Finding Vulnerabilities and Malware in Open-Source Code at Scale
Mark Curphey
Section One

The Rise of Open Source

Section Two

How Using Open Source Code Works

Section Three

What Can Go Wrong

Section Four

The Vulnerability Landscape

Section Five

The Security Graph Language
Section One

The Rise of Open Source
78% of companies rely on open-source
20% of US government code must be open-sourced
1.5 Billion

Year


Lines of Code

[SourceClear]
Section Two

How Does Using Open Source Code Work?
The Code Cocktail

Choose a Framework
The Code Cocktail

Write Custom Code
The Code Cocktail

Use Open-Source Libraries to Solve Common Problems
Open Source = ~ 90%

- Open-source code (~70%)
- Custom Code (~10%)
- Open-source code (~20%)
Open-Source Distribution
Direct and Indirect Dependencies
Number of Dependencies

<table>
<thead>
<tr>
<th></th>
<th>Python</th>
<th>Javascript</th>
<th>Java</th>
<th>Ruby</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>23</td>
<td>13</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td>430</td>
<td>38</td>
<td>59</td>
</tr>
<tr>
<td>1,000</td>
<td></td>
<td></td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

- Direct
- Transitive
Direct and Indirect Dependencies

Custom Code
Trend 1 - Peer to Peer Distribution
Trend 2 - Increasing Modularization of Code
Trend 3 - Always Using the Latest Code
Section Three

What Can Go Wrong
<table>
<thead>
<tr>
<th>Feature</th>
<th>Java</th>
<th>Javascript</th>
<th>Ruby</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verifies Code for Malware</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Verifies Code for Known Vulnerabilities</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Two Factor Authentication</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Requires Code Signing</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>
Web Application Ransomware

1. **Step One:** Publish malware to a registry

2. **Step Two:** Developer of CI server runs build

3. **Step Three:** Run code in production
Section Four

The Vulnerability Landscape
1st Generation

Map library names to public vulnerabilities
Map **BUILD OUTPUT** to public and **PRIVATE** vulnerabilities **AND** determine if vulnerable methods are being called.
Current State-of-the-Art

- Commit Logs
- Issue Trackers
- Mailing Lists
- Social
- Public Advisories
- Data Analysis, Machine Learning, Deep Learning, NLP...
- Secondary Analysis
- Private Vulnerabilities
- CVE
How big is the vulnerability pool?

- 2.5 LIBS: 50,000
- 100K LIBS: 150,000
- 1M LIBS: 250,000
Type of Issues

Inherited  Disclosed
Similar    Embedded
Reintroduced
Malware    Half-Days
Adware     Insecure Coding
Next Generation

Commit Logs | Issue Trackers | Mailing Lists | Social | Behaviors, Rules & Patterns | Public Advisories
---|---|---|---|---|---
Data Analysis, Machine Learning, Deep Learning, NLP... | Code Sequencing | Secondary Analysis | Public Advisories
Private Vulnerabilities
Section Five

The Security Graph Language
Ask Sophisticated Questions

Share Findings

Repeatable & Extendable
Global Dependency Graph
Global Call Graph
Global Class Hierarchy
Code Similarity (AST)
Binary Similarly
Method Info
Vulnerability Info

Security Graph Language
Gremlin

ETL

[:]SourceClear
35,000 libraries (nodes) and 66,000 edges

So far, we have 63+ million nodes, 420+ edges

Reference: https://exploringdata.github.io/vis/npm-packages-dependencies/
SGL Syntax Summary

vertex =
  'library'
  'library_hash'
  'method'
  'class'
  'vulnerability'
  'file'
  'method_hash'
  'version_range'
  'license';

edge =
  'depends_on'
  'embeds'
  'calls'
  'extends'
  'defines'
  'has_class'
  'has_method'
  'has_method_hash'
  'has_library'
  'has_file'
  'has_hash_file'
  'has_vulnerable_method'
  'has_method_hash'
  'has_library_hash'
  'has_file'
  'has_hash_file'
  'method_in_library'
  'method_in_file'
  'hash_for_library'
  'hash_in_file'
  'file_in_library'
  'file_in_file';

reverse_edge =
  'dependent_on'
  'embedded_in'
  'called_by'
  'extended_by'
  'defined_by'
  'in_library'
  'method_in_library'
  'method_in_file'
  'hash_for_library'
  'hash_in_file'
  'file_in_library'
  'file_in_file';

property =
  'language'
  'coord1'
  'coord2'
  'version'
  'subdirectory'
  'module_name'
  'class_name'
  'method_name'
  'descriptor'
  'at'
  'name'
  'cwe'
  'cvss'
  'query'
  'hash'
  'length'
  'from'
  'to'
  'type'
  'identity';

extra_step =
  'where'
  'not'
  'union'
  'state_put'
  'state_get'
  'limit'
  'count'
  'path'
  'project'
  'select'
  'order_by'
  'identity'
  'value_map'
  'id'
  'dedup';
1. Find all instances of an **Embedded** vulnerability
2. Find all instances of an **Inherited** vulnerability
3. Impact of a **Disclosed** vulnerability
4. Find all “hot spots” of dangerous coding
5. Find all instances of a **Similar** vulnerability
6. Understand the true impact of all **Disclosed & Half Day** vulnerabilities
// Version ranges of libraries that are affected by the sid-1847 Apache Commons Collections Deserialization vulnerability

vulnerability(identity: "1847") has_version_range

// Returns 7 libraries that are in the version range

// All library instances in those version ranges

vulnerability(identity: "1847") has_version_range has_library

// Returns 61 library versions that are affected by this vulnerability

// All embedded libraries that are affected by the vulnerability

vulnerability(identity: "1847") has_version_range has_library embedded_in

// Returns 11 libraries that are instances of an embedded vulnerability
// All directly dependent libraries that are affected by the vulnerability
vulnerability(identity: "1847") has_version_range has_library dependent_on
// Returns 843 directly dependent affected libraries

// All directly and indirectly dependent libraries that are affected by the vulnerability
vulnerability(identity: "1847") has_version_range has_library dependent_on*
// Returns 10359 directly and indirectly dependent affected libraries

// Combine direct, embedded and indirectly affected libraries using union
vulnerability(identity: "1847") has_version_range has_library union(dependent_on*, embedded_in*)
// Returns 10370 affected libraries captures the true impact of a disclosed vulnerability
// Vulnerable methods

vulnerability(identity: "1847") has_version_range has_vulnerable_method

// Returns 4 vulnerable methods

// All the callers of the vulnerable methods

vulnerability(identity: "1847") has_version_range has_vulnerable_method called_by*

// Returns 341 callers of the vulnerable methods

// All the libraries that have those callers

vulnerability(identity: "1847") has_version_range has_vulnerable_method called_by* method_in_library

// Returns 6033 libraries that call these methods
vulnerability(identity: "1847") has_version_range has_vulnerable_method called_by* method_in_library
where(library_in_version_range version_range_in_vulnerability vulnerability(identity: "1847"))

// Returns 23 vulnerable libraries that call these methods

vulnerability(identity: "1847") has_version_range has_vulnerable_method called_by* method_in_library
where(union(embeds*, depends_on*) library_in_version_range version_range_in_vulnerability vulnerability(identity: "1847"))

// Returns 184 affected (including indirect and embedded) libraries that call these methods in their libraries, these represent instances of an inherited vulnerability
// All libraries that call Runtime.exec()
method(class_name: 'java/lang/Runtime', method_name: 'exec') called_by method_in_library

// Returns 10773 libraries

// All libraries that forcefully collect garbage
method(class_name: 'java/lang/Runtime', method_name: 'gc') called_by method_in_library

// Returns 2659 libraries

// All libraries that connect to a database
method(class_name: 'java/sql/Connection') called_by method_in_library limit 100

// Returns 445566 libraries, takes a while to return so we limit to 100

[:] SourceClear
// Prevent XXE in Java

DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
dbf.setFeature("http://xml.org/sax/features/external-general-entities", false);

// SGL query for XXE pattern

let xml_new = method(class_name:'javax/xml/parsers/DocumentBuilderFactory', method_name:'newInstance') in
let xml_set_feature = method(class_name:'javax/xml/parsers/DocumentBuilderFactory', method_name:'setFeature') in
let results = xml_new called_by not(calls xml_set_feature) in
results method_in_library

// 503 libraries with potential XXE
// Impact of all disclosed and half-day vulnerabilities

vulnerability(_) has_version_range has_library union(identity, embedded_in*, dependent_on*)

// Before 1973 known vulnerable libraries
// After SGL >6548 known vulnerable libraries
That’s all Folks!
Conclusion

Questions?