POLYCRUISE: A Cross-Language Dynamic Information Flow Analysis

Wen Li*, Jiang Ming†, Xiapu Luo‡, Haipeng Cai*

*Washington State University
†University of Texas at Arlington
‡ The Hong Kong Polytechnic University
● Interfacing mechanisms between languages
  ◦ Uniform mechanism
    - inter-process communication (IPC)
      e.g., Remote Procedure Call (RPC) on socket, shared memory
  ◦ Language-specific mechanism
    - foreign function interface (FFI)
      e.g., JNI for Java-C, Python extension for Python-C

● Cross-language DIFA
  ◦ DIFA cross language boundaries
● Challenges in DIFA for multi-language program (MLP)
  ◦ Semantics disparity
    - Existing DIFAs → stopped at language boundaries
      – Stitching single-language DIFAs → not applicable
  ◦ Analysis cost-effectiveness
    - No instrumentation guidance for MLP
    - More complicated than SLP

● POLYCRUISE’s targets
  - unified instrumentation guidance
  - scalable DIFA
  - online bug detection
Example 1:
Sensitive data leaks on bidirectional invocations cross language units

Example 2:
Sensitive data leaks on implicit invocations cross language units
• POLYCRUISE Workflow
• **Static analysis & Instrumentation**

  ◦ Efficiency? Only instrument necessary points (slicing)
  ◦ How to obtain *unified instrumentation* guidance for different language units?

→ **Traditional data flow analysis? ×**
  - Single-language feasibility: stop at language boundaries
  - Heavy: memory usage & time cost
  - Consistency issue

→ **Symbolic Dependence Analysis (SDA) √**
  - Light weight & extensibility to new languages
  - **Steps:** LISR translation → SDA on LISR → Instrumentation guidance
Symbolic Dependence Analysis (SDA)

1. Source Code
2. typeA gValue
3. Output(typeB& arg)
4. print (arg)
5. typeB Foo(typeB N)
6. V := I
7. typeB S := V
8. V := N
9. while N != 0:
10. V := V ∗ N
11. N := N − 1
12. Output (S)
13. return S
14. LISR
15. gValue
16. Output (arg)
17. print (arg)
18. Symbolic def-use pairs

<1> Source → (S9, V)
<2> forward(true flow dependencies) → D(S9)∩U(S11) ≠ ∅
<3> backward (anti-dependencies) → U(S8)∩D(S9) ≠ ∅

Hence, the symbolic dependence set of S9 → {S8, S11}.
- Online dynamic analysis
  - Language-agnostic
  - Accumulated

- Dynamic information flow graph (DIFG)

```plaintext
1 T g
g = v
2 set (v) { g = v }
3 T1 () { set (1) }
4 T2 () { put (g) }
5 Entry () {
  6 Thread (T1)
  7 Thread (T2)
  8 Join ()
  9 }
```

- Interthread control flow edge
- Intra-thread control flow edge
- Interthread data flow edge
- Intra-thread data flow edge
• Bug detection

**bug detection plug-ins**

buffer-overflow  div-by-zero  Incomplete comparison  .....
● Three evaluation metrics
  - **Effectiveness**
    → PyCBench: 46 micro benchmarks for Python-C
  - **Efficiency**
    → Efficiency of SDA on 12 real-world Python-C programs
    → Run-time slowdown and memory usage on 12 real-world Python-C programs
  - **Capacity of bug discovery on real-world programs**

● **Environment**
  Ubuntu 18.04 workstation with an Intel i7-10875H CPU and 16GB RAM
### Effectiveness results of POLYCRUISE on PyCBench

<table>
<thead>
<tr>
<th>Group</th>
<th>#Inter-language path</th>
<th>#intra-language path</th>
<th>#false-negative</th>
<th>#false-positive</th>
</tr>
</thead>
<tbody>
<tr>
<td>General flow</td>
<td>10</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Global flow</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Field sensitivity</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Object sensitivity</td>
<td>9</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Dynamic invocation</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Summary</td>
<td>40</td>
<td>6</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

POLYCRUISE achieved 93.5% precision and 100% recall on PyCBench
**SDA on 12 real-world Python-C programs**

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Size (KLoC)</th>
<th>Time (second)</th>
<th>Memory (MB)</th>
<th>Instruction rate%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bounter</td>
<td>3.5</td>
<td>0.02</td>
<td>2.97</td>
<td>52%</td>
</tr>
<tr>
<td>Immutables</td>
<td>5.9</td>
<td>0.04</td>
<td>4.68</td>
<td>50%</td>
</tr>
<tr>
<td>Simplejson</td>
<td>6.4</td>
<td>0.03</td>
<td>4.47</td>
<td>56%</td>
</tr>
<tr>
<td>Japronto</td>
<td>9.4</td>
<td>0.02</td>
<td>3.89</td>
<td>47%</td>
</tr>
<tr>
<td>Pygit2</td>
<td>17.0</td>
<td>0.13</td>
<td>14.54</td>
<td>43%</td>
</tr>
<tr>
<td>Psycopg2</td>
<td>27.5</td>
<td>0.14</td>
<td>15.32</td>
<td>57%</td>
</tr>
<tr>
<td>Cvxopt</td>
<td>56.0</td>
<td>1.21</td>
<td>35.52</td>
<td>52%</td>
</tr>
<tr>
<td>Pygame</td>
<td>207.0</td>
<td>2.27</td>
<td>85.32</td>
<td>44%</td>
</tr>
<tr>
<td>PyTables</td>
<td>219.8</td>
<td>2.45</td>
<td>101.11</td>
<td>51%</td>
</tr>
<tr>
<td>Pyo</td>
<td>259.1</td>
<td>20.21</td>
<td>258.73</td>
<td>62%</td>
</tr>
<tr>
<td>NumPy</td>
<td>919.7</td>
<td>10.99</td>
<td>557.95</td>
<td>48%</td>
</tr>
<tr>
<td>PyTorch</td>
<td>6,419.2</td>
<td>175.19</td>
<td>7,414.95</td>
<td>51%</td>
</tr>
</tbody>
</table>
- Run-time slowdown and memory usage

Compared to whole-system instrumentation version:

→ **Slowdown**: the SDA improved the reduction of slowdown factor from 18.3% (in Japronto) to 66.2% (in PyTorch)

→ **Peak memory**: the SDA reduced the memory usage by 16.2% (in Japronto) to 67.1% (in Cvxopt)
## Bug Discovery by POLYCRUISE

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>#Integer-overflow</th>
<th>#Buffer-overflow</th>
<th>#Incomplete-comparison</th>
<th>#CVE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bounter</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Immutables</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Japronto</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cvxopt</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Pyo</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Numpy</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td><strong>1</strong></td>
<td><strong>8</strong></td>
<td><strong>5</strong></td>
<td><strong>8</strong></td>
</tr>
</tbody>
</table>
● Extensibility to support other languages
  - LISR translator
  - Instrumentor

● Limitations
  - Field-insensitive implementation
  - Failed to cover implicit data flows
  - Capability of bug discovery limited by test inputs
  - Support language interfacing: FFI
• POLYCRUISE, a novel dynamic information flow analysis (DIFA) for multilingual systems.
  - SDA-guided instrumentation
  - Online DIFA
  - Bug detection plug-ins

• 14 bugs on 6 real-world Python-C programs, 8 CVEs assigned
Thanks for Your Attention

Q & A

POLYCRUISE: A Cross-Language Dynamic Information Flow Analysis

Wen Li
Washington State University, Pullman
li.wen@wsu.edu

Xiapu Luo
The Hong Kong Polytechnic University
csxluo@comp.polyu.edu.hk

Ming Jiang
University of Texas at Arlington
jiang.ming@uta.edu

Haipeng Cai
Washington State University, Pullman
haipeng.cai@wsu.edu

Presenter: Wen Li
Email: li.wen@wsu.edu

Code, data, PoCs: https://github.com/Daybreak2019/PolyCruise