POLYFUZZ: Holistic Greybox Fuzzing of Multi-Language Systems

Wen Li[*], Jinyang Ruan[*], Guangbei Yi[*], Long Cheng[+], Xiapu Luo[×], Haipeng Cai[*]

[*]Washington State University
[+]Clemson University
[×]The Hong Kong Polytechnic University

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

32ND USENIX SECURITY SYMPOSIUM

- **What is a multi-language (polyglot) system**

Interactive language components

Flexibility

Different features

- **Multi-language software is prevalent and impactful**

Over **75%** are programmed with multiple programming languages

GitHub

Android

Java/c

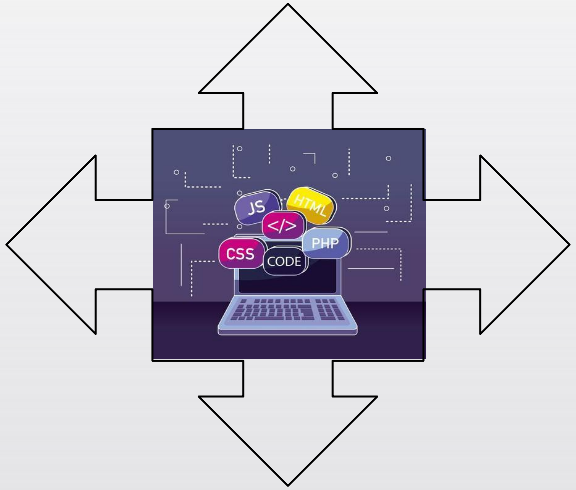
OS

Python/c

PyTorch

TensorFlow

Machine learning framework



JavaScript, Python, .NET

Azure

Cloud/distributed computing

Apache ZooKeeper

Java/c

Scientific

MATLAB

Java/c

- **Security of multi-language systems is critical**

Security risks are consequential in multi-language systems!

→ Threats in single-language systems also exist in multi-language systems

→ Threats in multi-language systems go deeper due to the greater complexity

PolyCruise@USENIX Security22, Cross-language dynamic information flow analysis

CVE-2021-33430, CVE-2021-41495, CVE-2021-41496,
CVE-2021-34141, CVE-2021-41497, CVE-2021-41500,
CVE-2021-41498, CVE-2021-41499

- **An example of risk of buffer-overflow cross Python-C code**

- - - - ▸ Data flow

- **Vulnerability detection: fuzzing is powerful and effective**

Program analysis

- Static analysis → *High false positives*
- Dynamic analysis → *Limited inputs* → *For functionality testing*

Fuzzing techniques

→ *Automated test generation*

→ *More chances to discover vulnerabilities*

→ *16K+ vulnerabilities detected by Fuzzers in various projects*

- **Existing fuzzing techniques are insufficient for multi-language systems**

► Existing fuzzing techniques primarily target single-language software

→ e.g., AFL/LibFuzzer for C program

► Limitations when fuzzing multi-language systems

→ Feasibility for different languages

→ Inefficiency due to incomplete feedback

→ Reproducibility of vulnerabilities

► Limitations on efficiency

→ 95%↑ mutations would be redundant!

• Challenges and design of PolyFuzz

Two primary challenges:

→ Challenge-1:

How to generate inputs that effectively exercise information flow across heterogeneous language units?

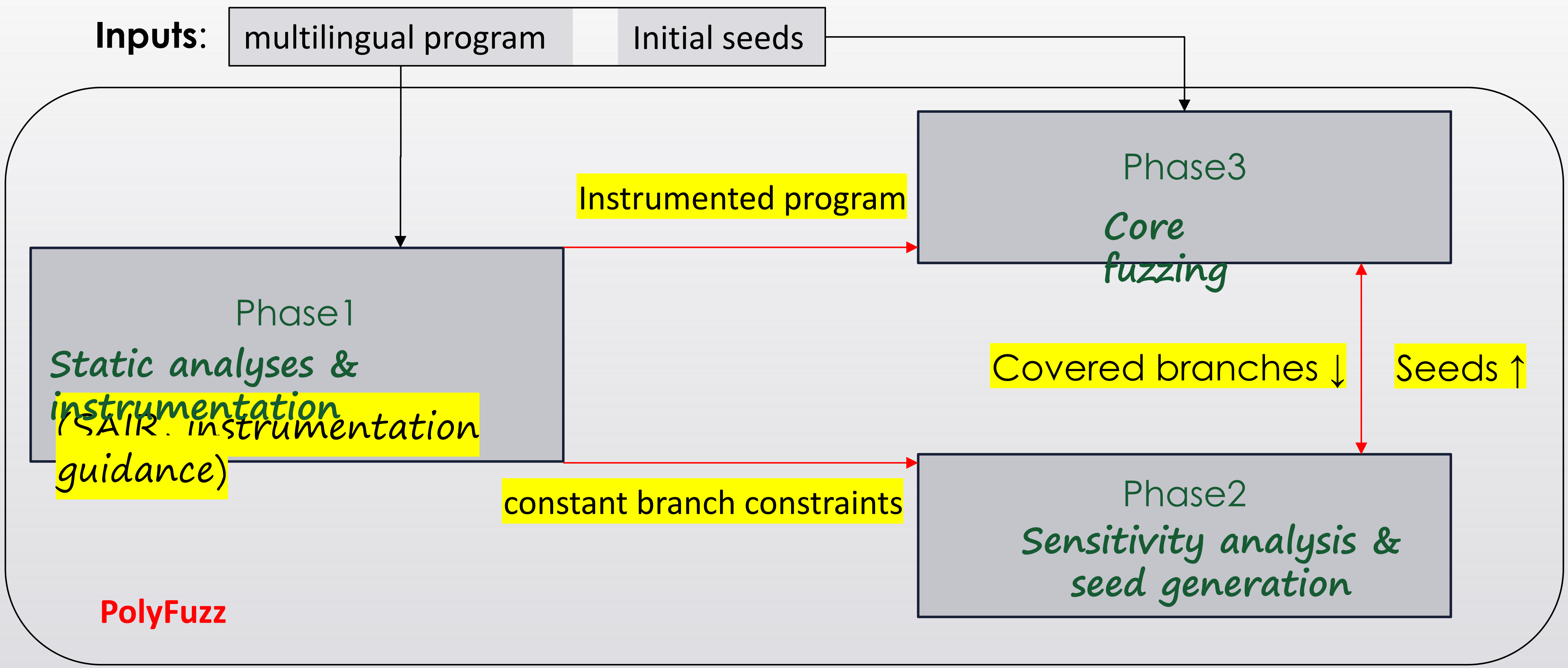
➡ Incorporate **sensitivity analysis** to guide seed generation

→ Challenge-2:

How to *achieve comprehensive* coverage *while accommodating* language extensibility?

➡ Run all heavy program analysis on a *custom* IR (**SAIR**) to minimize *language-specific analysis*

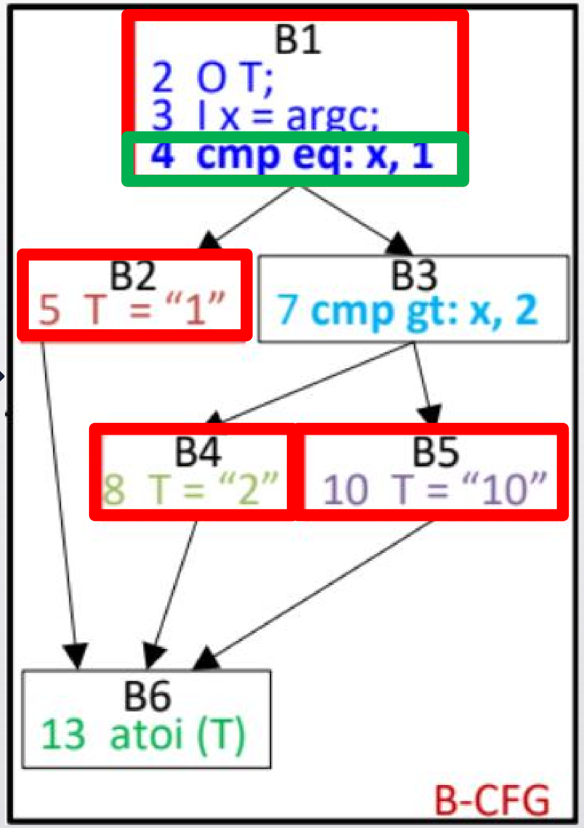
- **Overview of PolyFuzz**



- **Example of Phase1:** SAIR and instrumentation guidance

```
1 int func(int argc){
2     char *T;
3     int x = argc;
4     if (x == 1){
5         T = "1";
6     } else {
7         if (x > 2) {
8             T = "2";
9         else {
10            T = "0";
11        }
12    }
13    return atoi(T);
14 }
```
source

```
1 I func (I arg){
2     O T;
3     I x = argc;
4     cmp eq: x, 1
5     { T = "1" }
7     cmp gt: x, 2
8     { T = "2" }
10    { T = "0" }
13    atoi (T)
```
SAIR



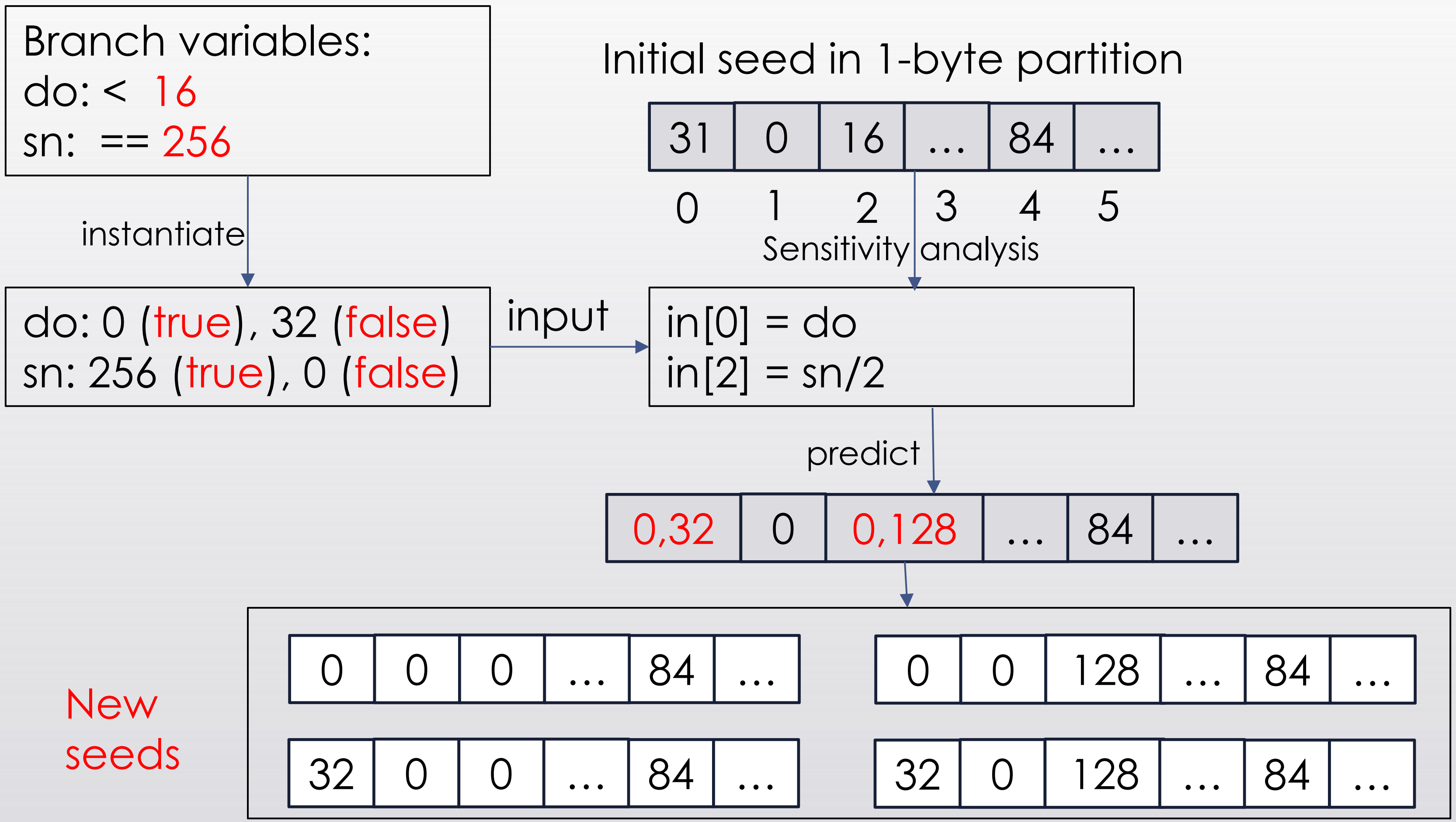
1. Minimized block-coverage (distinguish all execution paths):
→ [B1, B2, B4, B5]

2. Branch variable coverage:
→ [B1@s4]

3. Merged guidance:
→ [B1@s4,B2,B4,B5]

- **Example of Phase2:** the procedure of seed generation

```
void demo(byte in[16]){
    int do = in [0]
    instrument (do)
    if (do < 16) {
        do_onething (in)
    }
    else {
        int sn = in[2]*2
        instrument (sign)
        if (sn == 256) {
            do_other (in)
}}}
```



- **Regarding the effectiveness (#block, #bug)**

Baselines: Jazzer (Java), Atheris (Python), Honggfuzz (C)

Multi-language benchmarks

Benchmark	Jazzer	Jazz-C-ext	Atheris	Atheris-C-ext	PolyFuzz
10 Python-C (508.1 KLoC)	–	–	(1278, 1)	(5357, 3)	(1946/7319, 11)
5 Java-C (230.5 KLoC)	(1030, 0)	(1577, 0)	–	–	(1330/1976, 1)
Summary	↑(29.1%, 1)	↑(25.3%, 1)	↑(52.3%, 10)	↑(36.7%, 8)	–

Single-language benchmarks

Benchmark	Jazzer	Atheris	Honggfuzz	PolyFuzz
5 Java (332.3 KLoC)	(12319,1)	–	–	(13675, 1)
5 Python (545.7 KLoC)	–	(3964, 1)	–	(4782, 1)
5 C (1353.5 KLoC)	–	–	(6430, 0)	(7081, 0)
Summary	↑(11.0%, 0)	↑(20.1%, 0)	↑(10.1%, 0)	–

- **Regarding the Vulnerabilities Discovered**

Benchmark	#Bug	Symptom	#CVE
Libsmbios	1	Segment fault	**0**
Pillow	1	out of memory	1
Ultrajson	1	segment fault	1
Aubio	1	memory leak	**0**
Bottleneck	7	segment fault	**1**
Jansi	1	out of memory	**1**
Pyyaml	1	recursion error	**0**
Javaparser	1	JVM hung	**1**
Summary	**14**	–	**5**

CVE ID
CVE-2022-34070
CVE-2022-34072
CVE-2022-34073
CVE-2022-34074
CVE-2022-34075

► **PolyFuzz, a novel framework for holistic greybox fuzzing of multi-language software**

→ Measurement of whole-system block coverage

→ Effective seed generation via sensitivity analysis

→ Language extensible

Thanks for Your Attention

Q & A

POLYFUZZ: Holistic Greybox Fuzzing of Multi-Language Systems

Wen Li[1], Jinyang Ruan[1], Guangbei Yi[1], Long Cheng[2], Xiapu Luo[3], and Haipeng Cai[1*]

[1]*Washington State University* [2]*Clemson University* [3]*The Hong Kong Polytechnic University*

{li.wen, jinyang.ruan, guangbei.yi, haipeng.cai}@wsu.edu
lcheng2@clemson.edu csxluo@comp.polyu.edu.hk

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

Code, Data, PoCs: https://github.com/Daybreak2019/PolyFuzz