

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

```plaintext
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

<table>
<thead>
<tr>
<th></th>
<th>Query</th>
<th>Redundancy</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>glassfish_class defines read_object where(calls get_path)</td>
<td>391.2</td>
<td>105.8s</td>
</tr>
<tr>
<td>Reversed</td>
<td>get_path called_by read_object where(defined_by glassfish_class)</td>
<td>55.7</td>
<td>0.6s</td>
</tr>
</tbody>
</table>
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

```plaintext
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

```prolog
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