Bin2Wrong:

A Unified Fuzzing Framework for Uncovering Semantic Errors in Binary-to-C Decompilers

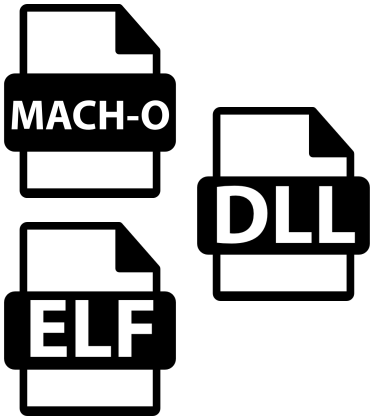
Zao Yang
University of Utah

Stefan Nagy
University of Utah

FUTURES^3
LAB FUTURE TECHNOLOGY FOR USABLE, RELIABLE, & EFFICIENT SECURITY OF SOFTWARE & SYSTEMS

Decompilers: Critical to Software & Systems

- **High-level goal:** recover equivalent C code from compiled binary artifacts



Compiled Binaries

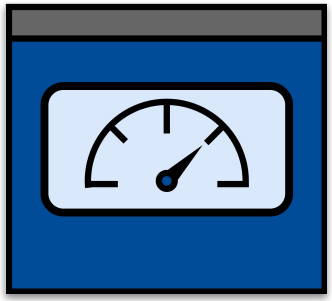
Decompilers

```c
int func1(int x) {
    int result = 0;
    if (x > 10) {
        result = x * 2;
    } else {
        result = x + 5;
    }

    for (int i = 0; i < 3; i++) {
        result += i;
    }

    return result;
}
```

Recovered C Code

Decompilers: Critical to Software & Systems

- **High-level goal:** recover equivalent C code from compiled binary artifacts
  - Fundamental to downstream tasks that center on **source-unavailable components**



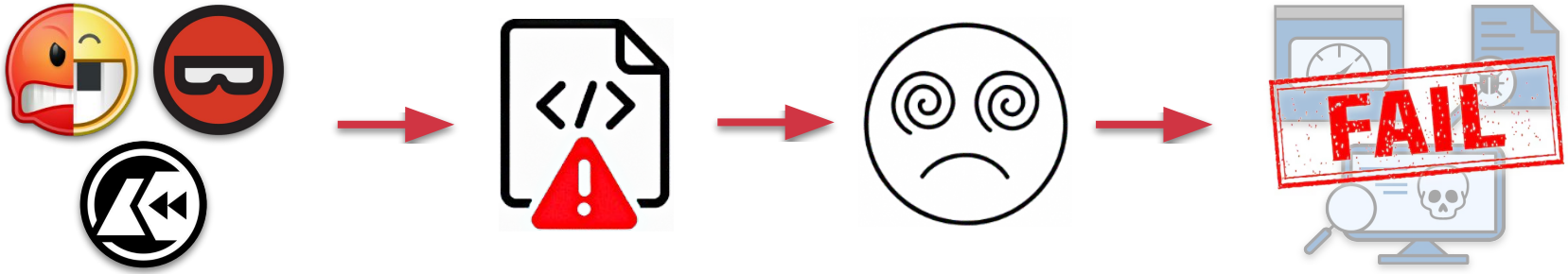
Proprietary Software
Performance Tuning

Obfuscated Binary
Malware Analysis

Commercial Software
Vulnerability Discovery

Decompilers: Critical to Software & Systems

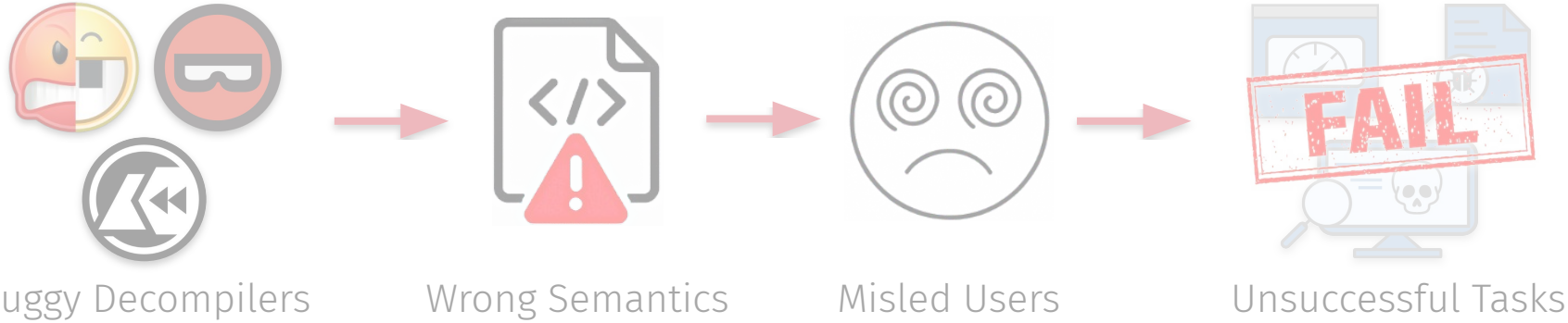
- **High-level goal:** recover equivalent C code from compiled binary artifacts
  - Fundamental to downstream tasks that center on **source-unavailable components**
  - **Success of these downstream tasks** often undermined by **incorrect decompilation**



Buggy Decompilers → Wrong Semantics → Misled Users → Unsuccessful Tasks

Decompilers: Critical to Software & Systems

- **High-level goal:** recover equivalent C code from compiled binary artifacts
  - Fundamental to downstream tasks that center on **source-unavailable components**
  - **Success of these downstream tasks** often undermined by **incorrect decompilation**



Problem: what factors chiefly influence **decompilation errors**?

Decompilers: Challenges to Accurate Decompilation

- Manually distilled **64** prior decompiler bugs into **four distinct factors**

Origin(s)	**Affected Decompilers**
Instructions	Ghidra [1], Radare2 [2], Reko [3], RetDec [4, 5]
Refinement	Angr [6, 7], Binary Ninja [8, 9, 10], RetDec [11]
If / Else	Binary Ninja [12, 13, 14], Ghidra [15], RetDec [16, 17]
Loops	Angr [18, 19, 20], Binary Ninja [21, 22], Reko [23]
Goto	Angr [24, 25], Binary Ninja [26], Reko [27], RetDec [28]
Switches	Binary Ninja [29, 30, 31], Radare2 [32], RetDec [33]
Arguments	Angr [34, 35, 36], Binary Ninja [37, 38, 39, 40]
Variables	Angr [41, 42], Binary Ninja [43], Ghidra [44, 45, 46]
Literals	Angr [47], Binary Ninja [48], Ghidra [49, 50], Reko [51]
Compilers, Opts	Angr [52, 53, 54], Binary Ninja [55], Ghidra [56, 57]
Executable Formats	Angr [58], Binary Ninja [59, 60, 61], Ghidra [62, 63]

Decompilers: Challenges to Accurate Decompilation

- Manually distilled **64** prior decompiler bugs into **four distinct factors**

```
int v0 =
 divide(nmr,dnr);
```

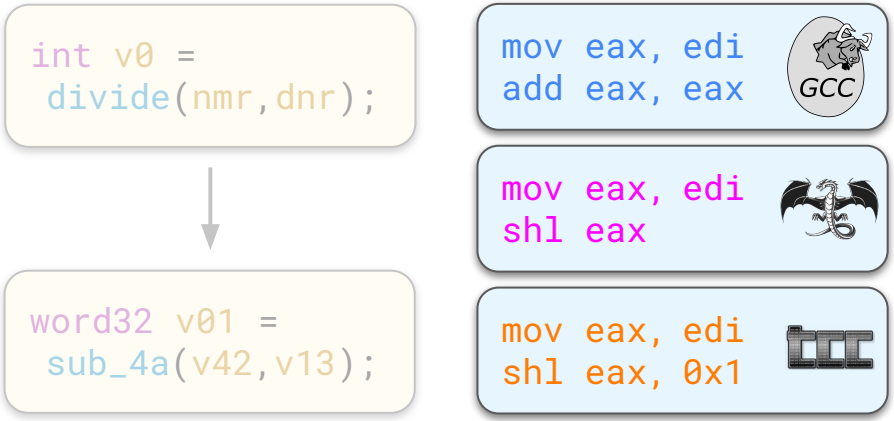


```
word32 v01 =
 sub_4a(v42,v13);
```

Opaqueness of
Source Semantics

Decompilers: Challenges to Accurate Decompilation

- Manually distilled **64** prior decompiler bugs into **four distinct factors**

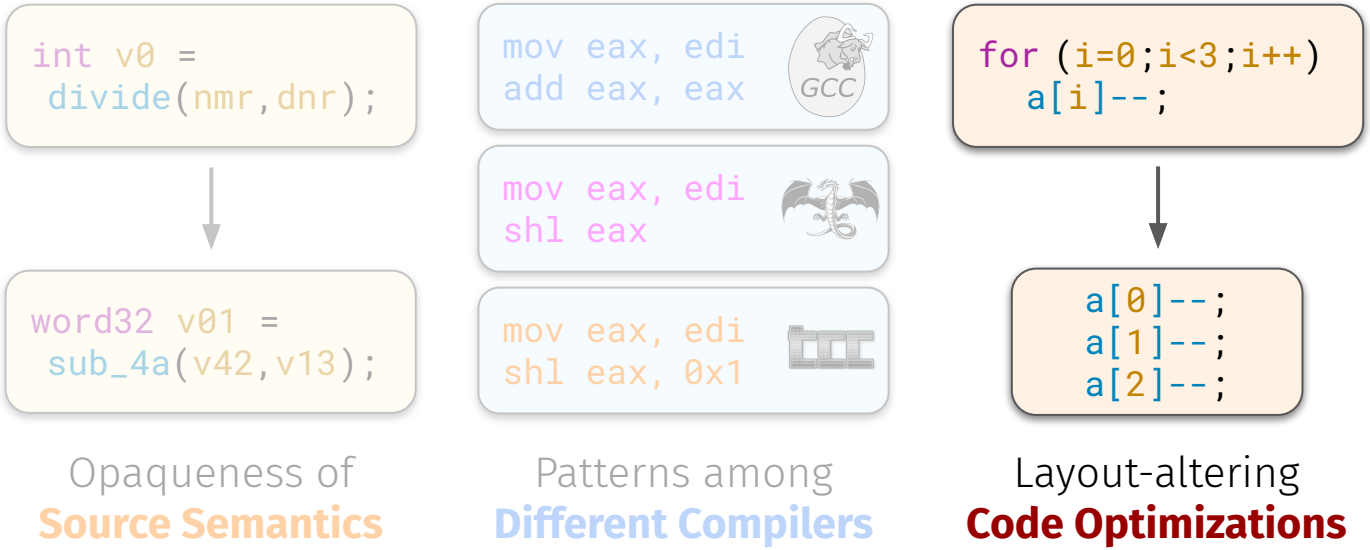


Opaqueness of
Source Semantics

Patterns among
Different Compilers

Decompilers: Challenges to Accurate Decompilation

- Manually distilled **64** prior decompiler bugs into **four distinct factors**



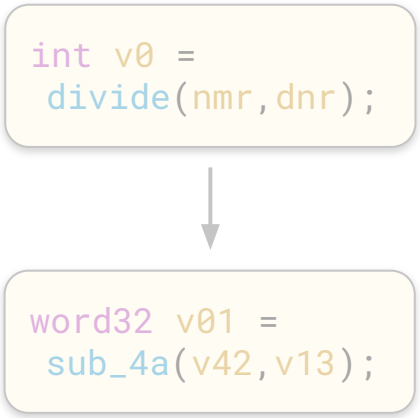
Opaqueness of **Source Semantics**

Patterns among **Different Compilers**

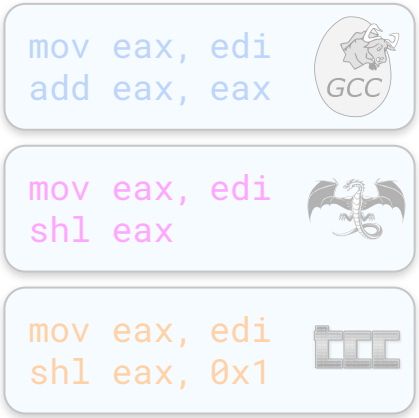
Layout-altering **Code Optimizations**

Decompilers: Challenges to Accurate Decompilation

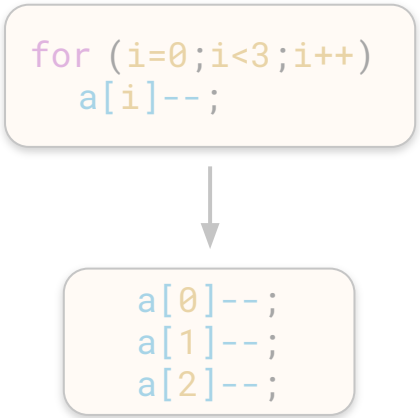
- Manually distilled **64** prior decompiler bugs into **four distinct factors**



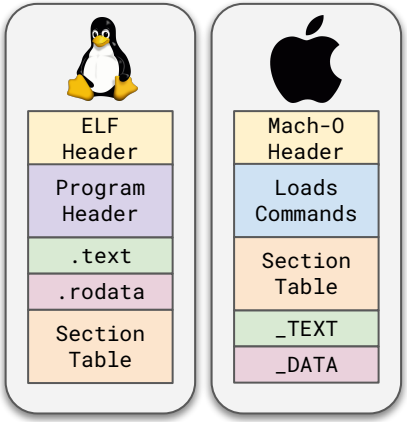
Opaqueness of
Source Semantics



Patterns among
Different Compilers



Layout-altering
Code Optimizations



Differences in
Executable Formats

Decompilers: Challenges to Accurate Decompilation

- Manually distilled **64** prior decompiler bugs into **four distinct factors**
  - Root causes stem not just from **individual factors**—but **combinations** as well

```
if (var != 0xF)
if (0xF != 0xF)
```

Incorrect if condition

```
if (x < 1.5)
if (1.5 < x)
```

Mi-swapped operands

```
0.00000000
0.0000000023283
```

Erroneous float value

```
void myFunc(int a, float b)
void myFunc(int a          )
```

Missed float argument in PE binaries

```
return tailFunc(input);
return 0;
```

Dropped call in tailcall optimization

Decompilers: Challenges to Accurate Decompilation

- Manually distilled **64** prior decompiler bugs into **four distinct factors**
  - Root causes stem not just from **individual factors**—but **combinations** as well

```
if (var != 0xF)
if (0xF != 0xF)
```

Incorrect if condition

```
if (x < 1.5)
if (1.5 < x)
```

Mi-swapped operands

```
0.00000000
0.0000000023283
```

Erroneous float value

```
void myFunc(int a, float b)
void myFunc(int a          )
```

Missed float argument in PE binaries

```
return tailFunc(input);
return 0;
```

Dropped call in tailcall optimization

- Preventing such errors demands **thorough testing along these factors**

Prior Work: Automated Decompiler Testing

- **Problem:** prior approaches only **marginally explore source & compilation**
  - Hardcoded to **limited source constructs**, **specific compilers/optimizations**, and **just ELF**

Approach	Source Construct Diversity			Compilation Configuration Diversity					Decompiler Agnostic
	Expressions	Control Flows	Data	Compilers	Optimizations	ELF	PE	MachO	
DecFuzzer	✔	👎	👎	1	✘	✔	✘	✘	✔
Cornucopia	✔	✔	✔	2	✔	✔	✘	✘	✔
D-Helix	✔	✔	👎	2	👎	✔	✘	✘	✘
DSmith	✔	👎	👎	1	👎	✔	✘	✘	✔

Key: ✔ = fully support, ✘ = no support, 👎 = limited constrained support

Prior Work: Automated Decompiler Testing

- **Problem:** prior approaches only **marginally explore source & compilation**
  - Hardcoded to **limited source constructs**, **specific compilers/optimizations**, and **just ELF**

Approach	Source Construct Diversity			Compilation Configuration Diversity					Decompiler Agnostic
	Expressions	Control Flows	Data	Compilers	Optimizations	ELF	PE	MachO	
DecFuzzer									✔
Cornucopia									✔
D-Helix									✘
DSmith	✔	👎	👎	1	👎	✔	✘	✘	✔

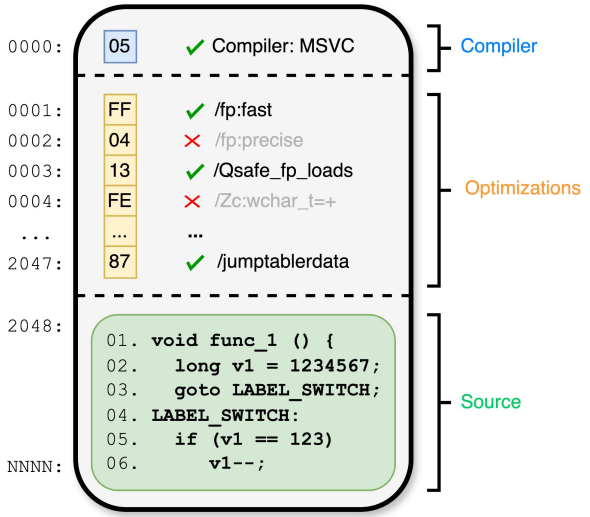
Motivation: how can fuzzing systematically explore these **individual** binary factors and **their many combinations**?

Key: ✔ = fully support, ✘ = no support, 👎 = limited constrained support

Our Solution: Bin2Wrong

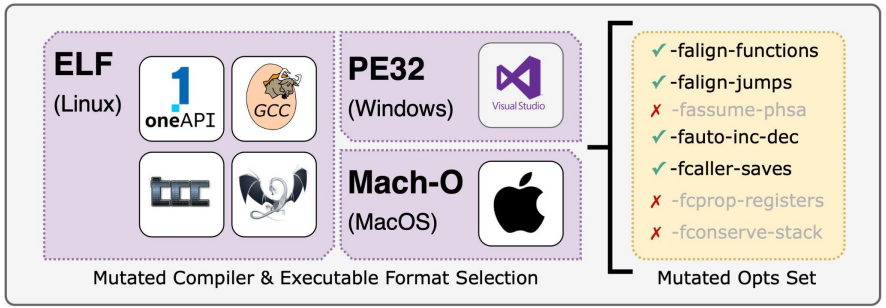
- **Idea:** mutate source code, compilers, optimizations, and format **altogether**
  - **Backbone:** multi-dimensional test case format
  - Mutate **compilation configuration** via byte flips
  - Mutate **source layout** via AST-level mutations

- **Out-of-the-box support for:**
  - **Compilers:** GCC, Clang, ICX, TCC, AppleClang, MSVC
  - **Optimizations:** 5,183 total across all compilers
  - **Formats:** Linux ELF, MacOS MachO, Windows PE
  - **Source:** all C constructs incl. strings/floats/gotos



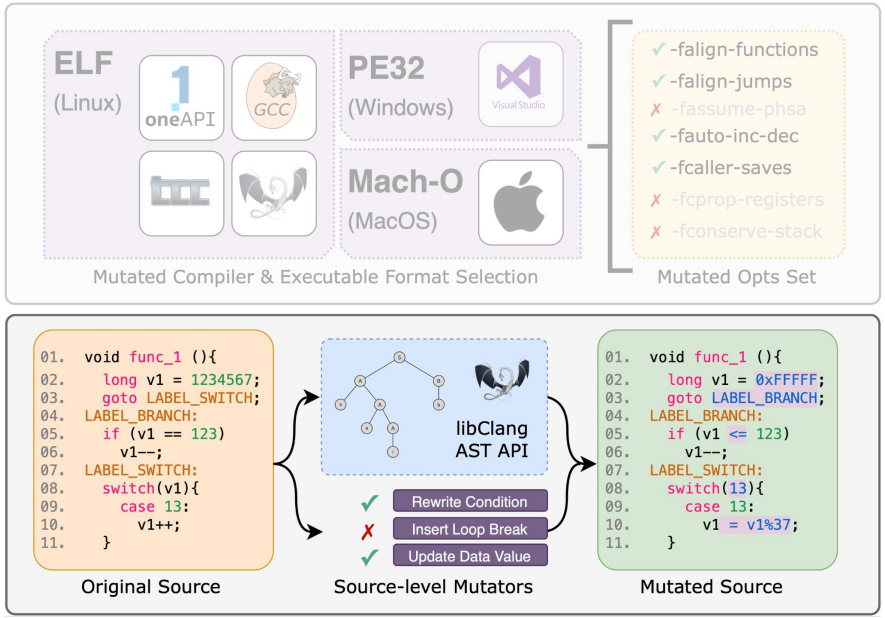
Our Solution: Bin2Wrong

- **Idea:** mutate source code, compilers, optimizations, and format **altogether**



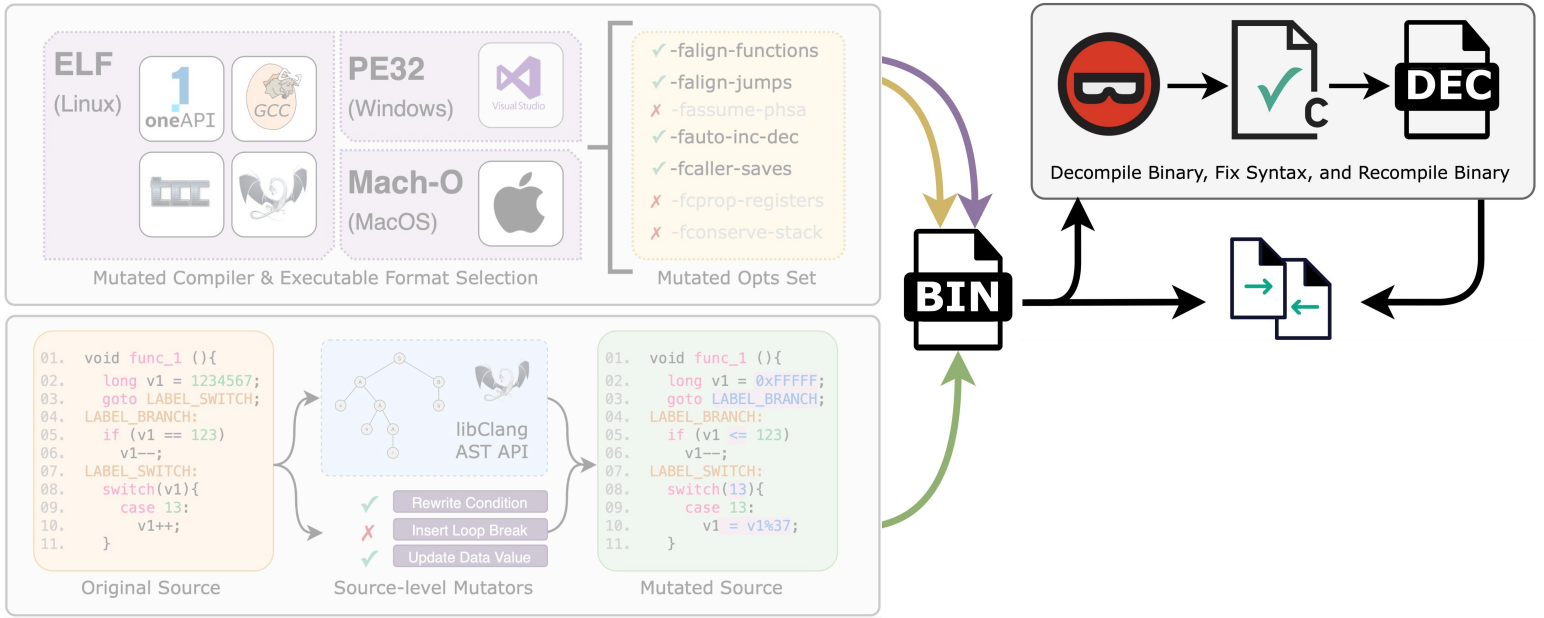
Our Solution: Bin2Wrong

- **Idea:** mutate source code, compilers, optimizations, and format **altogether**



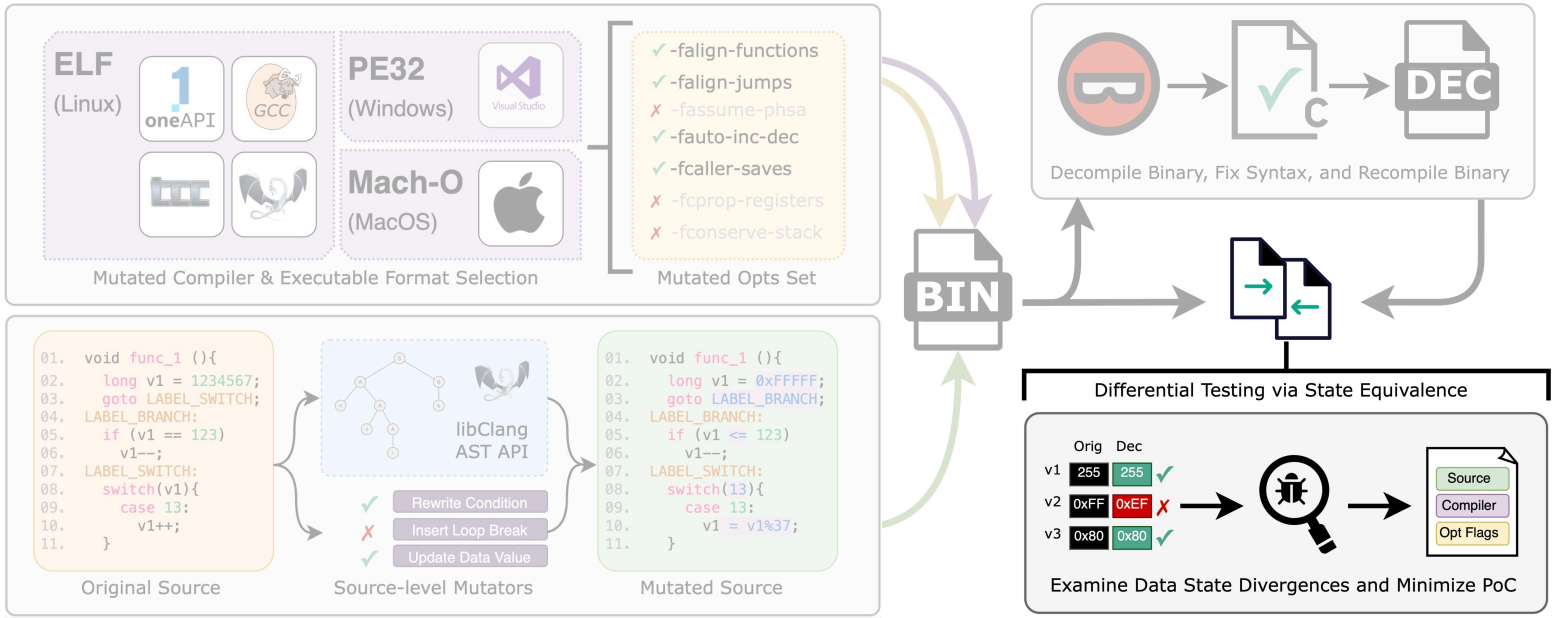
Our Solution: Bin2Wrong

- **Idea:** mutate source code, compilers, optimizations, and format **altogether**



Our Solution: Bin2Wrong

- **Idea:** mutate source code, compilers, optimizations, and format **altogether**



Our Solution: Bin2Wrong

- **Idea:** mutate source code, compilers, optimizations, and format **altogether**

Approach	Source Construct Diversity			Compilation Configuration Diversity					Decompiler Agnostic
	Expressions	Control Flows	Data	Compilers	Optimizations	ELF	PE	MachO	
DecFuzzer	✔	👎	👎	1	✘	✔	✘	✘	✔
Cornucopia	✔	✔	✔	2	✔	✔	✘	✘	✔
D-Helix	✔	✔	👎	2	👎	✔	✘	✘	✘
DSmith	✔	👎	👎	1	👎	✔	✘	✘	✔
Bin2Wrong	✔	✔	✔	6	5,183	✔	✔	✔	✔

By maximizing source *and* compilation diversity, **Bin2Wrong** enables **the most thorough evaluation** of decompilation correctness to date.

Evaluation: Overview

- **Fundamental questions:**
  - Does Bin2Wrong's unified mutation attain **greater binary diversity**?
  - Can Bin2Wrong-generated binaries test **more decompiler internals**?
  - Will Bin2Wrong's binaries discover **more decompilation bugs**?

- Competing decompiler fuzzers:
  - Corncopia (mutates optimizations)
  - DecFuzzer (mutates source code)

Open-source Decompilers	Angr, Radare2, Relyze, RetDec, Reko, Rev.Ng
Commercial Decompiler	Binary Ninja

Benchmarked Decompilers

Evaluation: Binary Diversity

- Measured **mean binary-to-binary diversity** across 10,000 binaries
  - Similarity scoring calculated via **three** state-of-the-art diffing algorithms

Diffing Algorithms
BinDiff *(Zynamics's Graph-Based)*
Radiff2-M *(Eugene W. Myers' O(ND))*
Radiff2-L *(Levenshtein's Edit Distance)*

Evaluation: Binary Diversity

- Measured **mean binary-to-binary diversity** across 10,000 binaries
  - Similarity scoring calculated via **three** state-of-the-art diffing algorithms

Diffing Algorithms	Bin2Wrong vs. Cornucopia	Bin2Wrong vs. DecFuzzer
BinDiff *(Zynamics's Graph-Based)*	+9.398 ✕	+16.189 ✕
Radiff2-M *(Eugene W. Myers' O(ND))*	+8.119 ✕	+15.941 ✕
Radiff2-L *(Levenshtein's Edit Distance)*	+6.131 ✕	+15.089 ✕

- **Takeaways:** Bin2Wrong's unified mutation greatly **increases binary diversity**

Evaluation: Decompiler Code Coverage

- Measured **mean decompiler code coverage** across 24-hour fuzzing trials
  - Computed basic block coverage via the AFL-QEMU-Cov utility

Metrics	Bin2Wrong vs. Cornucopia	Bin2Wrong vs. DecFuzzer
Basic Block Coverage	**+16%**	**+32%**

Evaluation: Decompiler Code Coverage

- Measured **mean decompiler code coverage** across 24-hour fuzzing trials
  - Computed basic block coverage via the AFL-QEMU-Cov utility

Metrics	Bin2Wrong vs. Cornucopia	Bin2Wrong vs. DecFuzzer
Basic Block Coverage	**+16%**	**+32%**
Generated Binaries	**-74%**	**-95%**

- **Takeaways:** Bin2Wrong's binaries each **exercise more decompiler internals**

Evaluation: Decompiler Bugs

- Enumerated **total unique bugs** throughout all 24-hour fuzzing campaigns
  - All bugs manually deduplicated and reported to their respective decompiler developers

Metrics	DecFuzzer	Cornucopia
Total Found Bugs	0	10
Individually-found	0	4
Confirmed or fixed	0	5

Decompiler	DecFuzzer	Cornucopia
Angr	0	2
BinNinja	0	0
Reko	0	3
R2Ghidra	0	1
Relyze	0	2
RetDec	0	2
rev.ng	0	0

Evaluation: Decompiler Bugs

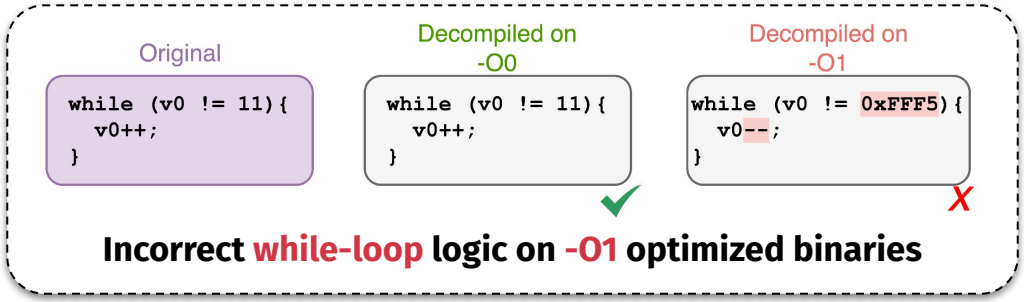
- Enumerated **total unique bugs** throughout all 24-hour fuzzing campaigns
  - All bugs manually deduplicated and reported to their respective decompiler developers

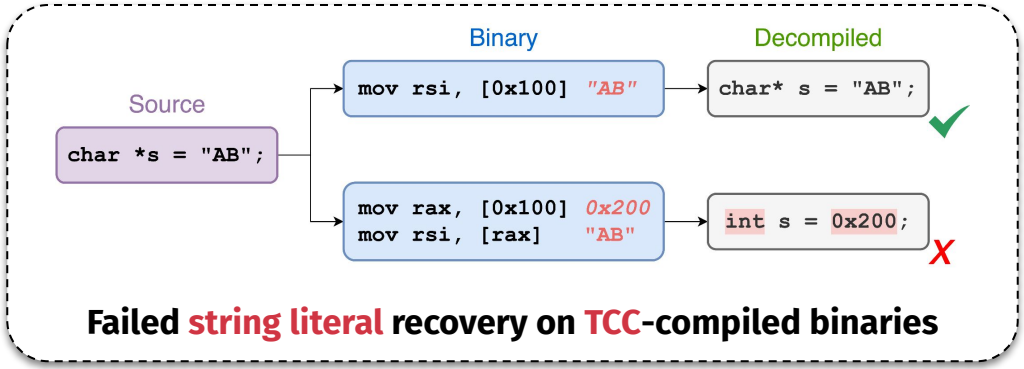
Metrics	DecFuzzer	Cornucopia	Bin2Wrong
Total Found Bugs	0	10	**48**
Individually-found	0	4	**42**
Confirmed or fixed	0	5	**30**

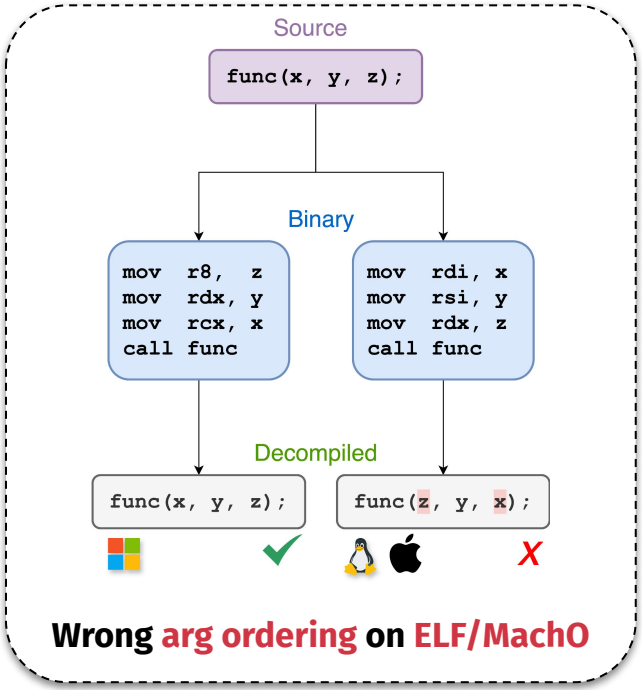
Decompiler	DecFuzzer	Cornucopia	Bin2Wrong
Angr	0	2	**9**
BinNinja	0	0	**11**
Reko	0	3	**6**
R2Ghidra	0	1	**2**
Relyze	0	2	**7**
RetDec	0	2	**11**
rev.ng	0	0	**2**

- **Takeaways:** Bin2Wrong's diverse binaries **expose more decompiler bugs**
  - Since reporting, **30 are now confirmed and/or fixed**

Case Studies: Bin2Wrong-found Bugs

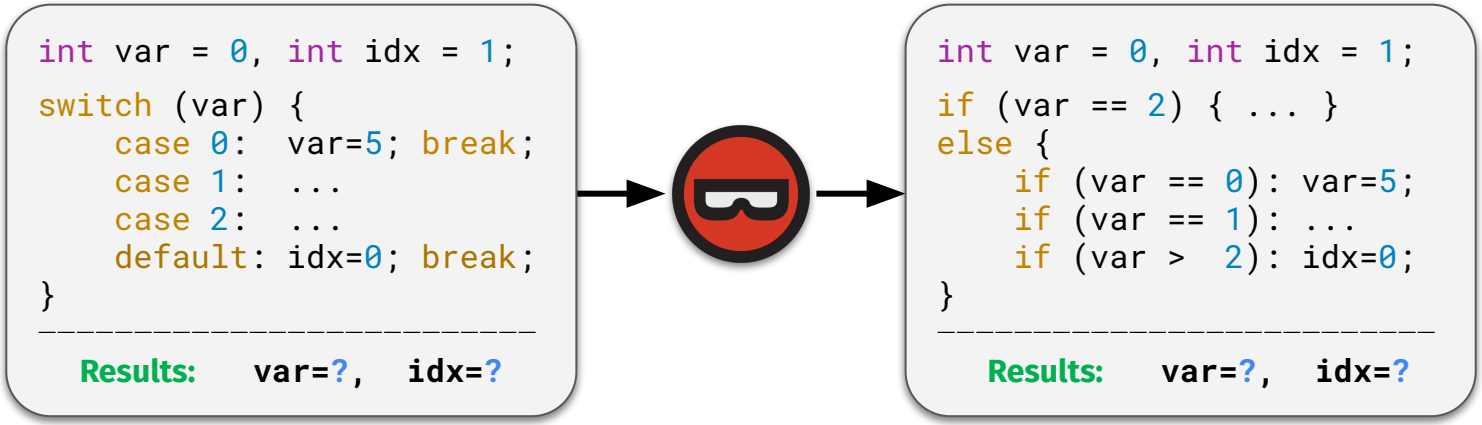






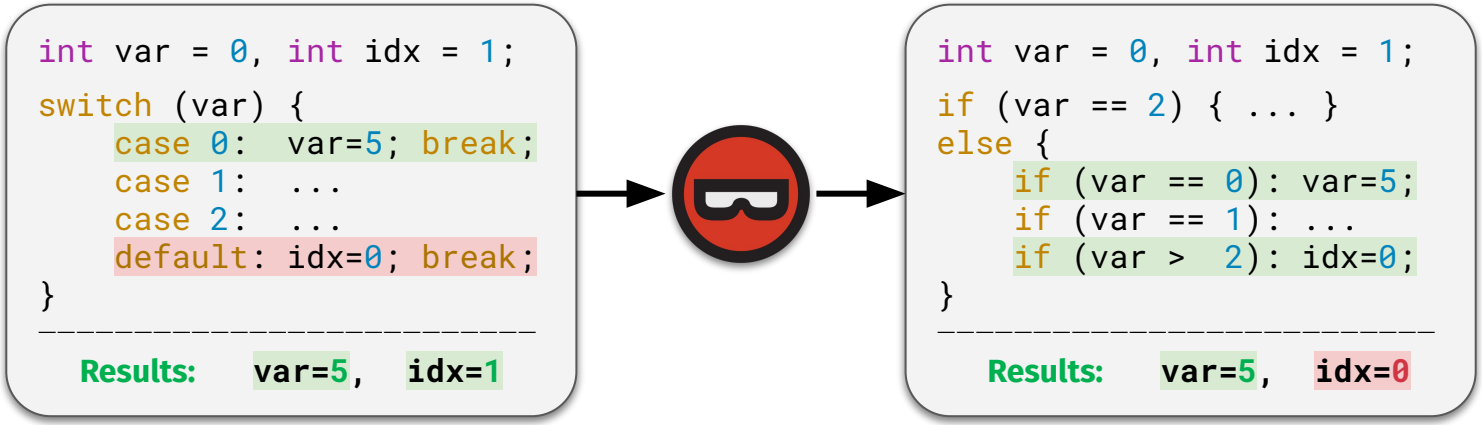
Case Studies: Bin2Wrong-found Bugs

- Bin2Wrong found a **critical bug** in commercial decompiler **Binary Ninja**



Case Studies: Bin2Wrong-found Bugs

- Bin2Wrong found a **critical bug** in commercial decompiler **Binary Ninja**



Case Studies: Bin2Wrong-found Bugs

- Bin2Wrong found a **critical bug** in commercial decompiler **Binary Ninja**
  - Spurred **an overhaul** of Binary Ninja's internal control flow recovery processes

binary.ninja/2024/06/19/restructuring-the-decompiler.html

- **Takeaways:** Bin2Wrong **exposes critical decompiler bugs missed by others**

Conclusion: Why Bin2Wrong?

- **Decompiler errors** create **downstream failures**
  - Hence, **testing decompilers** is critical to fixing them
- Prior fuzzers **fail to thoroughly test** decompilers
  - Only partial coverage of source/compilation factors
  - Vast majority of binary diversity thus left unexplored

Conclusion: Why Bin2Wrong?

- **Decompiler errors** create **downstream failures**
  - Hence, **testing decompilers** is critical to fixing them
- Prior fuzzers **fail to thoroughly test** decompilers
  - Only partial coverage of source/compilation factors
  - Vast majority of binary diversity thus left unexplored
- Our solution: **Bin2Wrong**
  - Unified mutation **coalescing source *and* compilation**
  - Support for **6 compilers**, **5,183 optimizations**, all major **executable formats**, and **virtually all C code constructs**
  - **Outcome:** systematic decompiler fuzzer exploring all of these individual dimensions—and **combinations thereof**

Key Results:

6–16× higher binary diversity,

16–32% higher code coverage,

48 new errors, **30** confirmed

Thank you!

github.com/FuturesLab/Bin2Wrong

Contact:
zao.yang@utah.edu
@yangzaocn
@zaoyang.bsky.social

FUTURES^3
LAB FUTURE TECHNOLOGY FOR USABLE, RELIABLE, & EFFICIENT SECURITY OF SOFTWARE & SYSTEMS