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

usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

31ST USENIX SECURITY SYMPOSIUM

- **Interfacing mechanisms between languages**
  - Uniform mechanism
    - inter-process communication (IPC)

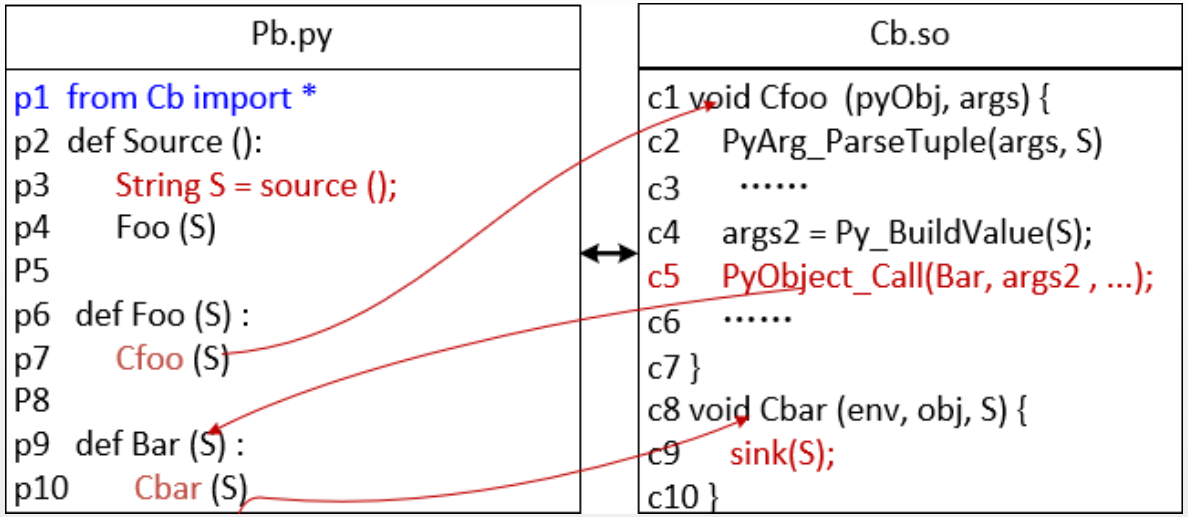
      e.g., Remote Procedure Call (RPC) on socket, shared memory
  - Language-specific mechanism
    - foreign function interface (FFI)

      e.g., JNI for Java-C, Python extension for Python-C

- **Cross-language DIFA**
  - DIFA cross language boundaries

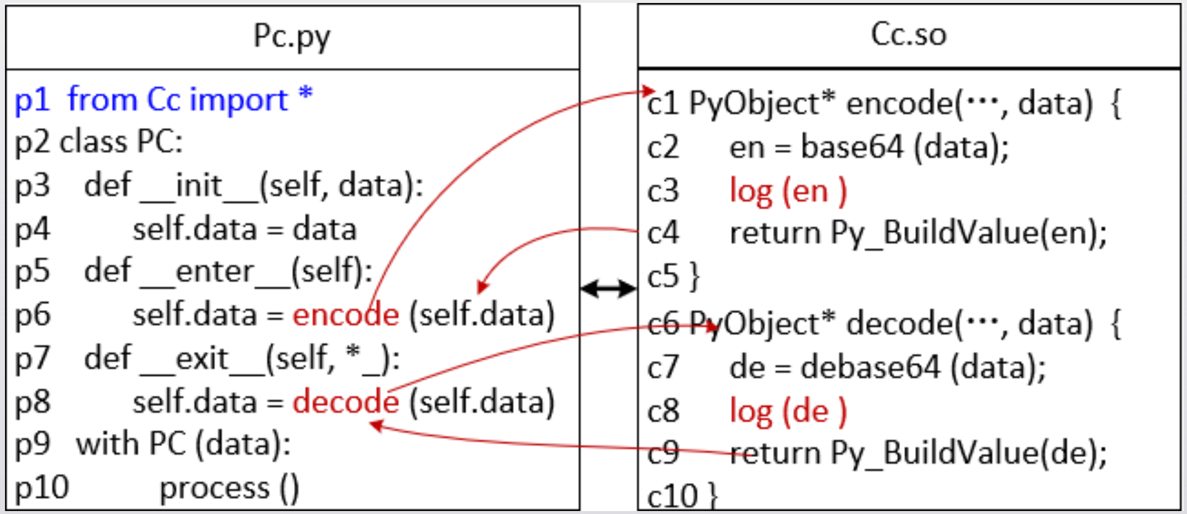
- **Challenges in DIFA for multi-language program (MLP)**
  - Semantics disparity
    - Existing DIFAs → stopped at language boundaries
    - Stitching single-language DIFAs → not applicable
  - Analysis cost-effectiveness
    - No instrumentation guidance for MLP
    - More complicated than SLP

- **POLYCRUISE's targets**
  - unified instrumentation guidance
  - scalable DIFA
  - online bug detection



Example 1:

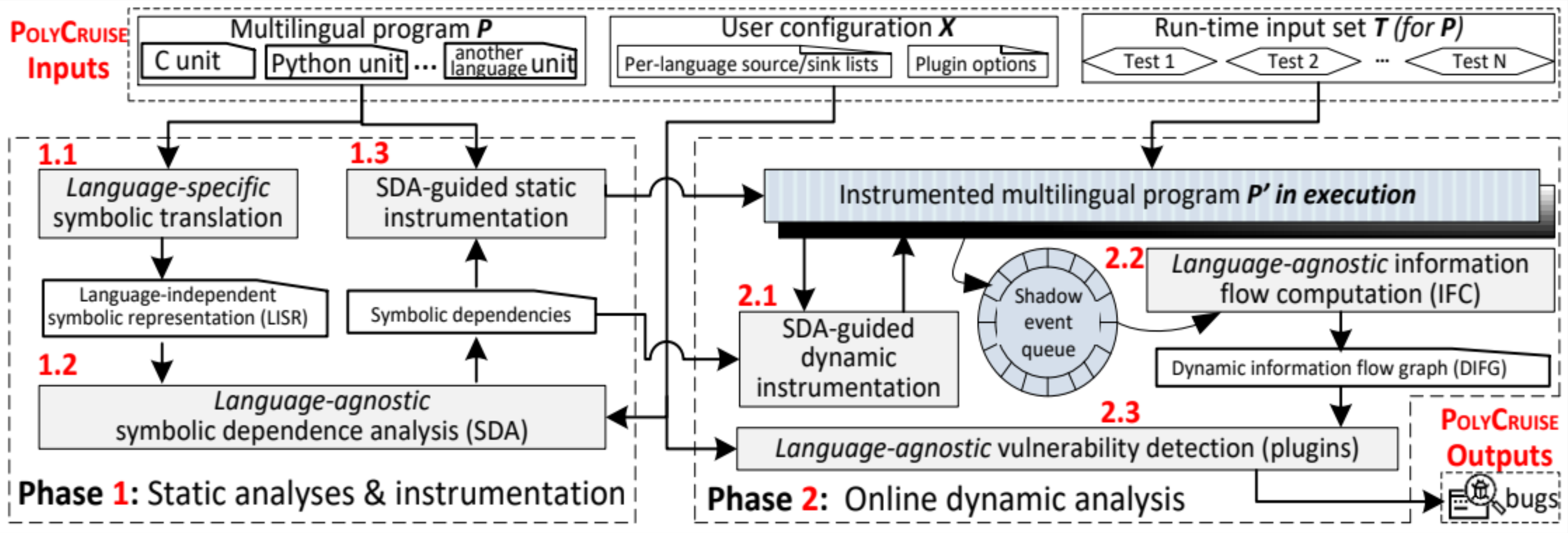
Sensitive data leaks on bidirectional invocations cross language units



Example 2:

Sensitive data leaks on implicit invocations cross language units

- **POLYCRUISE Workflow**



static analysis & instrumentation → online dynamic analysis → bug detection on DIFA

- **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)

#	Source Code	LISR	symbolic def−use pairs
2	typeA gValue	gValue	
3	Output(typeB& arg)	Output(arg)	
4	print (arg)	print(arg)	D[4]={},U[4]={arg}
5			
6	typeB Foo(typeB N)	T Foo(N)	
7	typeB V := 1	V = C	D[7] ={V},U[7] ={C}
8	typeB& S := V	S = V	D[8] ={S},U[8] ={V}
9	V := N	V = N	D[9] ={V},U[9] ={N}
10	while N != 0:	N	D[10]={ },U[10]={N}
11	V := V ∗ N	V = V,N	D[11]={V},U[11]={V,N}
12	N := N − 1	N = N,C	D[12]={N},U[12]={N,C}
13	Output (S)	Output(S)	D[13]={},U[13]={S}
14	return S	return S	D[14]={},U[14]={S}

<1> Source → (S9, V)

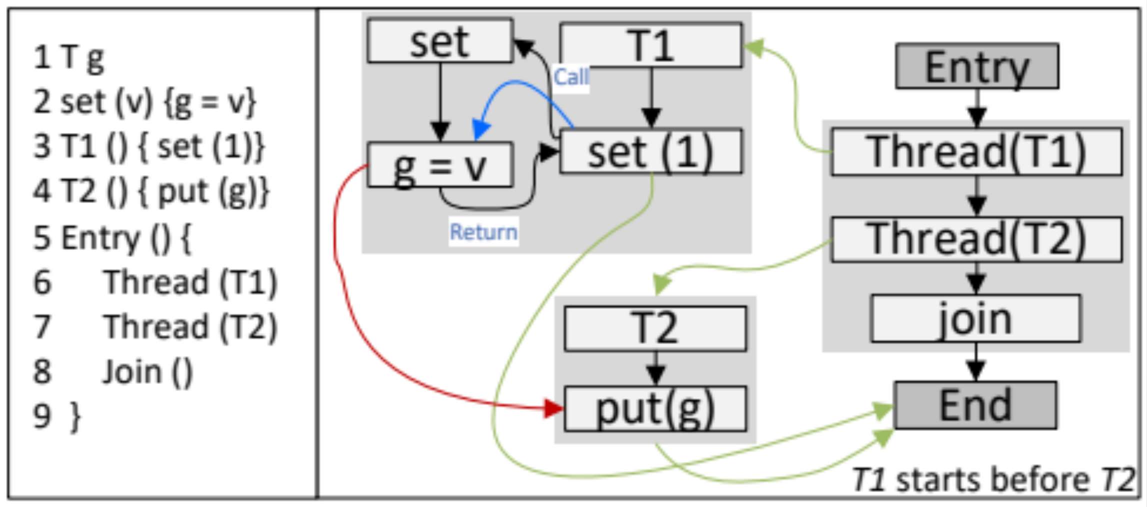
<2> forward(true flow dependencies) → D(S9)∩U(S11) ≠ ∅

<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)**

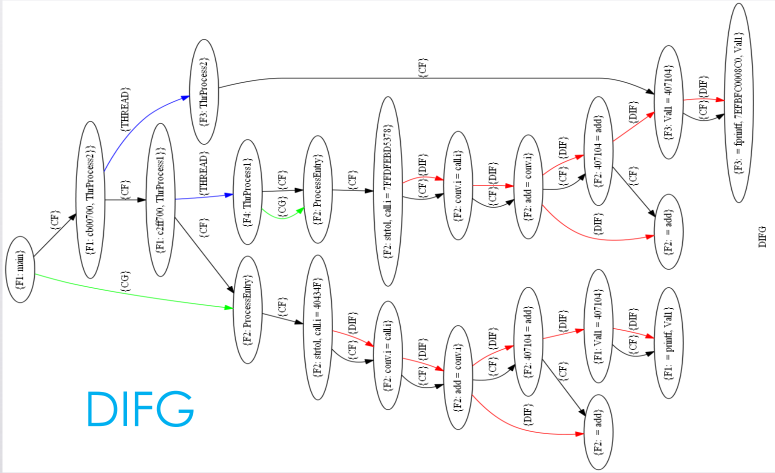


- Interthread control flow edge
- Intra-thread control flow edge
- Interthread data flow edge
- Intra-thread data flow edge

- **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

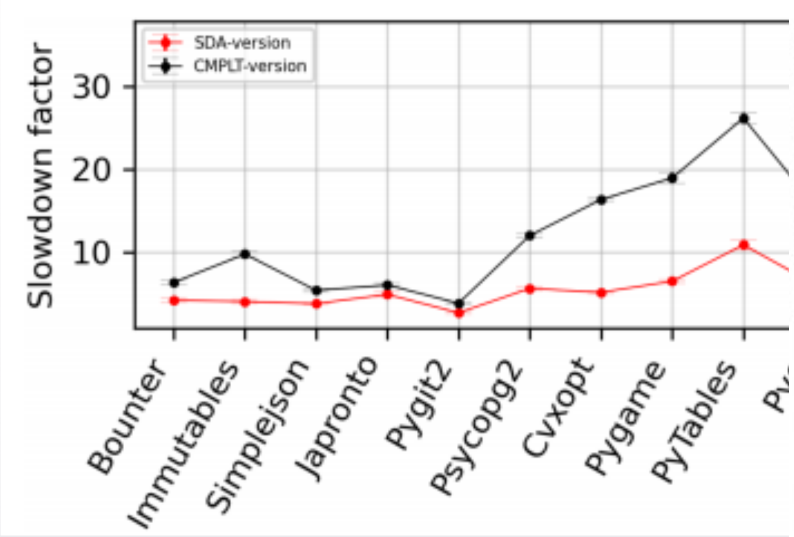
Group	#Inter-language path	#intra-language path	#fasle-negative	#false-positive
General flow	10	4	0	0
Global flow	9	0	0	0
Field sensitivity	8	0	0	2
Object sensitivity	9	2	0	1
Dynamic invocation	4	0	0	0
Summary	40	6	0	3

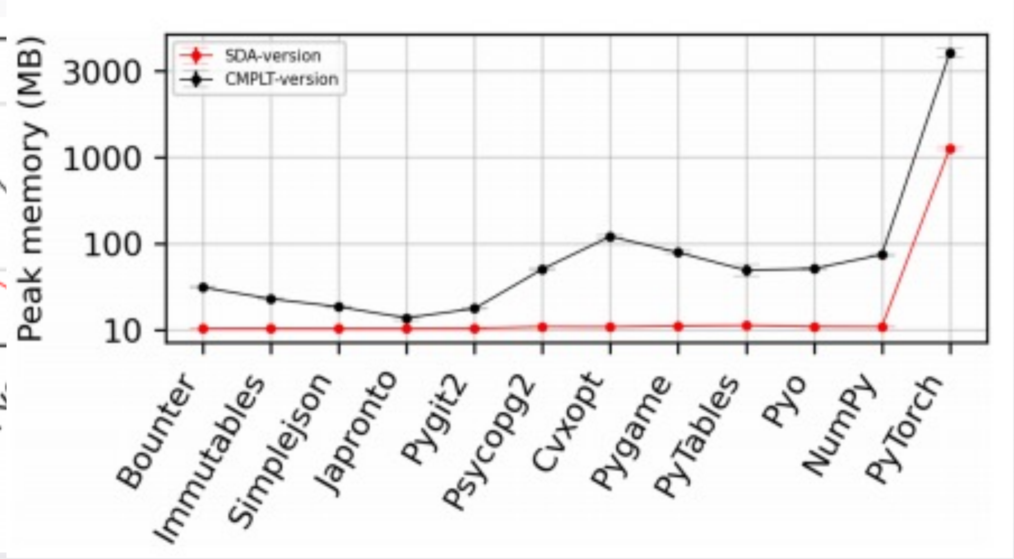
POLYCRUISE achieved 93.5% precision and 100% recall on PyCBench

- **SDA on 12 real-world Python-C programs**

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

● 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**

Benchmark	#Integer-overflow	#Buffer-overflow	#Incomplete-comparison	#CVE
Bounter	0	1	0	**1**
Immutables	0	1	0	0
Japronto	0	1	0	0
Cvxopt	0	0	4	**1**
Pyo	0	2	0	**2**
Numpy	1	3	1	**4**
Summary	**1**	**8**	**5**	**8**

- **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

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

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

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