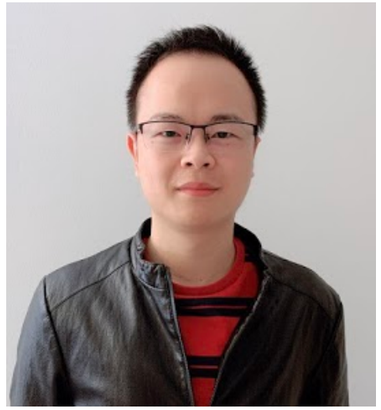
black hat ASIA 2020
OCTOBER 1-2, 2020
BRIEFINGS

Demystify Today's Binary Disassembly and How Modern ABI Makes it Easier

Who Are We

Chengbin Pang
- Visiting Scholar, Stevens Institute of Technology
- PhD Student, Nanjing University

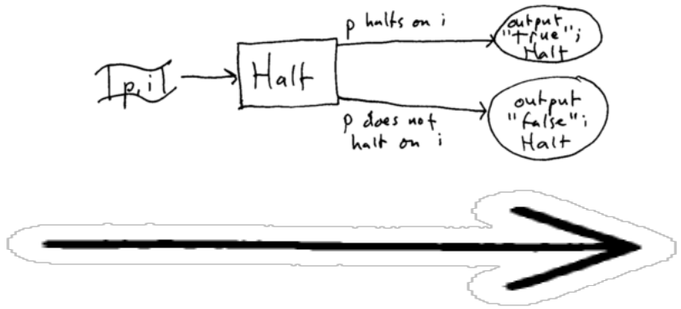


Jun Xu
- Assistant Professor, Stevens Institute of Technology

Eric Koskinen
- Assistant Professor, Stevens Institute of Technology

Begin of Our Story

X86/X64 binaries



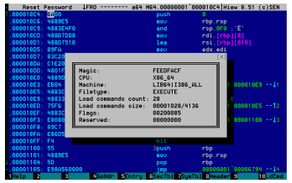
Temporal properties

Image source:
[1] https://medium.com/@bfil/how-code-becomes-legacy-2e6b58e839eb
[2] https://armacad.info/winter-school-engineering-and-computer-science-on-formal-verification-17-21-december-2017-israel

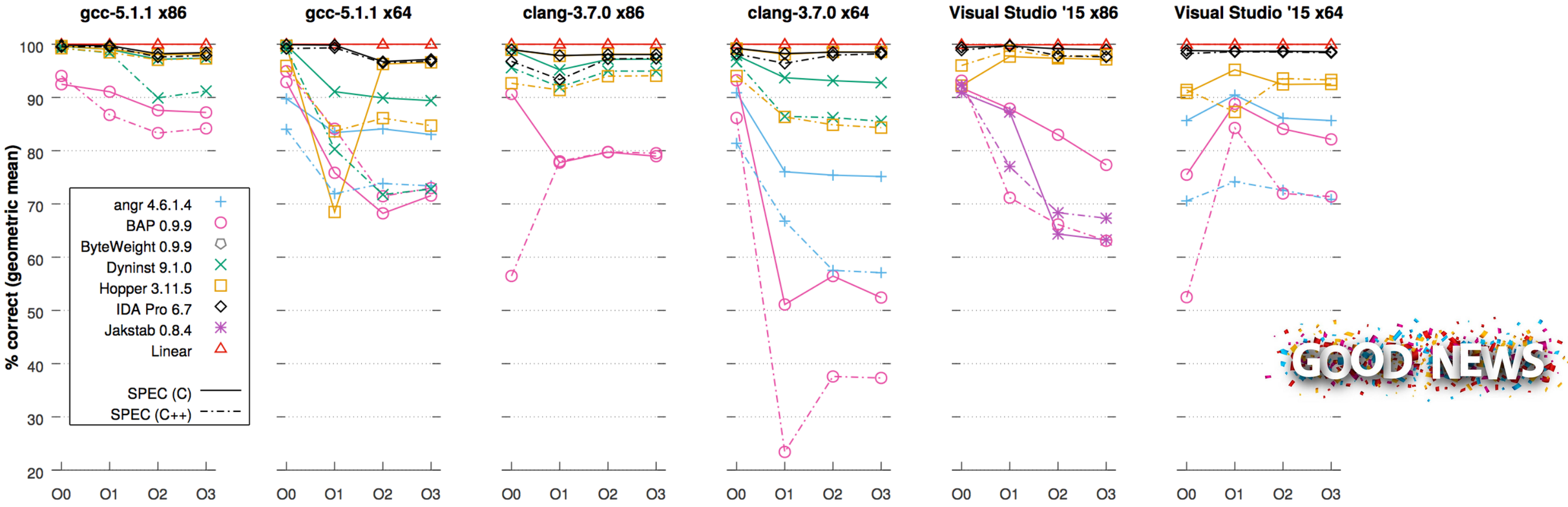
OK, Let's Disassemble the Code First

But...

12 GNU Binutils Tools



Many Disassemblers are *Super Good*



(a) Correctly disassembled instructions.

Data source:
[1] https://www.usenix.org/system/files/conference/usenixsecurity16/sec16_paper_andriesse.pdf

But Not Good Enough for Verification

```
1 ; wrong decoding
2 640069: add [%rax],%eax
3 64006b: add [%rax],%cl
4 ...
5 640126: call unwind
6 ; non-return
```

Wrong Instructions

```
<autohelperowl_defendpat156>:
480970:  push   %rbp
480971:  push   %r15
480973:  push   %r14
480975:  push   %rbx
480976:  push   %rax
```

Dyninst

Wrong Functions

```
which){
  case 't': ...
  ...
  default:
     undefined();
}
```

```
1 ; no restriction on %al
2 sub $0x64,%eax
3 movzbl %al,%eax
4 movslq (%r10,%rax,4),%rax
5 add %r10,%rax
6 jmpq *%rax
```

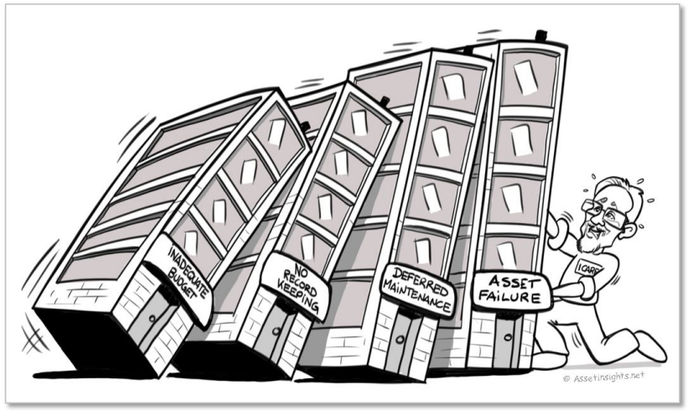
Wrong Pointers

```
mov 0x6ab8a0, %esi
mov %rbx, %rdi
```

```
; data;
0x6ab8a0: 69 00 64 00
...
```

```
; wrong decoding
640069: add [%rax],%ea
64006b: add [%rax],%cl
...
640126: call unwind
; non-return
```

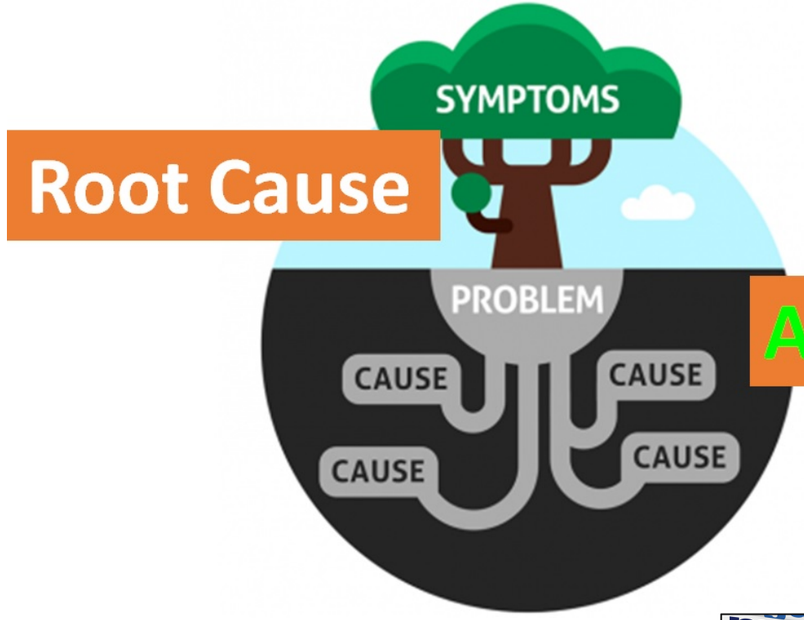
Wrong Control Flows



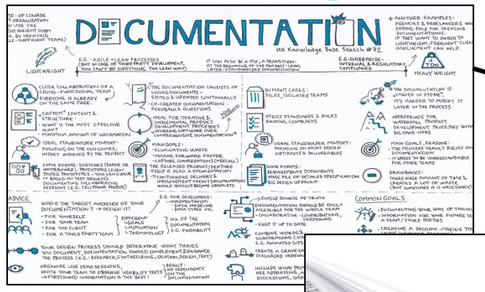
Errors from disassembly will propagate all the way to the verification results.

Image source:
[1] https://assetinsights.net/Glossary/G_Cascading_Effect.html

Let's Figure Out Why Errors and Fix Them



Analysis



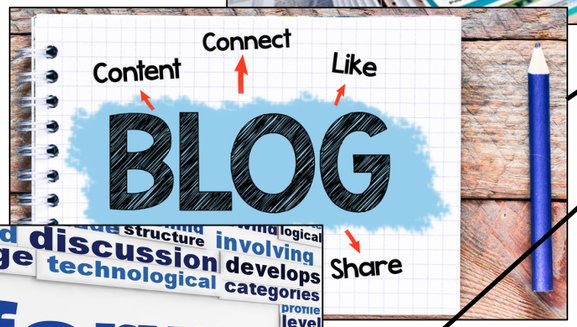


Image source:
[1] https://www.foodsafety-experts.com/food-safety/powerful-root-cause-analysis

Fine, Let's Read Their Source Code

- ONJDUMP
- UROBOROS
- PSI

12 GNU Binutils Tools

- RADARE@

- BAP

- GHIDRA

- MCSEMA

- ANGR

- DYNINST

Now, We Know Everything About The Disassemblers

15 principled algorithms
(correctness guaranteed)

29 heuristics
(sorry, no guarantee)

https://arxiv.org/abs/2007.14266

Alg.		*Algorithms & Heuristics*	*Goals*	*Tools*
Disassembly	Linear Sweep	① Start from code addresses with symbols	Code accuracy	OBJDUMP, PSI, UROBOROS
		❶ Continuous scanning for instructions	Code coverage	OBJDUMP, PSI, UROBOROS
		❷ Skip bad opcodes	Code accuracy	OBJDUMP
		❸ Replace padding and re-disassembly	Code accuracy	PSI
		❹ Exclude code around errors	Code accuracy	UROBOROS
	Recursive Descent	② Follow control flow to do disassembly	Code accuracy	DYNINST, GHIDRA, ANGR, BAP, RADARE2
		③ Start from program entry, `main`, and symbols	Code accuracy	DYNINST, GHIDRA, ANGR, BAP, RADARE2
		❺ Function entry matching	Code coverage	DYNINST, GHIDRA, ANGR, BAP, RADARE2
		❻ Linear sweep code gaps	Code coverage	ANGR
		❼ Disassembly from targets of xrefs	Code coverage	GHIDRA, RADARE2
Symbolization	Xrefs	④ Exclude data units that are floating points	Xref accuracy	ANGR
		❽ Brute force operands and data units	Xref coverage	UROBOROS, MCSEMA, GHIDRA, ANGR
		❾ Pointers in data have machine size	Xref accuracy	UROBOROS, MCSEMA, GHIDRA, ANGR
		❿ Alignment of pointers in data	Xref accuracy	UROBOROS, MCSEMA, GHIDRA
		⓫ Pointers in data or referenced by other xrefs can be non-aligned	Xref coverage	GHIDRA, ANGR
		⓬ References to code can only point to function entries	Xref accuracy	GHIDRA
		⓭ Enlarge boundaries of data regions	Xref coverage	GHIDRA, ANGR
		⓮ Address tables have minimal size of 2	Xref accuracy	GHIDRA
		⓯ Exclude pointers that may overlap with a string	Xref accuracy	MCSEMA, GHIDRA
		⓰ While scanning data regions, use step-length based on type inference	Xref accuracy	ANGR
Function Entry	MAIN Function	⑤ Identify `main` based on arguments to `__libc_start_main`	Func coverage	ANGR, BAP
		⓱ Identify `main` using patterns in `_start`/`__scrt_common_main_seh`	Func coverage	DYNINST, RADARE2
	General Function	⑥ Identify function entries based on symbols	Func coverage	DYNINST, GHIDRA, ANGR, BAP, RADARE2
		⑦ Identify function entries based on exception information	Func coverage	GHIDRA
		⑧ Identify function entries based on targets of direct calls	Func coverage	DYNINST, GHIDRA, ANGR, BAP, RADARE2
		⑨ Identify function entries by resolving indirect calls	Func coverage	GHIDRA, ANGR
		⓲ Identify function entries based on prologues/decision-tree	Func coverage	DYNINST, GHIDRA, ANGR, BAP, RADARE2
		⓳ Consider begins of code discovered by linear scan as function entries	Func coverage	ANGR
CFG	Indirect Jump	⑩ Use VSA to resolve jump table targets	CFG accuracy	DYNINST, GHIDRA, ANGR
		⓴ Follow patterns to determine jump tables	CFG accuracy	DYNINST, GHIDRA, RADARE2
		㉑ Discard jump tables with index bound larger than a threshold	CFG accuracy	GHIDRA, ANGR, RADARE2
		㉒ Restrict the depth of slice for VSA	Efficiency	DYNINST, ANGR
	Indirect Call	⑪ Identify targets based on constant propagation	CFG coverage	GHIDRA, ANGR
	Tail Call	⑫ Consider a jump to the start of another function as a tail call	CFG accuracy	DYNINST, ANGR
		㉓ Determine tail call based on distance between the jump and its target	CFG accuracy	RADARE2
		㉔ A tail call and its target cross multiple functions	CFG accuracy	GHIDRA
		㉕ Tail calls cannot be conditional jumps	CFG accuracy	GHIDRA, ANGR
		㉖ A tail call tears down its stack	CFG accuracy	DYNINST, ANGR
		㉗ A tail call does not jump to the middle of a function	CFG accuracy	ANGR
		㉘ Target of a tail call cannot be target of any conditional jumps	CFG accuracy	ANGR
	Non-returning Function	⑬ Identify system calls or library functions that are known non-returning	CFG accuracy	DYNINST, GHIDRA, ANGR, BAP, RADARE2
		⑭ Identify functions with no `ret` and no tail calls that return	CFG accuracy	DYNINST, ANGR, RADARE2
		⑮ Identify functions that always call non-returning functions	CFG accuracy	BAP
		㉙ Detect non-returning functions based on fall-through after the call-sites	CFG accuracy	GHIDRA

Examples of Error-Prone Heuristics

- ❑ Linear scanning code gaps to recover instructions
- ❑ Considering integers that point to legitimate locations as pointers
- ❑ Finding functions in code gaps with common signatures of prologues
- ❑ Detecting jump tables based on common patterns

But Which Strategies Make What Errors?

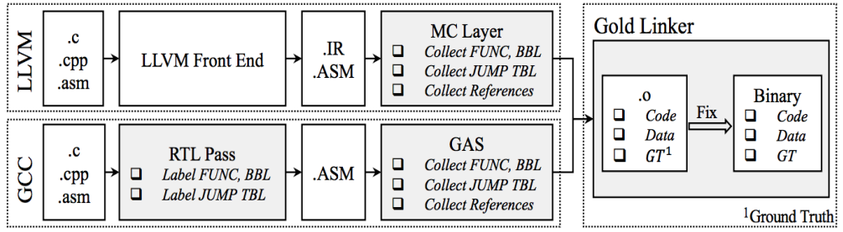
Well, Let's Run An Evaluation

BENCHMARK

Type	Name	Programs/Binaries	
		Linux	Windows
Benchmark	SPEC CPU2006	30 / 546	15 / 120
Utilities	Unzip-6.0 Coreutils-8.30 7-zip-19 Findutils-4.4 Binutils-2.26 Tiff-4.0	125 / 2500	26 / 196
Clients	Openssl-1.1.0l Putty-0.73 D8-6.4 Filezilla-3.44.2 Busybox-1.31 Protobuf-c-1 ZSH-5.7.1 VIM-8.1 XML2-2.9.8 Openssh-8.0 Git-2.23	13 / 154	13 / 104
Servers	Lighttpd-1.4.54 Mysqld-5.7.27 Nginx-1.15.0 SQLite-3.32.0	3 / 49	2 / 16
Libraries	Glibc-2.27 libpcap-1.9.0 libv8-6.4 libtiff-4.0.10 libxml2-2.9.8 libsqlite-3.32.0 libprotobuf-c-1.3.2	6 / 79	3 / 24
Total		177 / 3328	59 / 460

About 4K binaries built from scratch

groundtruth



Tracing the compilation step-by-step

Test each strategy separately

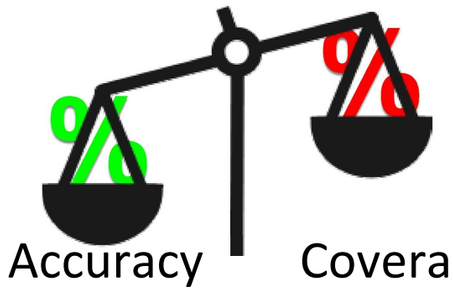
- ✓ Measure its contribution
- ✓ Evaluate its accuracy
- ✓ Understand its errors

No More Excuses of Reusing Existing Disassemblers

- ❑ Heuristics are heavily used
  - ➢ Nearly 50% the instructions are recovered with heuristics
  - ➢ About 20% of the functions are detected by heuristics
  - ➢ …

- ❑ All Heuristics Introduce Errors
  - ➢ Which Heuristics Cause Which Errors Are Understood (https://arxiv.org/abs/2007.14266)

- ❑ Fixing The Errors Will Reduce the Coverage!!!



Did Our Study Kill Our Research? NO!

Findings

- ✓ System-v x64 ABI mandates a call frame for each function for exception handling.
- ✓ The call frame gives the start address of the corresponding function
- ✓ Call frames persist even in stripped, commercial binaries

skype-8.55.0.141
trillian-6.1.0.5
opera-65.0.3467.69
yandex-browser-19.12.3
SpiderOakONE-7.5.01
slack-4.2.0
rainlendar2-2.15.2
sublime-3211
netease-cloud-music-1.2.1
wps-11.1.0.8865
wpp-11.1.0.8865
wpspdf-11.1.0.8865
wpsoffice-11.1.0.8865
ida64-7.2
zoom-7.19.2020
binaryninja-1.2
FoxitReader-4.4.0911

SVABI

System V Application Binary Interface

Exploiting Call Frames to Do Binary Disassembly

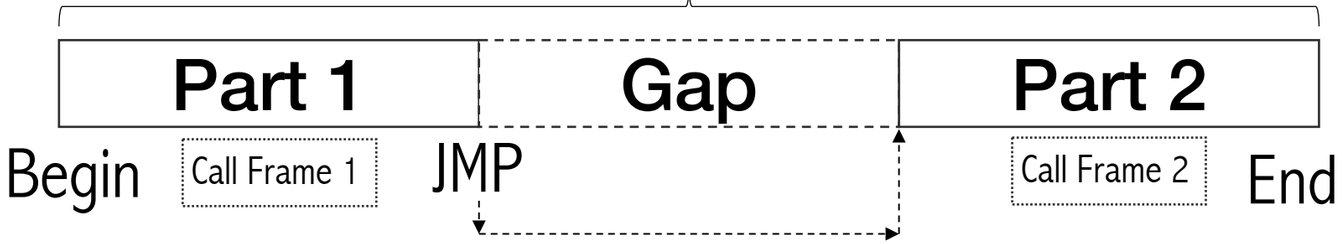
- ❑ Extract function starts from call frames

- ❑ Recursive disassembly from each function start
- ✓ Recover instructions, functions, and control flows
- ✓ Avoid using any heuristics

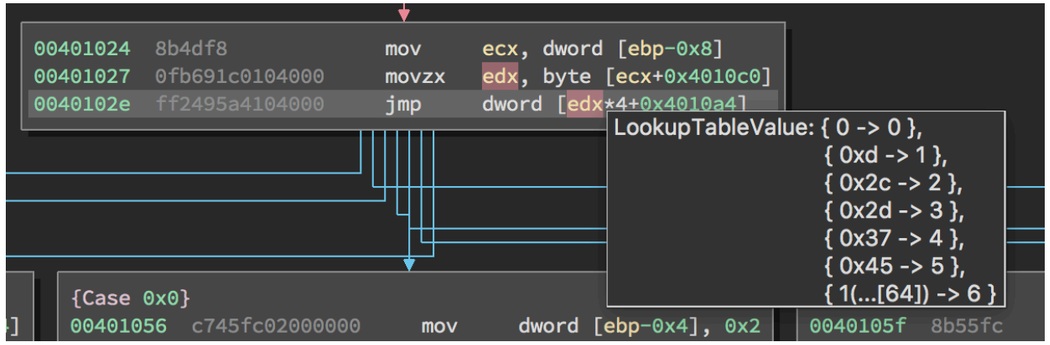
- ❑ Job done!

Wait, Reality is Always More Complicated

Non-continuous Functions



Jump Tables



Solution

- ❑ Checking the JMP is not a tail call
- ✓ Stack height at the jump site is not 0 (or)
- ✓ Part 2 does not meet calling conventions (or)
- ✓ Part 2 is not referenced in other places

Solution

- ❑ Intra-procedural value set analysis to decide targets of jump tables

Now, The Results are Verification Friendly

OPT	ANGR		DYNINST		GHIDRA		BAP		RADARE2		NUCLEUS		BYTEWEIGHT		IDA PRO		BINARY NINJA		Symbol		FETCH	
	P	R	P	R	P	R	P	R	P	R	P	R	P	R	P	R	P	R	P	R	P	R
O2	83.66	95.25	90.85	63.71	97.13	99.75	75.29	54.20	97.47	50.89	97.79	90.21	96.38	96.51	99.92	72.67	97.72	92.31	97.29	99.99	99.96	99.79
O3	83.56	95.63	90.41	62.09	96.38	99.78	73.20	55.55	97.18	50.61	97.42	91.98	97.91	97.75	99.92	75.41	97.63	94.03	96.52	99.99	99.96	99.78
Os	92.61	96.42	92.76	69.65	99.84	99.91	86.39	66.54	98.84	69.66	97.64	91.23	97.83	96.05	99.98	81.94	98.75	98.81	99.98	99.99	99.99	99.96
Of	81.89	95.16	90.71	62.73	96.56	99.77	73.58	56.30	98.14	50.77	96.00	91.03	97.02	96.90	99.95	76.19	97.75	94.22	96.67	99.99	99.91	99.81
Avg.	**85.43**	**95.62**	**91.18**	**64.55**	**97.48**	**99.80**	**77.12**	**58.15**	**97.92**	**55.23**	**97.21**	**91.11**	**97.29**	**96.80**	**99.94**	**85.28**	**97.84**	**94.84**	**97.62**	**99.99**	**99.96**	**99.84**

Our Tool

Example: Results of Function Detection (P = Precision; R = Recall)

The missing cases or errors are harmless!

Remaining Problems for Tomorrow

- ❑ Detection of pointers from binary code
- ❑ Handling binaries other than x64 system-v binaries
- ❑ Engineering issues in the tools we are running

Concluding Remarks

- ❑ We fully demystify the strategies used by existing disassemblers
  - ✓ Deepen the understanding of the community about binary disassembly
  - ✓ Help users gain awareness of the risks of existing disassemblers
- ❑ We unveil the performance and pitfalls of each strategy of each disassembler
  - ✓ Bring insights for the community to further improve binary disassembly
  - ✓ Help users select the correct tools to best satisfy their demands
- ❑ We exploit call frames to achieve verification-friendly binary disassembly
  - ✓ Provide an alternative approach for users that have high requirements of correctness
  - ✓ Bring insights of leveraging extra sources of information to aid disassembly

Software and Data Release

- ❑ Findings of our study
  - ✓ https://arxiv.org/abs/2007.14266

- ❑ Data & code of our study (publicly available)
  - ✓ https://github.com/junxzm1990/x86-sok

- ❑ Code of our new disassembler (private, available on request)
  - ✓ https://github.com/ruotongyu/eh_frame

Thank you for listening