THE CHIPS TO SYSTEMS CONFERENCE

SHAPING THE NEXT GENERATION OF ELECTRONICS

JUNE 23-27, 2024

MOSCONE WEST CENTER
SAN FRANCISCO, CA, USA

南方科技大学

SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

COMPASS Lab

COMPuter And System Security Lab

Tatoo: A Flexible Hardware Platform for Binary-Only Fuzzing

Jinting Wu, Haodong Zheng, Yu Wang, Tai Yue, Fengwei Zhang

Southern University of Science and Technology, China

Backgroud

- Fuzzing is one of the most effective vulnerability discovery techniques.
  - ClusterFuzz has found around 27,000 bugs in Google.
- Code coverage is an effective way to collect feedback.
- Collecting coverage is challenging in binary-only fuzzing. Traditional methods include:
  - Software Instrumentation
    - Dynamic Binary Translation
    - Static Binary Rewriting
  - Hardware Tracing

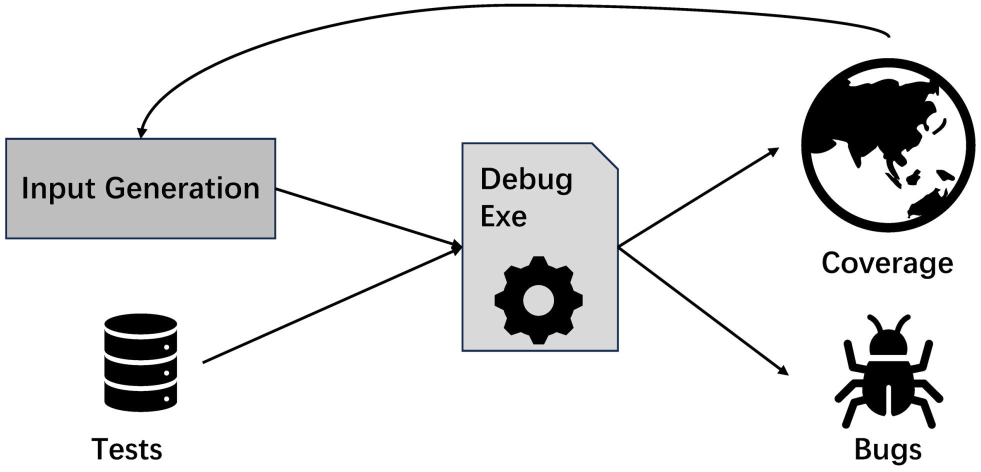


Figure 1: Coverage-guided greybox fuzzing process

Backgroud

- Existing information flow tracking technology:

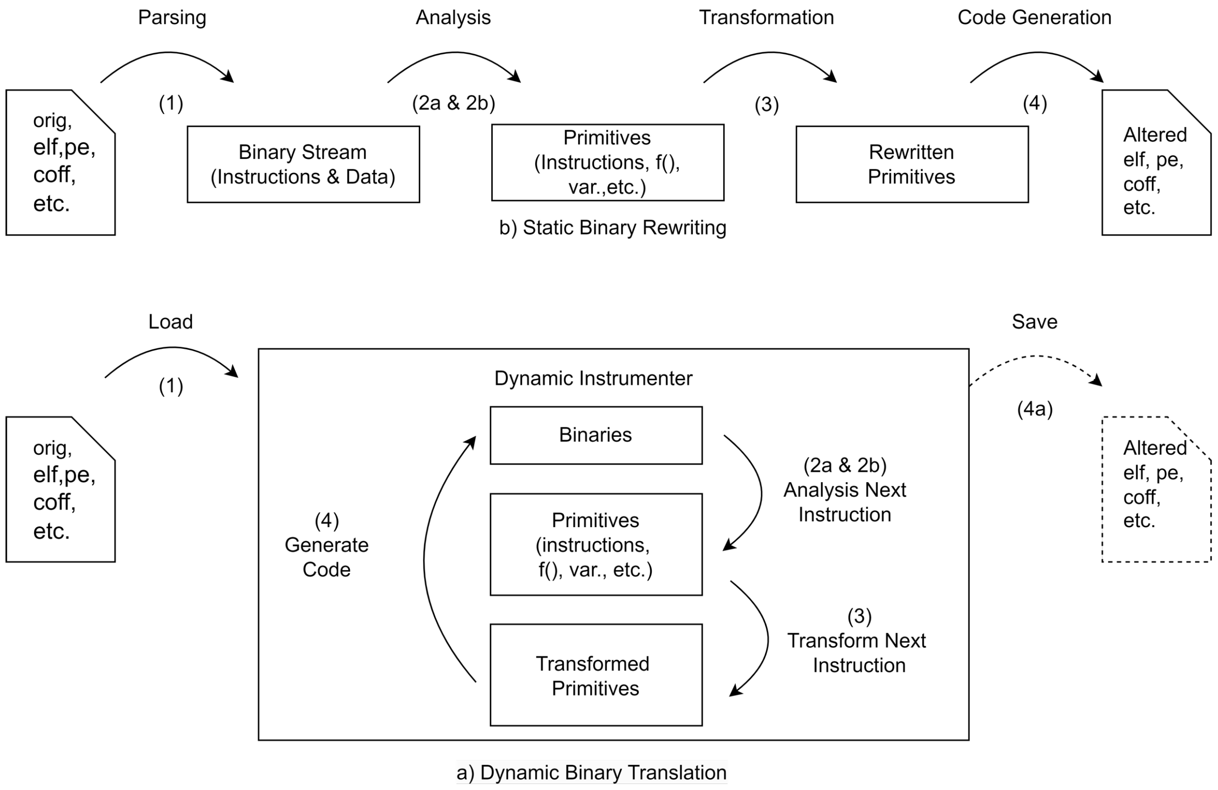


Figure 2: Software-based techniques

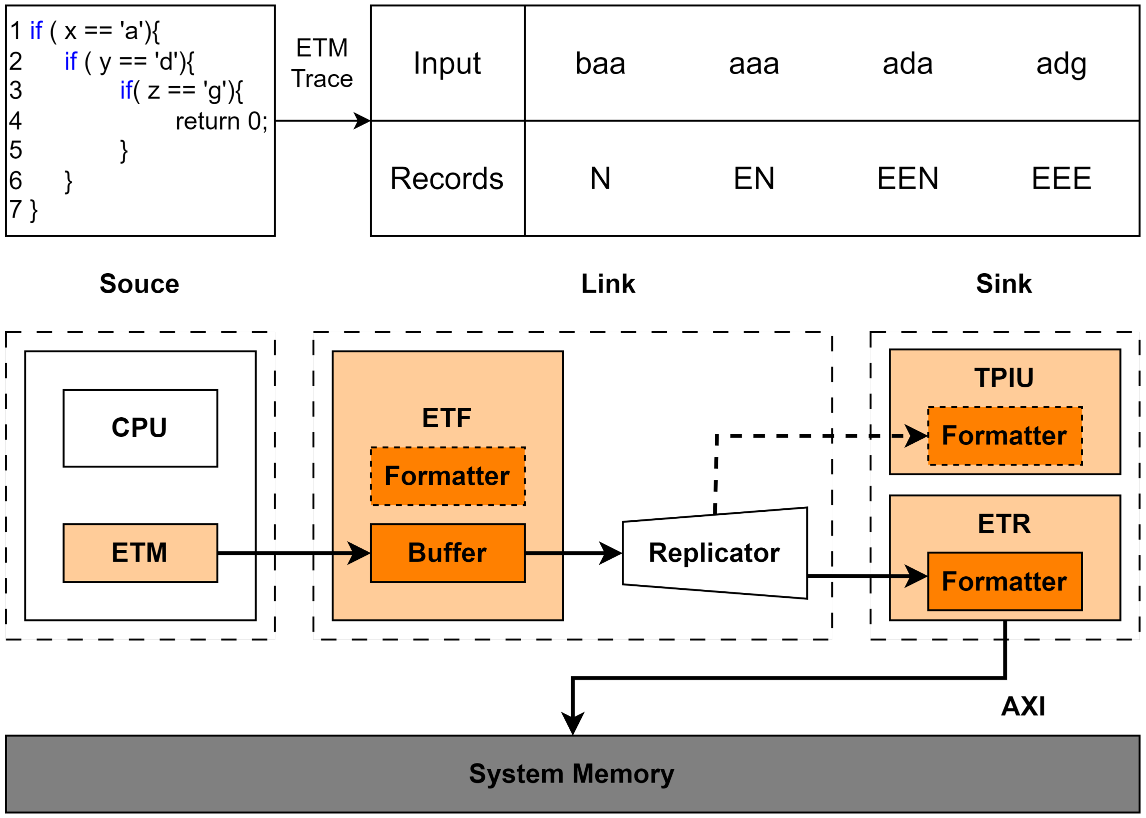


Figure 3: The overview of ARM ETM

Motivation

- The shortcomings of traditional hardware (e.g., ARM Coresight).

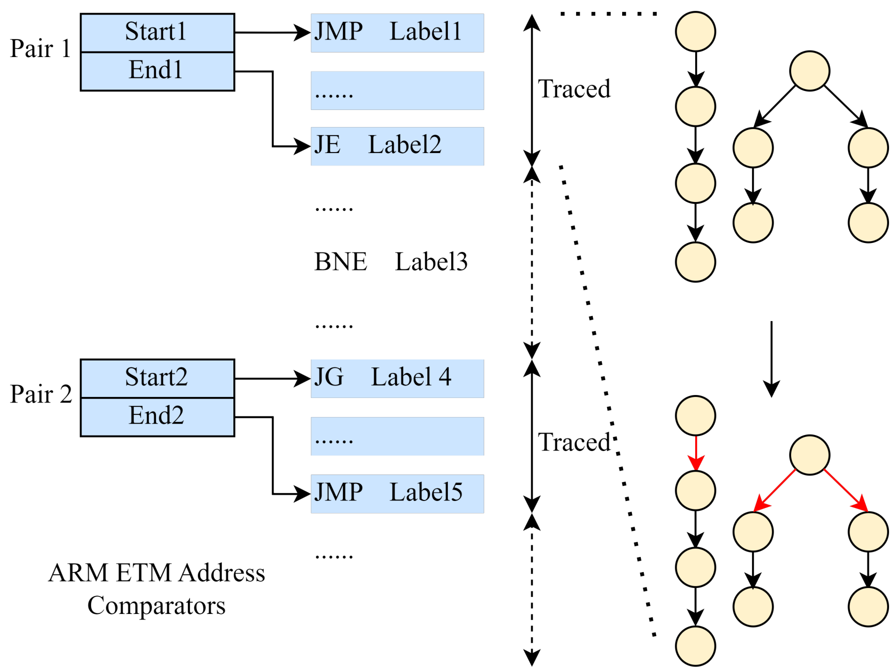


Figure 4: Trace irrelavant data

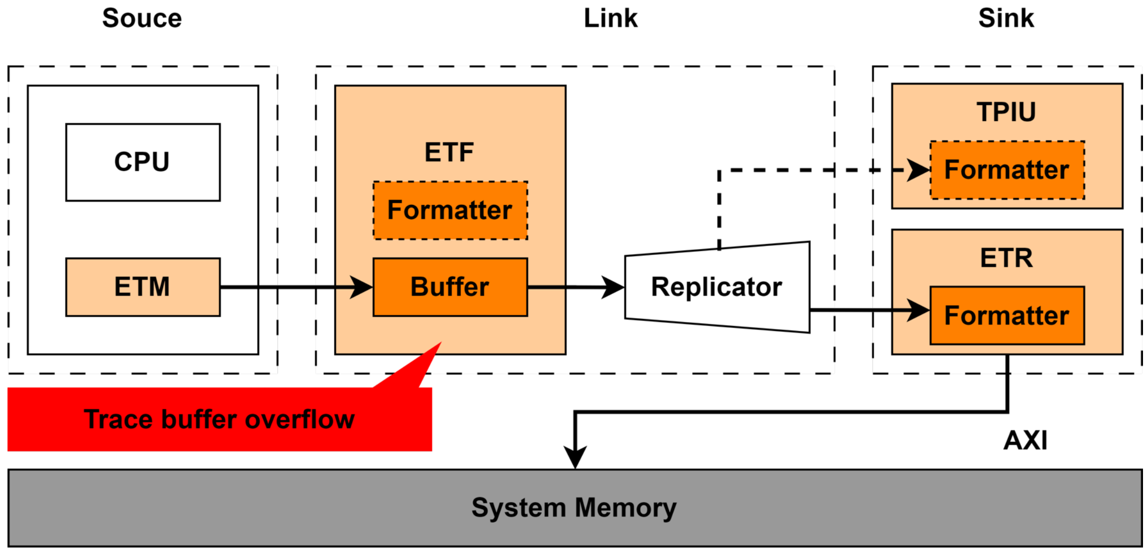


Figure 5: Trace dataflow may overwhelm trace buffer

- Hardware tracing tool should trace the necessary basic block and dataflow for fuzzing.

Main idea

- **We design Tatoo to achieve flexible tracing.**
  - Instruction tagging - flexibly gather data.
  - Taint inference - efficiently assist dataflow fuzzing.

Instruction tagging

- Tatoo differentiates and filters the instructions by tagging them.
  - Reduce the volume of traced data.
  - Achieve efficient tracing.
- Add memory tagging and a programmable coprocessor.
- Send the instruction log to the coprocessor when the instruction is in the write-back stage.

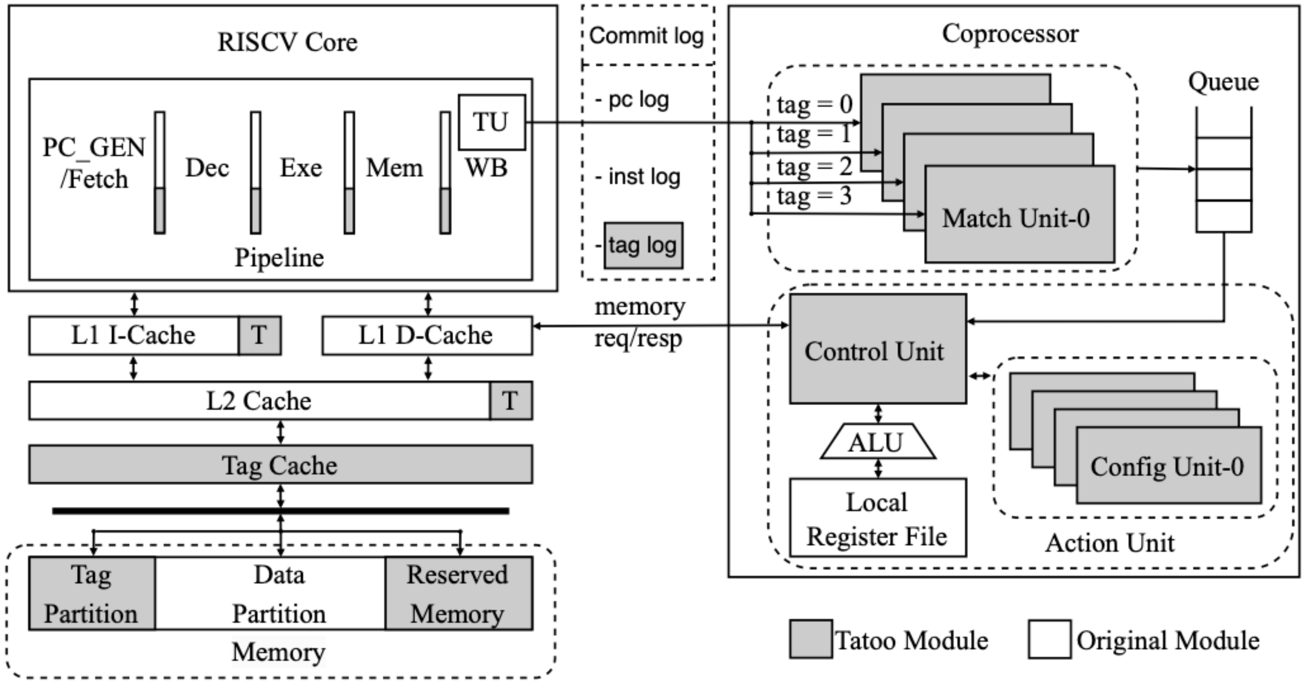


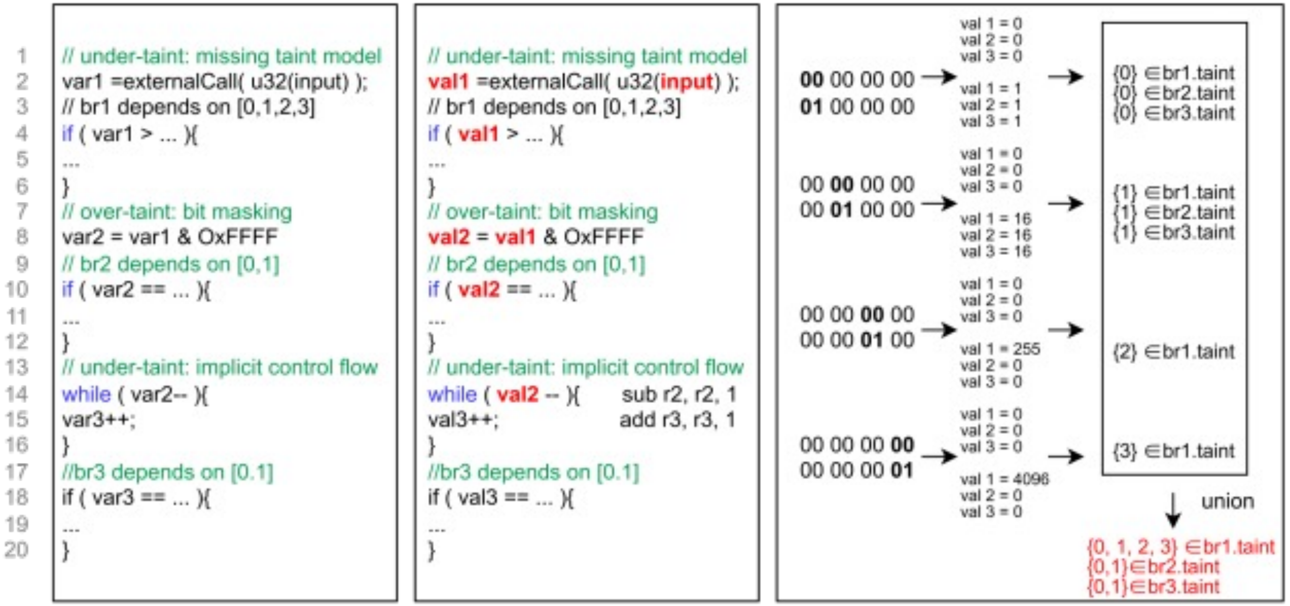
Figure 6: The architecture of Tatoo

Taint inference for dataflow-assisted fuzzing

- Tatoo uses Taint Inference rather than Dynamic Taint Analysis (DTA).

Table 1: The difference between DTA and taint inference

	Overhead	Effectiveness	Manual effort
Dynamic Taint Analysis	High	Suffer from implicit flow issues	Substantial
Taint Inference	Low	Can tackle implicit flow issues	Minimal



(a) code sample (b) DTA (c) Taint Inference

Figure 7: DTA and taint inference

The workflow of Tatoo

Preparation Stage:

① Perform static analysis.

② Create a tagged program.

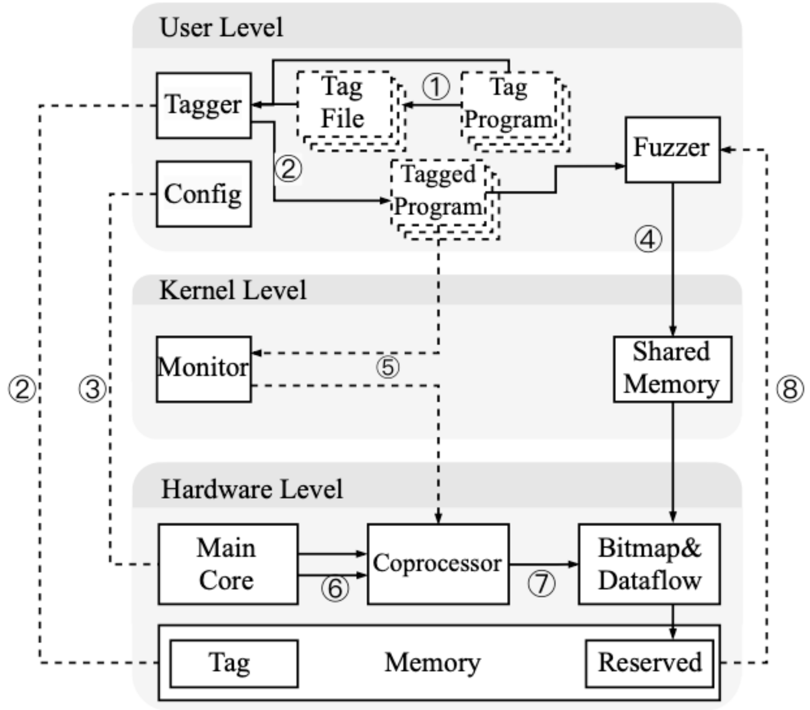


Figure 8: The workflow of Tatoo

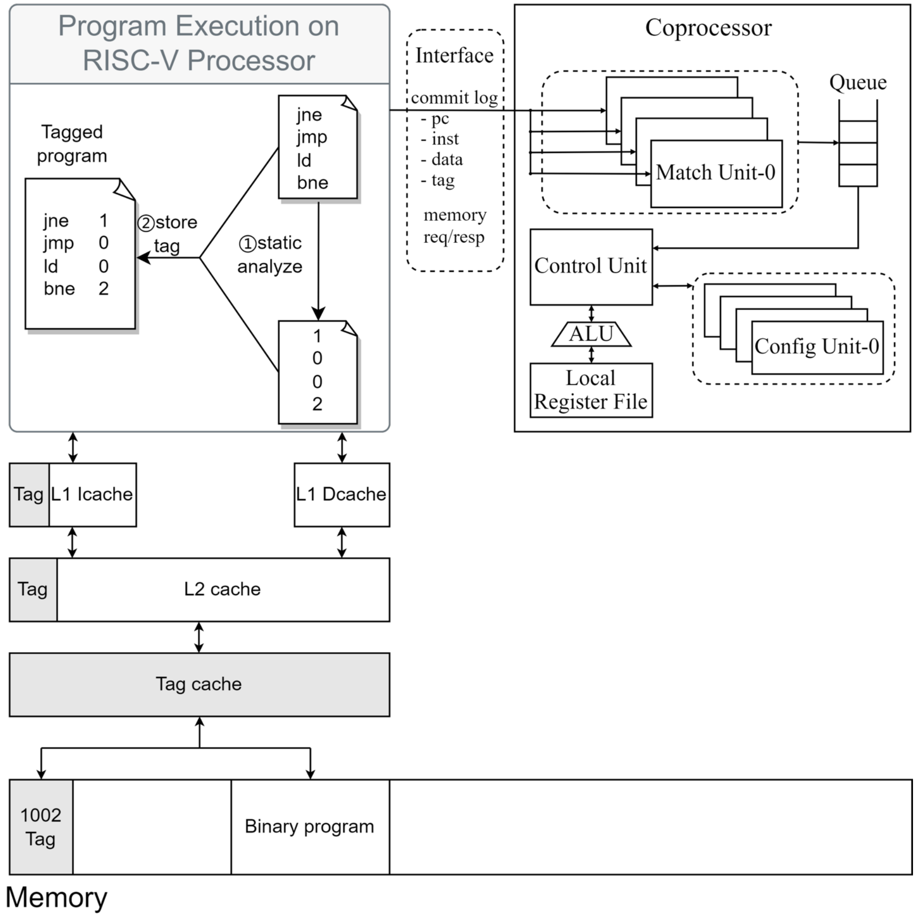


Figure 9: The hardware detail of Tatoo

The workflow of Tatoo

Preparation Stage:

③ Configure coprocessor.

④ Assign shared memory.

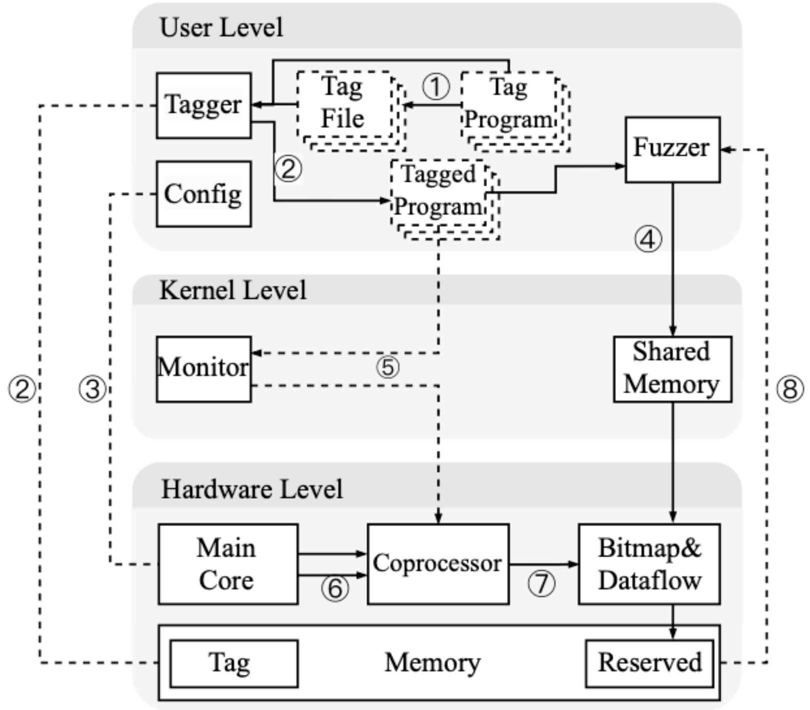


Figure 8: The workflow of Tatoo

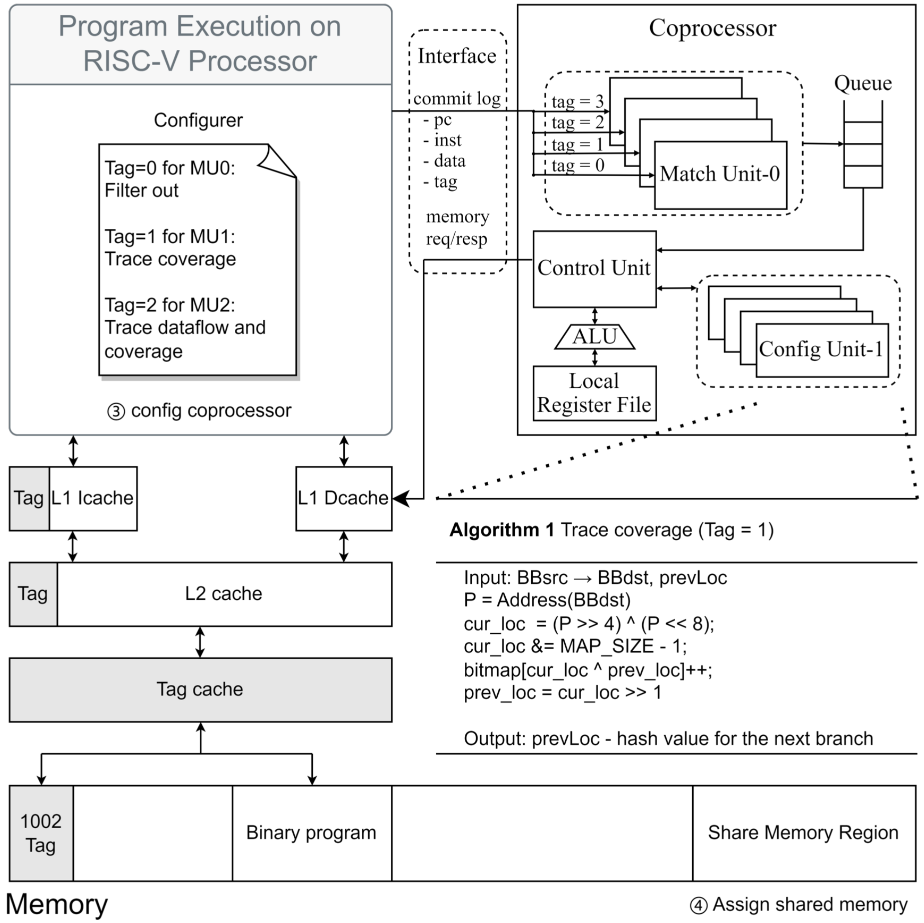


Figure 10: The hardware detail of Tatoo

The workflow of Tatoo

Fuzzing Stage:

⑤ Monitor tagged program.

⑥ Execute program.

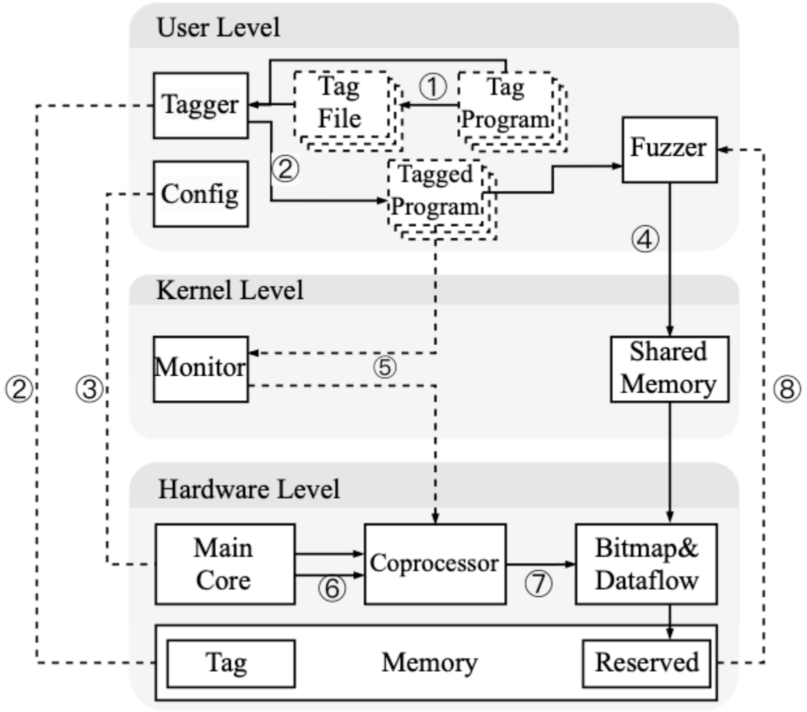


Figure 8: The workflow of Tatoo

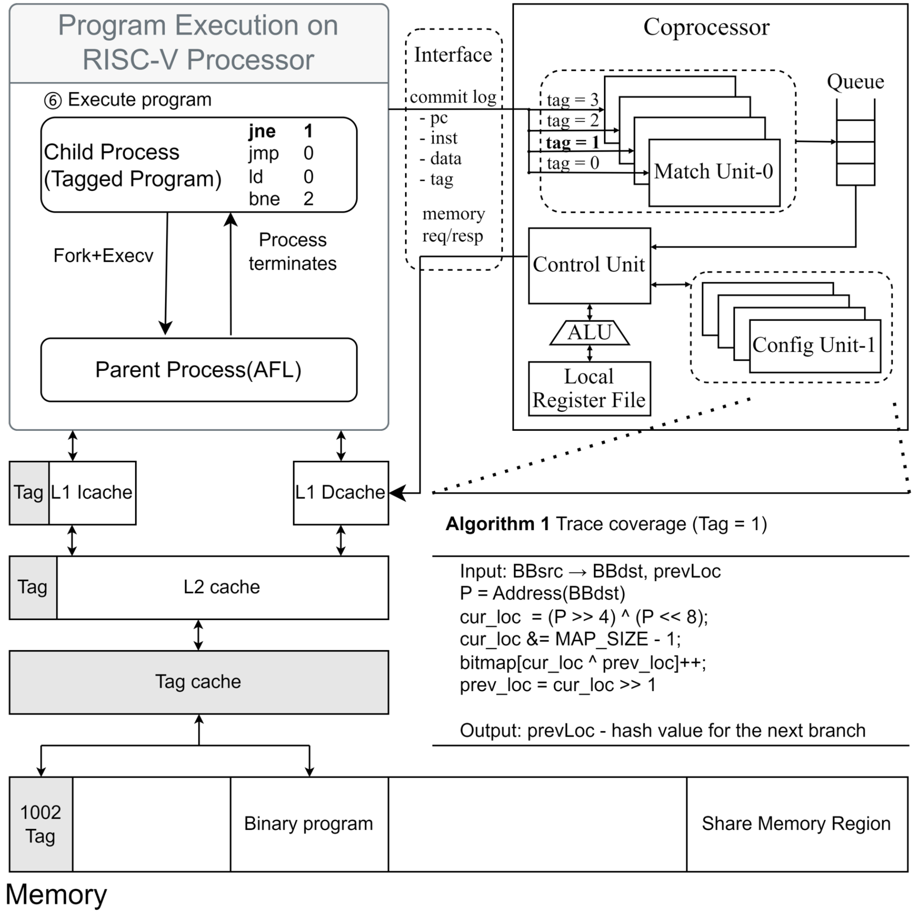


Figure 11: The hardware detail of Tatoo

The workflow of Tatoo

Fuzzing Stage:

⑤ Monitor tagged program.

⑥ Execute program.

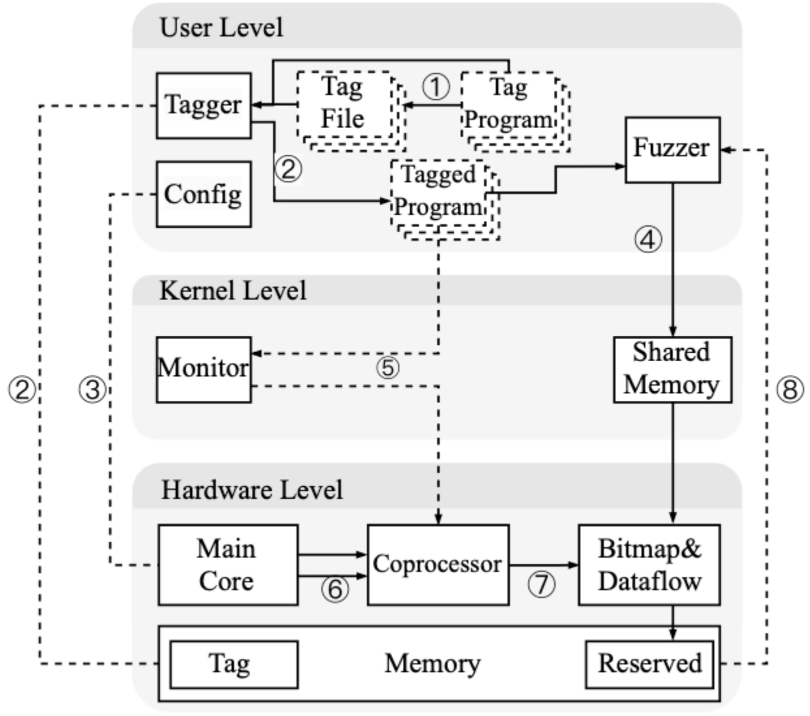


Figure 8: The workflow of Tatoo

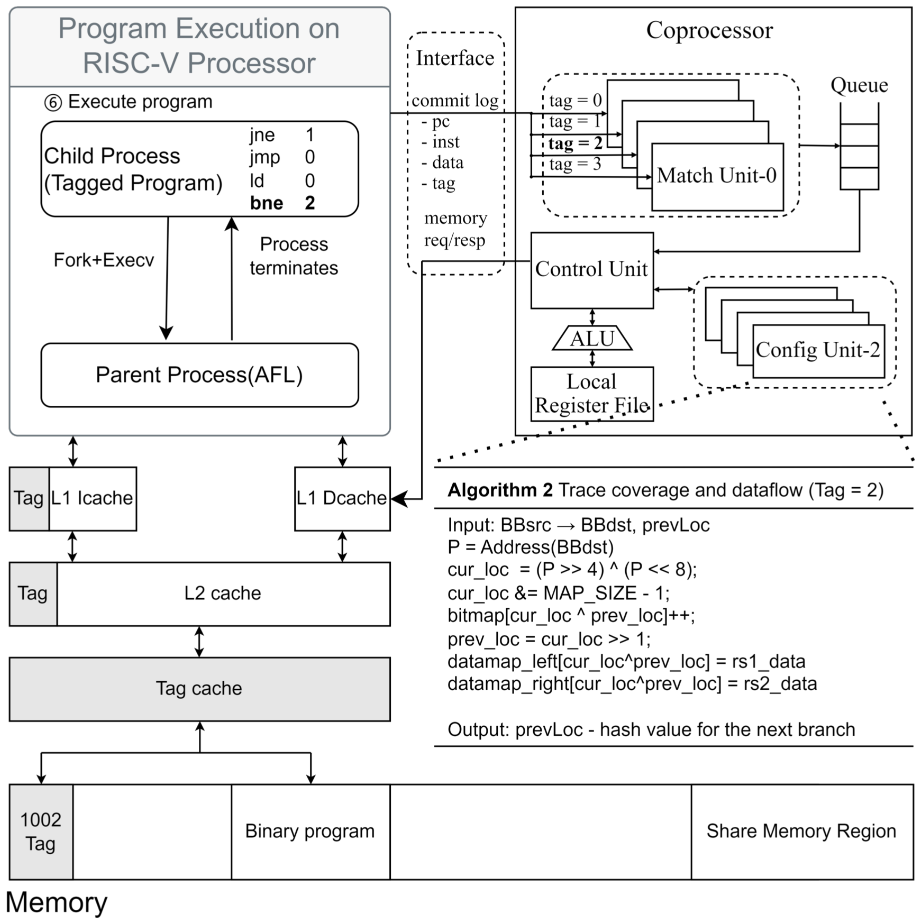


Figure 12: The hardware detail of Tatoo

The workflow of Tatoo

Fuzzing Stage:

⑦ Collect runtime data.

⑧ Analyze the data.

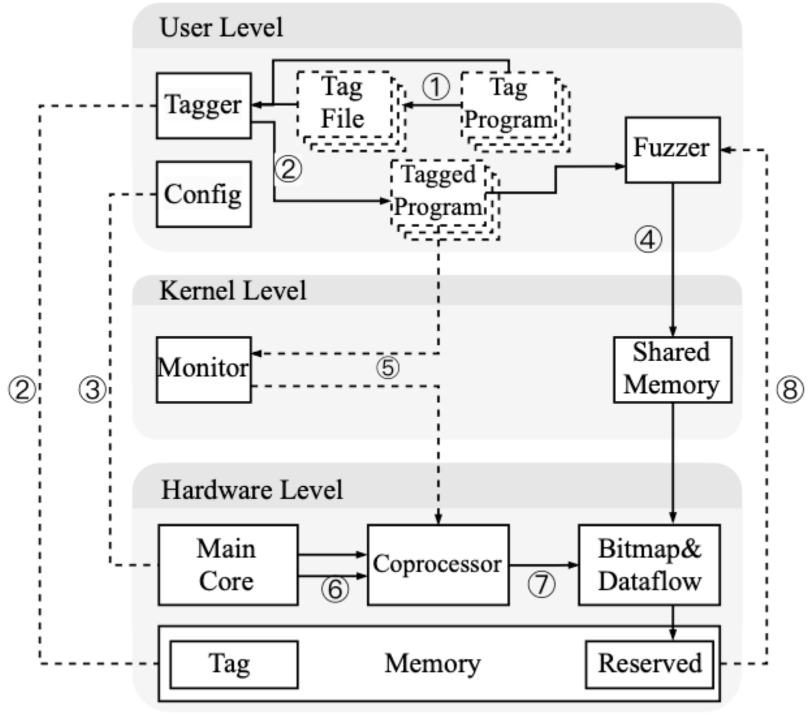


Figure 8: The workflow of Tatoo

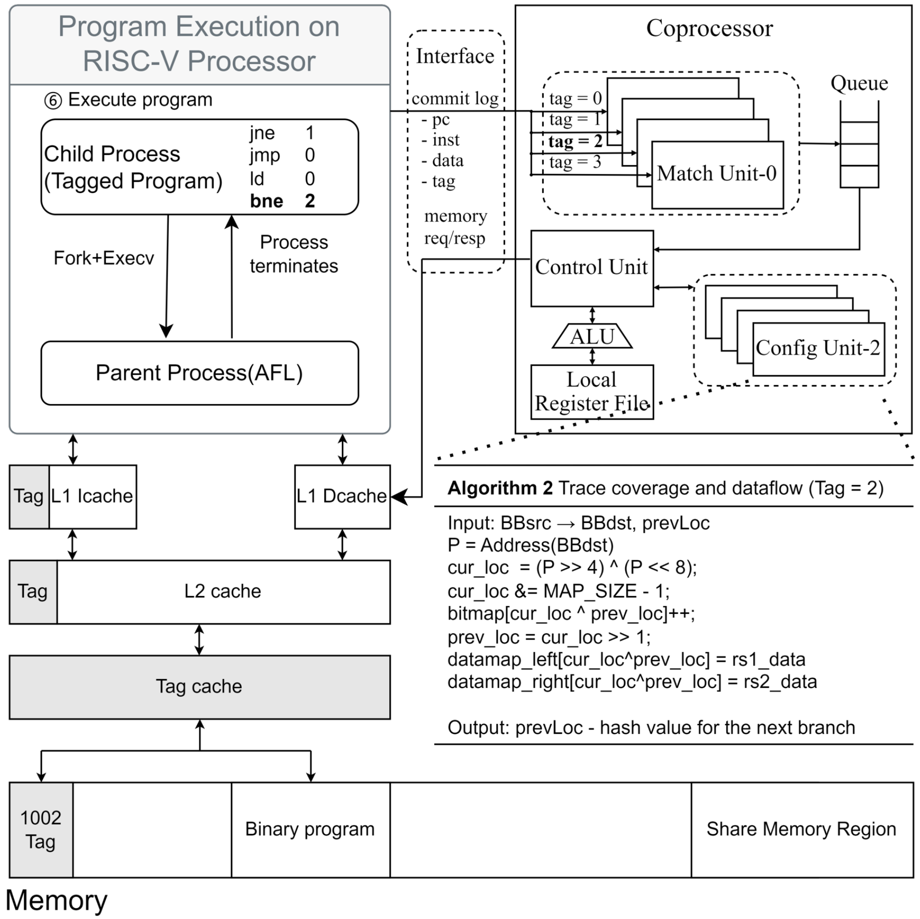


Figure 12: The hardware detail of Tatoo

Evaluation

- Deployed on the Xilinx Kintex-7 FPGA KC705 evaluation board.
- Evaluated by 7 real-world applications.
- 1. Performance Overhead
- 2. Throughput
- 3. Edge Coverage
- 4. Area Overhead

Table 2: Target binaries evaluated in our evaluation

Subjects	Size Change
objdump –dwarf-check -C -g -f -dwarf -x @@	10.08 M to 12.79 M
readelf -a @@	5.02 M to 7.92 M
size @@	5.27 M to 5.34 M
nasm -f elf -o sample @@	5.80 M to 8.04 M
bison @@	5.22 M to 9.27 M
tiff2bw @@ /dev/null	1.29 M to 1.35 M
tiffnfo @@	1.36 M to 1.40 M

Evaluation

- 1. Performance Overhead
  - Baseline: The original program
  - 2,000 seeds for each program
  - Around 5% to 12%, average 8.7%
  - > AFL(60%), PHMon(11.55%)
- 2. Throughput
- 3. Edge Coverage
- 4. Area Overhead

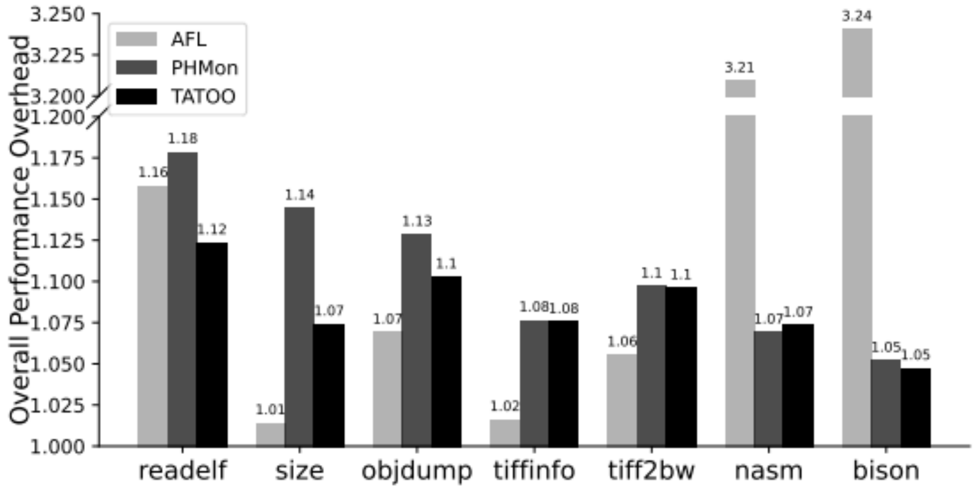


Figure 13: The overall performance measured by real-world programs

Evaluation

- 1. Performance Overhead
- **2. Throughput**
  - **24-hour experiment**
  - **> PHMon(4.10%), AFL(29.03%), AFL_QEMU(769.88%)**
- 3. Edge Coverage
- 4. Area Overhead

Table 3: Throughput in our evaluation

Binary	AFL	PHMON	AFL_QEMU	TATOO
readelf	213,532	230,181	44,753	232,856
objdump	194,865	213,353	26,648	215,445
size	192,172	209,527	23,887	214,242
nasm	68,677	147,442	7,386	158,351
bison	158,394	185,359	24,190	201,095
tiffinfo	225,086	231,928	56,156	243,002
tiff2bw	227,444	234,146	53,124	242,670

Evaluation

- 1. Performance Overhead
- 2. Throughput
- **3. Edge Coverage**
  - **> AFL(8.2%), PHMon(8.6%), AFL-QEMU(24%) except tiff2bw**
- 4. Area Overhead

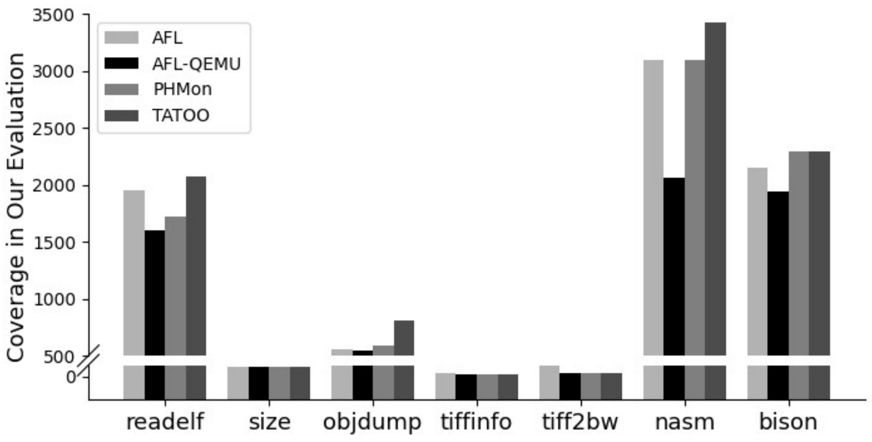


Figure 14: Edge coverage in our evaluation

Evaluation

- 1. Performance Overhead
- 2. Throughput
- 3. Edge Coverage
- **4. Area Overhead**
  - **Memory Tagging (15%)**
  - **Hardware Tracing (18%)**

Table 4: Hardware resource cost of Tatoo

	Whole Systems		Power
	Slice LUTs	**Slice Registers**	
Without Tatoo	73,784	39,035	3.324W
With Tatoo	98,807	52,210	3.309W
%	+ 33.9%	+ 33.8%	- 0.5%

Summary

- A flexible hardware tracing platform.
- Solution: Utilizing memory tagging architecture and hardware tracing to achieve flexible tracing.
- Scenario: Binary-only fuzzing.
- Code: https://github.com/Compass-All/TATOO
- Mail: zhangfw@sustech.edu.cn

TATOO

THE CHIPS TO SYSTEMS CONFERENCE

SHAPING THE NEXT GENERATION OF ELECTRONICS

JUNE 23-27, 2024

MOSCONE WEST CENTER
SAN FRANCISCO, CA, USA

Thank you !
Any Question ?