FDTC 2025

14th September 2025 in Kuala Lumpur, Malaysia

Improving Fault Vulnerability Detection via Rehosting and Comparative Analysis of Open-source Tools

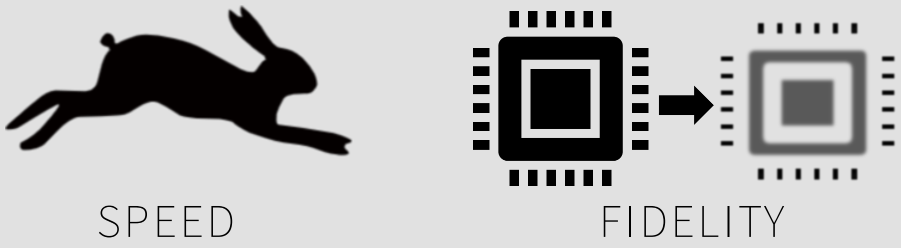
Shoei Nashimoto*

*Mitsubishi Electric Corporation

This presentation is based on results obtained from a project, JPNP24003, commissioned by the New Energy and Industrial Technology Development Organization (NEDO).

Challenges

1) Fast but low fidelity

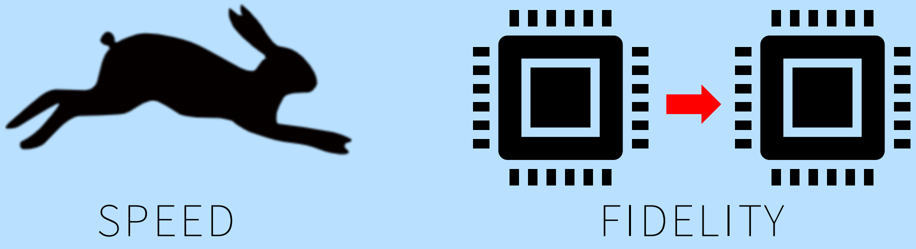


2) Lack of benchmark dataset

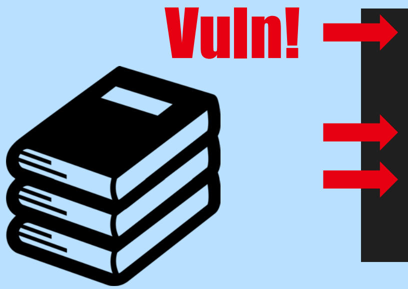
```
ldr r3, [r7, #20] ; i
ldr r2, [r7, #12] ; a1
add r3, r2
ldrb  r2, [r3, #0] ; r2
ldr r3, [r7, #20] ; i
```

Solution

1) Rehosting improves fidelity



2) Creation of labeled dataset




```
ldr r3, [r7, #20] ; i
ldr r2, [r7, #12] ; a1
add r3, r2
ldrb  r2, [r3, #0] ; r2
ldr r3, [r7, #20] ; i
```

Results: Rehosting achieves near-perfect recall (99%-100%) while keeping high speed

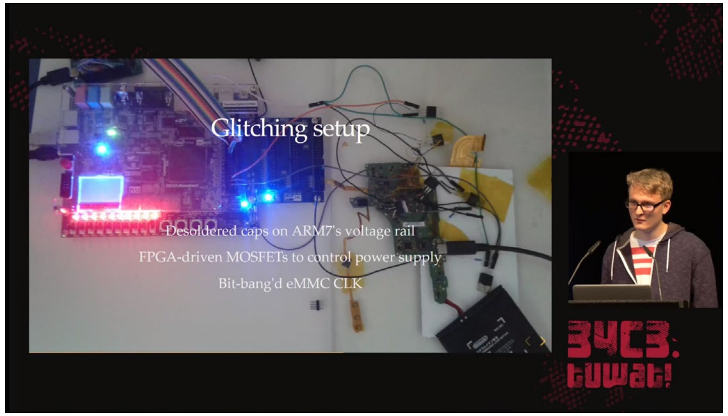
Outline

- Background
- Fault Vulnerability Detection
- Rehosting
- Experiment
- Discussion
- Conclusion

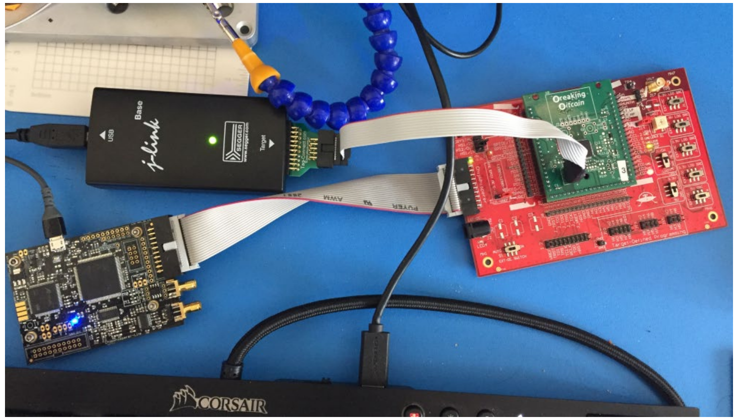
Background

Fault Injection Attack (FIA) against Actual Product

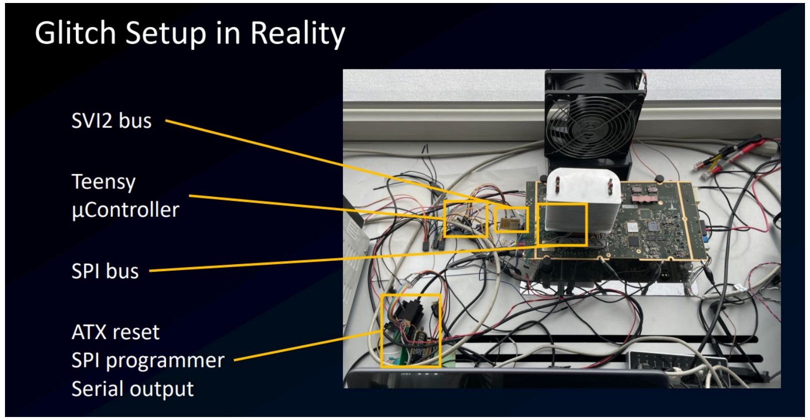
If software is secure, attack hardware



Game console [5]



Cryptocurrency wallet [6]



Automotive unit [4]

- Software security for embedded devices is improving
- Attackers seek the next "weakest point," targeting fault attacks

[4] C. Werling, N. K¨uhnapfel, H. N. Jacob, and O. Drokin, "Jailbreaking an Electric Vehicle in 2023 or What It Means to Hotwire Tesla's x86-Based Seat Heater," BlackHat Asia, 2023.
[5] plutoo, derrek and naehrwert, "Console Security – Switch", 2017
[6] J. Datko, C. Quartier, and K. Belyayev, "Breaking Bitcoin Hardware Wallets: Glitches cause stitches!" DEF CON 2017, 2017

FIA Resistance in Software Design Phase

MITSUBISHI ELECTRIC
Changes for the Better

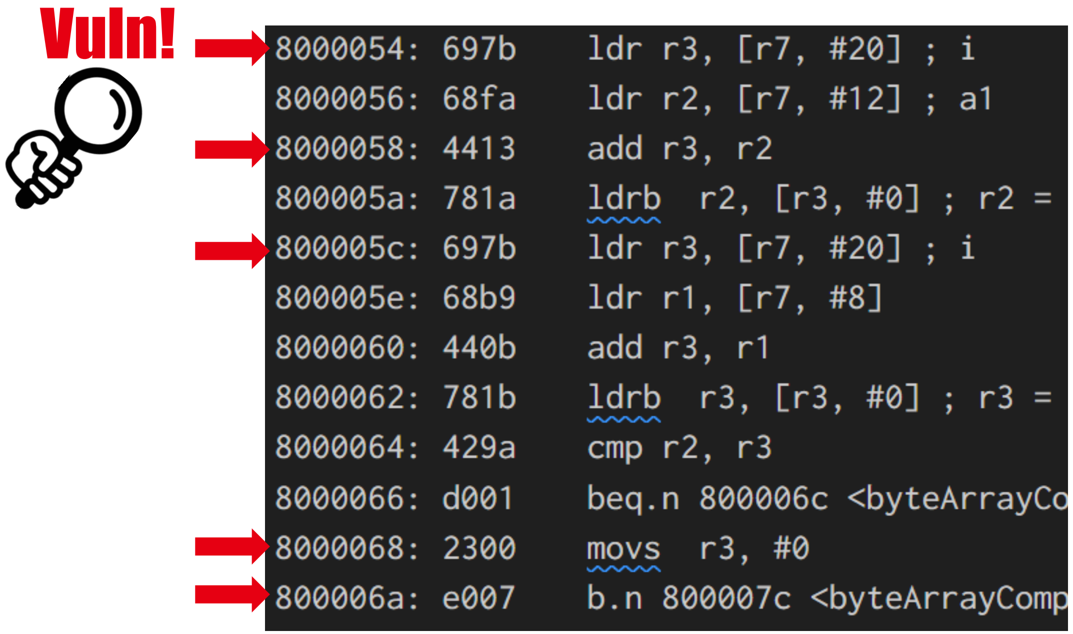
Increasing demand for fault vulnerability detection (FVD) from *binary*

Why software-based detection?

- Scalable, reproducible, cost-effective comparing to physical FIA

Why binary?

- Higher-level languages diverge from physical mechanisms
- Binary is the closest to hardware within the software layer



Concept of FVD from binary

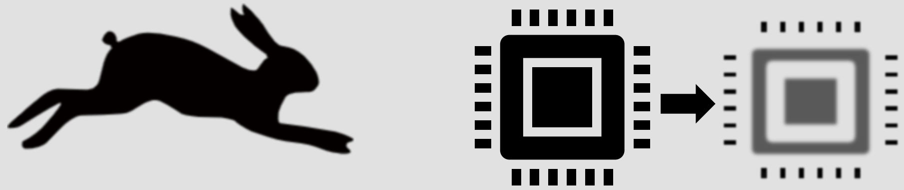
Challenges

1) Speed vs. Fidelity, 2) Lack of benchmark dataset

1. Trade-offs between speed and fidelity:

- Instruction Set Emulation (ISE)-based tools prioritize speed by omitting system-level emulation (↔ QEMU)
- This simplification leads to inaccurate memory and register initialization, causing false positives and negatives

1) Fast but low fidelity



2. Lack of Standardized Benchmarks:

- Lack of ground-truth labeled datasets makes fair and reproducible tool comparison difficult
- This is due to the difficulty in manual verification of “all fault patterns”

2) Lack of benchmark dataset

?

```
ldr r3, [r7, #20] ; i
ldr r2, [r7, #12] ; a1
add r3, r2
ldrb  r2, [r3, #0] ; r2
ldr r3, [r7, #20] ; i
```

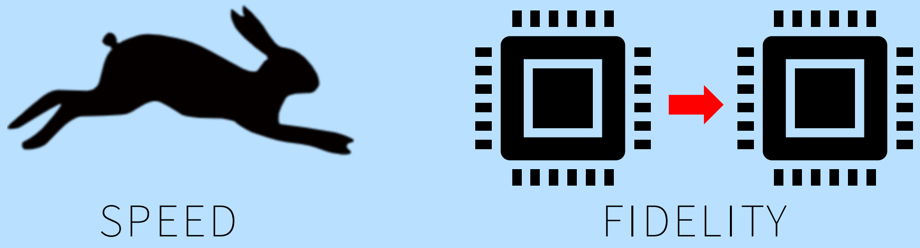
Our Contributions

1) Rehosting for FVD, 2) Integration of results from multiple tools

1. Rehosting for Improved Fidelity:

- Introduced a rehosting technique to mitigate misclassification in ISE-based tools (e.g., FaultFinder).
- Accurately reproduces memory and register states, ensuring faithful system initialization

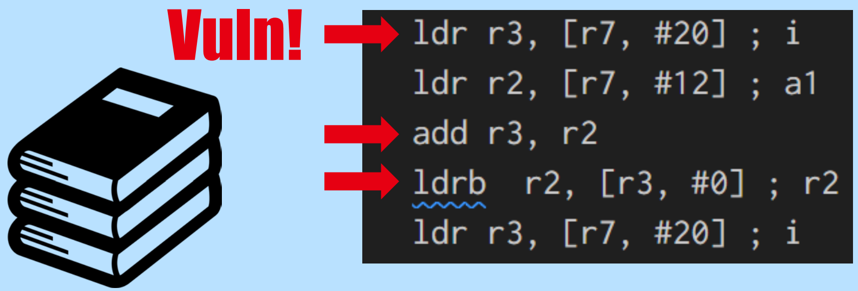
1) Rehosting improves fidelity



2. Labeled Benchmark Creation and Comparative Analysis

- Constructed a labeled benchmark dataset by manually verifying vulnerabilities from multiple tools
- This significantly reduces verification cost and enables systematic misclassification pattern identification
- This benchmark enables to evaluate the speed and detection performance of FVD tools

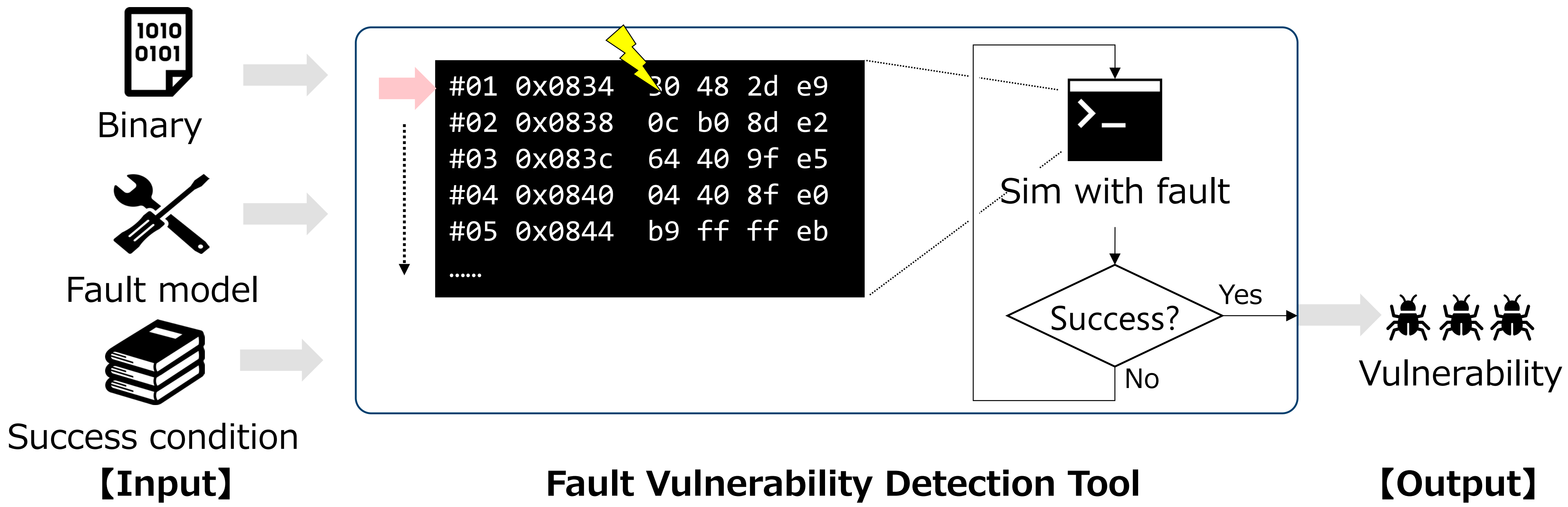
2) Creation of labeled



Fault Vulnerability Detection

Processing Flow of Fault Vulnerability Detection (FVD)

Run binary simulating fault and check if it satisfies success condition



- Vulnerabilities can be comprehensively detected by injecting faults into all instructions and verifying success conditions
- Example) Password authentication, DFA on AES

FVD Characteristics of FVD Tools

Differences in tools make common evaluation difficult

Tool	Architecture	Platform	Success condition
FaultFinder	ARM / x86 / RISC-V	Unicorn **(ISE)**	Target addr., reg/mem
FaultArm	ARM / x86	- (Parser)	Anti-pattern
Fault-injection-simulation*	ARM	angr	Target addr.
ARCHIE	ARM / / RISC-V	**QEMU**-*system*	Target addr., reg/mem
ARMORY	ARM	M-ulator **(ISE)**	Target addr.
Chaos Duck	ARM / x86	Native/ **QEMU**-*user*	stdout / exit code
FaultInjectionSimulator*	/ x86	Native	exit code

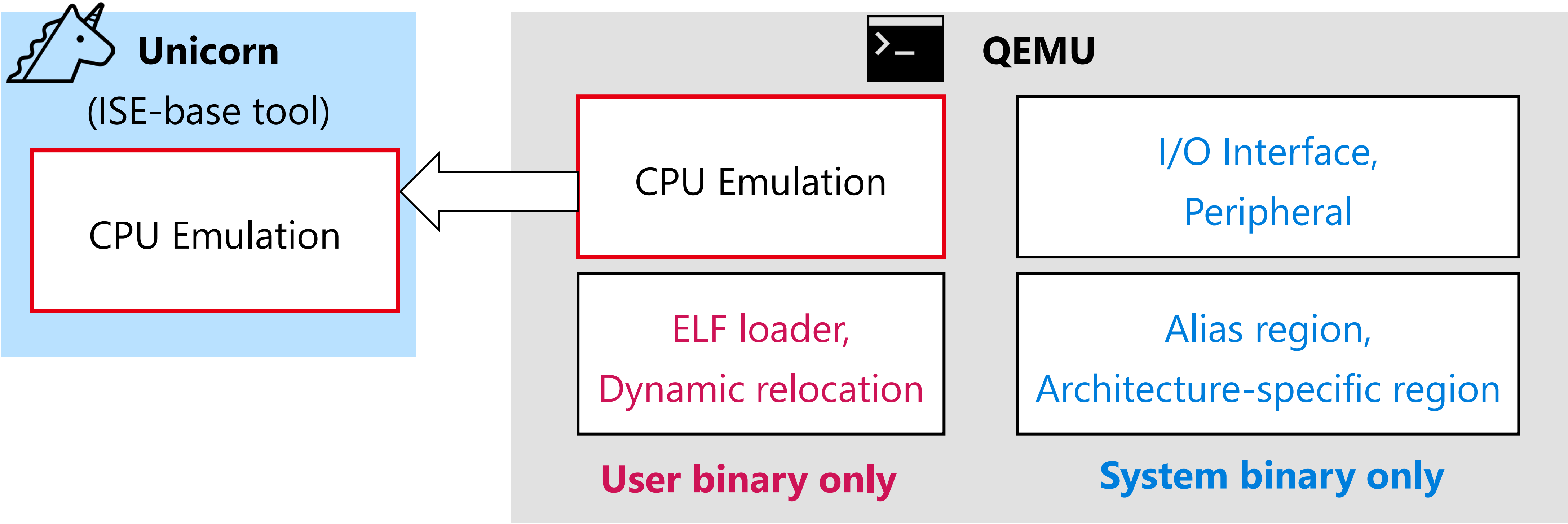
*Not published as a paper

- Seven OSS-based FVD tools selected from GitHub
- Differences:
  - Architecture: Even if the architecture is the same, the user binary/system binary differs
  - Platform: ISE, QEMU, and native execution are primary
  - Success condition: target address, register/memory condition, exit code, ⋯

Rehosting

Problem: ISE-based Tools Lack Memory Reproducibility

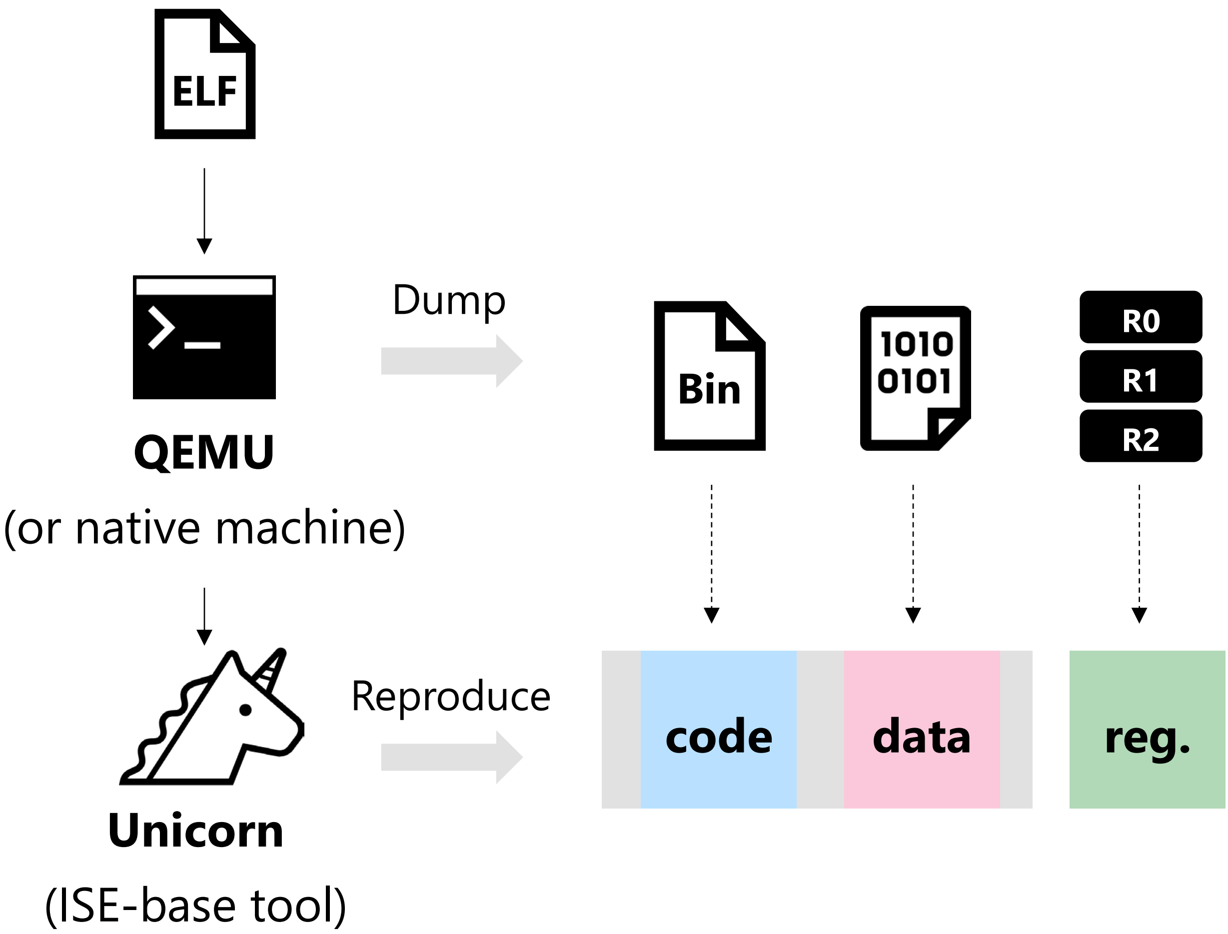
ISE-based tools are fast but have low fidelity



- ISE-based tools achieves high speed by emulating only execution of instruction set
- Omitted functions cause **differences in memory state** at entry point (*main()*)
  → Leads to false positives and false negatives in vulnerability detection

Proposed: Rehosting

Porting the processor state (memory) during QEMU/native execution



Concept of rehosting

1. Execute ELF on QEMU
2. Debug ELF with gdb
3. Break at entry point (`main()`)
4. Dump memory (code and memory)
5. Dump CPU state (register)
6. Reproduce CPU state

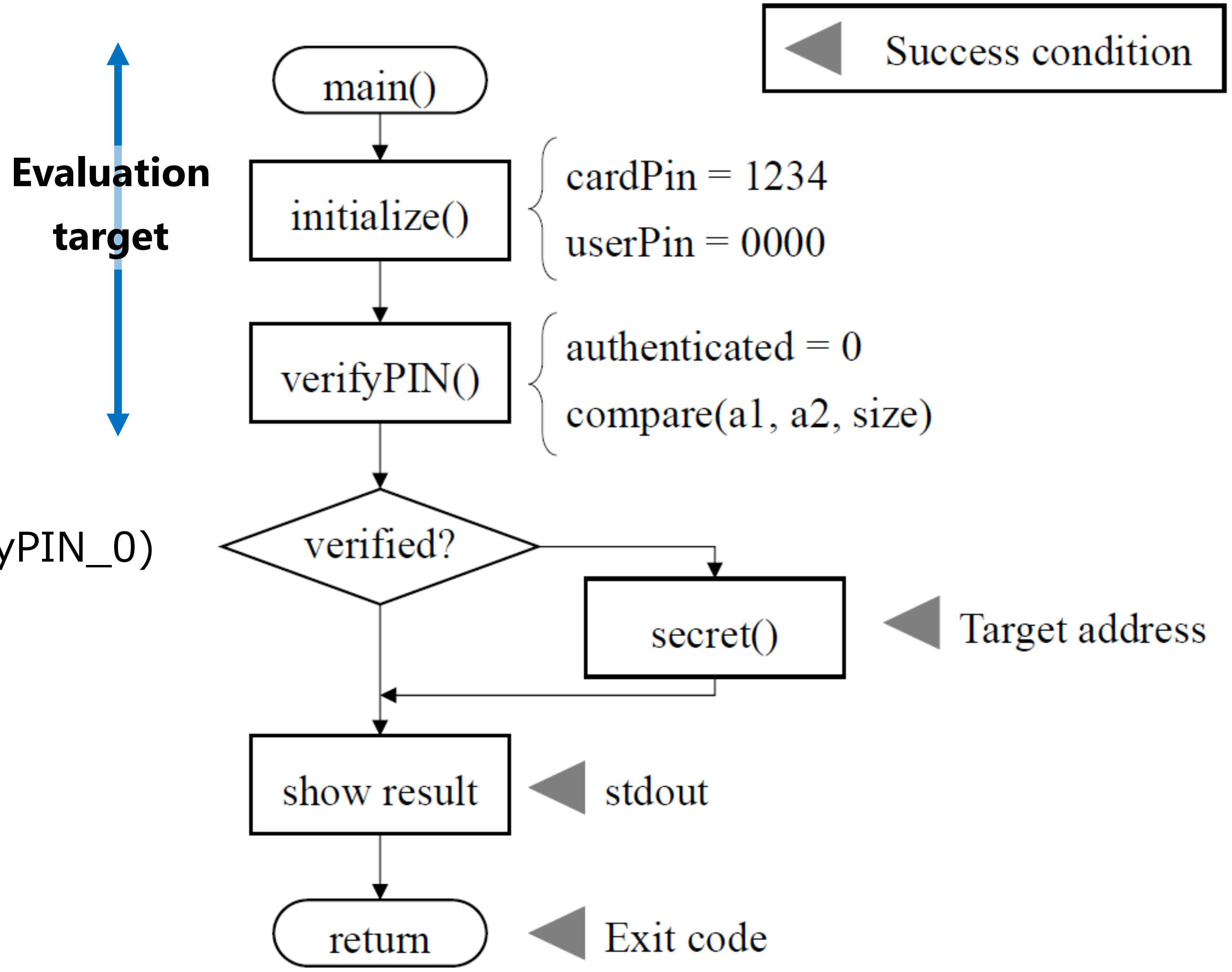
Typical rehosting flow

Experiment

Setup: Cross-Tool Evaluability Design

Case study: "PIN verification" as a common benchmark

- Target: ARMv7m, ARMv7a, x86
- Fault models:
  - Instruction skipping (IS)
  - Bit-flip on Instruction (BF-I)
  - Bit-flip on Register (BF-R)
- Benchmark program
  - PIN verification without countermeasure (VerifyPIN_0) from FISSC [12]
    - Hardcoded userPIN fails in PIN verification
  - Program modification: Support all success conditions (stdout / reached path / exit code)



[12] L. Dureuil, G. Petiot, M. Potet, T. Le, A. Crohen, and P. de Choudens, "FISSC: A fault injection and simulation secure collection," in Computer Safety, Reliability, and Security - 35th International Conference, SAFECOMP 2016, Trondheim, Norway, September 21-23, 2016, Proceedings, 2016, pp. 3–11.

Evaluation Metrics

Common vulnerability detection approach: maximizing recall while maintaining high precision

Metrics 1: Missed Faults

- Proportion of detected cases out of all vulnerabilities
- $Recall = \frac{TP}{TP+FN}$

Metrics 2: Effective Faults

- Proportion of actual vulnerabilities out of all detected cases
- $Precision = \frac{TP}{TP+FP}$

Ground Truth

- Integrating vulnerabilities reported by each tool, followed by manual analysis

Predict \ Actual	TRUE	FALSE
TRUE	**TP** True Positive	**FP** False Positive
FALSE	**FN** False Negative	**TN** True Negative

Result: Model1 Instruction Skipping

FaultFinder run on multi architectures with high performance and fast speed

- Rehosting eliminates all FNs, increasing recall up to 100%, while decreasing precision in some cases
- Rehosting keeps fast speed (low overhead)
- ISE-based tools are **>50x** faster than QEMU-based tools

	Tool*	Time [s]	Recall	Precision
ARM v7a	**FaultFinder (rehost)**	1.7	1.00	1.00
	FaultFinder	1.6	0.88	0.94
	FaultArm	2.0	0.59	0.29
	Chaos Duck	107.0	0.88	1.00
ARM v7m	**FaultFinder (rehost)**	1.7	1.00	0.89
	FaultFinder	1.6	0.94	1.00
	FaultArm	1.9	0.31	0.31
	ARCHIE	173.7	1.00	0.89
	ARMORY	0.4	0.75	1.00
x86	**FaultFinder (rehost)**	1.4	1.00	1.00
	FaultFinder	1.3	1.00	1.00
	FaultArm	1.8	0.07	0.20
	Chaos Duck	12.4	0.93	1.00

ISE **QEMU** **Others**

Both time and recall are good **Good** **Bad**

* Only tools proposed in papers

Result: Model2,3 Bit-Flip on Instruction/Register

MITSUBISHI ELECTRIC
Changes for the Better

(Same trend) FaultFinder run on multi architectures with high performance and fast speed

(repeated)

- Rehosting eliminates all FNs, increasing recall up to 100%, while decreasing precision in some cases
- Rehosting keeps fast speed (low overhead)
- ISE-based tools are **>400x** faster than QEMU-based tools

	Tool*	Time [s]	Recall	Precision
ARM v7a BF-I	**FaultFinder (rehost)**	8.9	0.99	0.95
	FaultFinder	5.2	0.80	0.89
	Chaos Duck	3883.9	0.85	0.96
ARM v7m BF-I	**FaultFinder (rehost)**	4.4	1.00	0.85
	FaultFinder	2.8	0.92	1.00
	ARCHIE	8661.5	1.00	0.85
	ARMORY	0.4	0.70	0.93
ARM v7m BF-R	**FaultFinder (rehost)**	7.4	1.00	0.99
	FaultFinder	4.3	0.97	1.00
	ARCHIE	8401.7	1.00	0.90
	ARMORY	0.4	0.62	1.00

ISE **QEMU** **Others**

Both time and recall are good **Good** **Bad**

* Only tools supporting bit-flip

Misclassification Analysis & Limitation

Misclassification Analysis

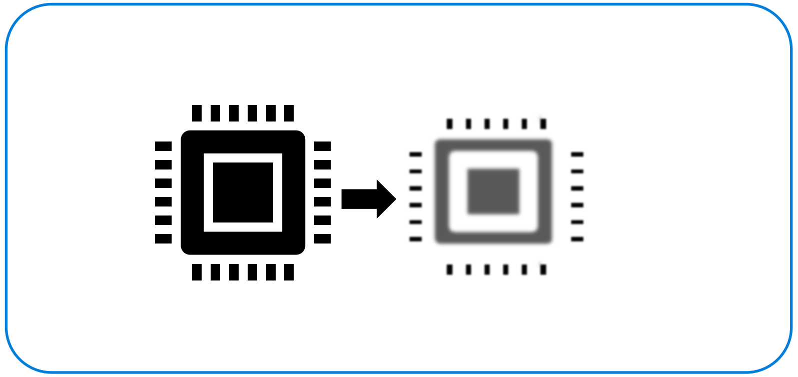
Misclassification fall into 10 patterns

Tool	FP	FN	Init. state	Invalid access	Write attribute	Unimplemented inst.	Others
FaultFinder (rehost)	56	1	0	0	56	1	0
FaultFinder	30	88	110	1	6	1	0
Chaos Duck	10	47	0	0	0	0	57
ARCHIE	59	0	0	0	41	0	18
ARMORY	11	133	68	1	1	3	71

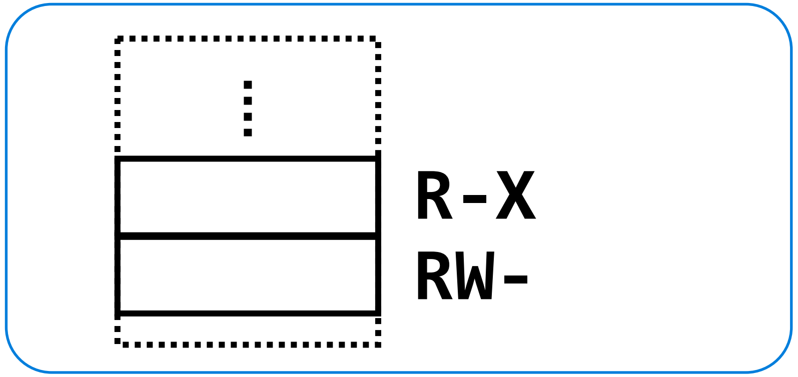
- **Initialization state / Invalid access**: inaccurate memory/register init. and memory mapping
  → Rehosting
- **Write attribute**: Rehosting increased FPs due to the ability to correctly recognize memory regions
  → Implementation of memory attributes
- **Unimplemented instruction**: ISE omits some specific instructions (e.g., ARM NEON).
  → Manual verification

Key Takeaways

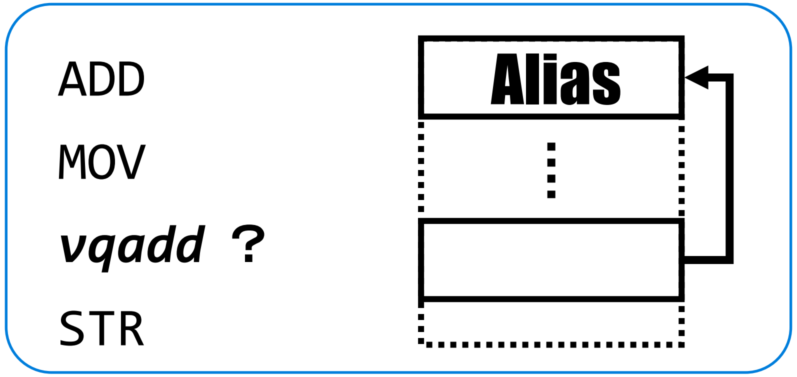
6 design principles for FVD



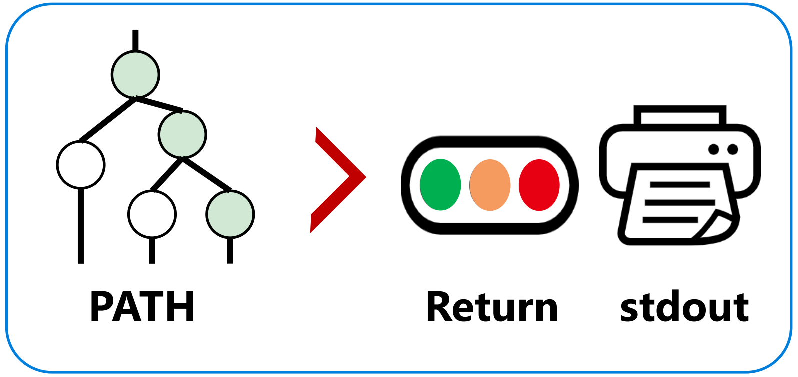
1. Initial state



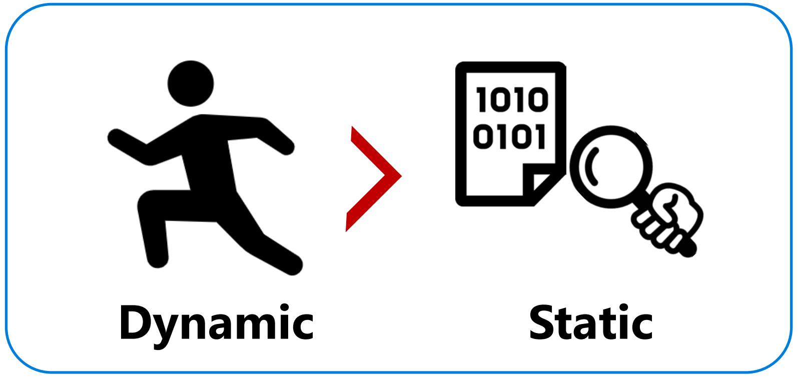
2. Memory map & attribute



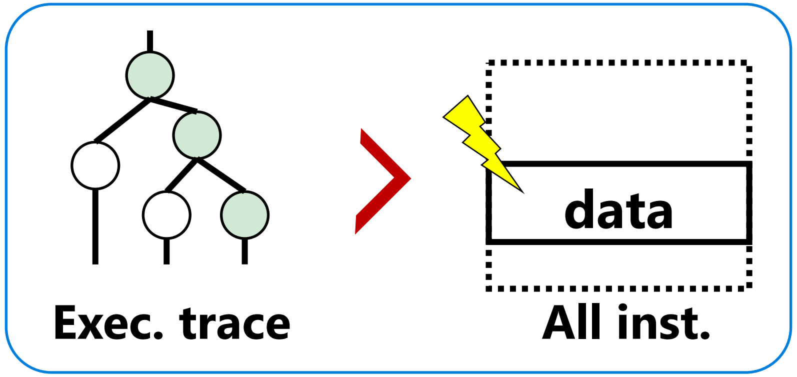
3. Unimplemented inst. & arch.-specific feature



4. Address reachability (success condition)



5. Dynamic analysis



6. Execution trace

Applicability & Limitation

Ground truth depends on 1) 7 FVD tools capabilities and 2) execution platform

Rehosting Applicability

- Applicable to *any* ISE-based tools that support pre-execution memory modification
  → ARMORY does not have such interface

Limitation

1. Tool dependency
    - Our ground truth is limited by the detection capabilities of the selected tools
2. Execution platform dependency
    - Our ground truth depends on I) native execution for x86, II) QEMU for ARM

Conclusion

Challenge: 1) Speed vs. Fidelity for FVD tools. 2) No benchmark dataset.

Contributions:

- *Rehosting* improved fidelity in ISE-based tool
- We created dataset "**FIVBinBench**" and it is available on GitHub[*]
- Comparative analysis of existing OSS-based FVD tools

- ***Conclusion***:
  FaultFinder with rehosting offers the best balance of speed and performance

[*] https://github.com/pyth0n14n/FIVBinBench

Future work

- Implement memory attribute on FaultFinder
- More complex programs such as cryptography and VerifyPIN with countermeasure

MITSUBISHI ELECTRIC
Changes for the Better

Fault Injection Vulnerability Binary Benchmark

Main components

- *Dataset* including 1) binary and 2) result
- *Tools* including FaultFinder with rehosting

Instruction information	**Detection results**	**Ground truth**
IP / code / opcode	in each tool	with misclass. type

IP	code	opcode	oprand	comment	FaultARM	fault-injection-simulation	ChaosDuck	FaultFinder (rehosting)	FaultFinder	Grand Truth	Type
00000598 <byteArrayCompare>:											
598:	e52db004	push	{fp}	@ (str fp, [sp, #-4]!)							
5b0:	e54b3015	strb	r3, [fp, #-21]			o	o	o	o	o	-
5b4:	e3a03000	mov	r3, #0		o	o	o	o	o	o	-
5b8:	e50b3008	str	r3, [fp, #-8]		o		o	o		o	Initial state

Example of result (TSV format)

MITSUBISHI ELECTRIC

Changes for the Better