A Broad Comparative Evaluation of x86-64 Binary Rewriters

10 Minute Presentation

Eric Schulte
schulte.eric@gmail.com

Michael Brown
Trail of Bits
michael.brown@trailofbits.com

Vlad Folts
GrammaTech
vfolts@grammatech.com

2022-08-08

Contents

Significant and active research area

[Romer et al.(1997), Schwarz et al.(2001), Van Put et al.(2005), Tilevich and Smaragdakis(2005), Laurenzano et al.(2010), Wartell et al.(2012), Zhang and Sekar(2013), Smithson et al.(2013), Zhang et al.(2014), Hiser et al.(2014), Dinaburg and Ruef(2014), Wang et al.(2015), Wang et al.(2017), Di Federico et al.(2017), Qian et al.(2019), Dinesh(2019), Williams-King et al.(2020), Duck et al.(2020), Kiaei et al.(2020), Flores-Montoya and Schulte(2020), Galois(2021), Microsoft(2022), Software Engineering Institute(2022)]

Use Cases

Optimization

- Link Time Optimization (LTO)

Instrumentation

- Greybox fuzz testing
- Dynamic analysis

Patching

- Bug repair
- Capture the flag

Configuration

- Hard-coded configuration
- Deep parameter customization

Hardening

- Inline memory protection
- Inline control-flow protection

Debloating

- Reduce attack surface

Goals of this work

1. Illuminate classes of binary rewriter utility

Practical Rewriters	Immature Rewriters	Speculative Rewriters
Reaching practical utility	Significant engineering challenges remain	Significant open research challenges remain

2. Post questions and suggest directions for the research community

Related Work

Survey

[Wenzl et al.(2019)]
Categorize binary rewriting by:

- Use case
- Analysis technique
- Code transformation method
- Code generation method

Disassembler Evaluations

[Andriesse et al.(2016), Meng and Miller(2016), Li et al.(2020), Pang et al.(2021)]
Document disassembler:

- Approach
- Challenges
- Trade-offs
- Shortcomings

Binary Analysis Evaluations

[Woodruff et al.(2021), Xu et al.(2019), Dasgupta et al.(2020)]
Evaluation of binary analysis, often focus on depth over breadth.

Types of Binary Rewriters

Trampoline

e9patch — National University of Singapore [Duck et al.(2020)]

LLVM Rewriting

SecondWrite — SecondWrite LLC

McSema — Trail of Bits

mctoll — Microsoft [Microsoft(2022)]

Rev.Ng — The rev.ng Srls Company [Di Federico et al.(2017)]

reopt — Galois [Galois(2021)]

Direct Rewriting

Zipr — University of Virginia [Hiser et al.(2014)]

Egalito — Columbia University [Williams-King et al.(2020)]

multiverse — UT Dallas [Bauman et al.(2018)]

Reassemblable Disassemblers

Uroboros — Penn State

Ramblr — UCSB [Wang et al.(2017)]

DDisasm — GrammaTech [Flores-Montoya and Schulte(2020)]

Retrowrite — Purdue [Dinesh(2019)]

Benchmark Programs

Table: Benchmark programs used in this evaluation

Program	KSLOC	Description	Program	KSLOC	Description
anope	65.4	IRC Services	openvpn	89.3	VPN Client
asterisk	771.2	Comm. Framework	pidgin	259.4	Chat Client
bind	376.1	DNS System	pks	40.8	Public Key Server
bitcoind	229.9	Bitcoin Client	poppler	188.2	PDF Reader
dnsmasq	34.7	Network Services	postfix	135.0	Mail Server
filezilla	176.3	FTP Client and Server	proftpd	544.2	FTP Server
gnome-calculator	0.3	Calculator	qmail	14.7	Message Transfer Agent
leafnode	12.9	NNTP Proxy	redis	14.7	In-memory Data Store
Libreoffice	5,090.9	Office Suite	samba	1,864.0	Windows Interoperability
libzmq	62.4	Messaging Library	sendmail	104.5	Mail Server
lighttpd	89.7	Web Server	sipwitch	17.1	VoIP Server
memcached	33.5	In-memory Object Cache	snort	344.9	Intrusion Prevention
monerod	394.8	Blockchain Daemon	sqlite	292.4	SQL Server
mosh	12.9	Mobile Shell	squid	212.8	Caching Web Proxy
mysql	3,331.7	SQL Server	unrealircd	91.0	IRC Server
nginx	170.6	Web Server	vi/vim	394.1	Text Editor
ssh	127.4	SSH Client and Server	zip	54.4	Compression Utility

Variant Compilation

Table: Variant configuration options

Compiler	Flags	Relocation (Position-)	Symbols	Operating Systems
clang gcc icx	O0 O1 O2 O3 Os Ofast	Independent Dependent	Present Stripped	Ubuntu 16.04[1] Ubuntu 20.04
OLLVM	fla sub bcf[2]	Independent Dependent	Present Stripped	Ubuntu 20.04

[1]Some binaries could not be built on this OS due to unavailable dependencies.
[2]Probability variable set to always insert (100%)

Evaluation

Rewrite binaries with Null and AFL transforms

Artifact Metrics

(Rewritten Artifact)

- **IR** Produce an IR
- **EXE** Produce an executable

Functional Metrics

(Rewritten Artifact)

- Smoke Tests
- Functional Tests

Non-Functional Metrics

(Rewritten Artifact)

- File size
- Runtime
- Memory Consumption

Non-Functional Metrics

(Rewriter)

- Runtime
- Memory Consumption

Aggregate Rewriting Success

Table: Percentage of successfully rewritten x86-64 Linux binaries.

Tool	IR	Null EXE	Null Func.	AFL EXE	AFL Func.
ddisasm	98.14%	88.87%	85.91%	90.31%	70.15%
e9patch		100.00%	78.34%	100.00%	36.24%
egalito		98.50%	29.39%	74.55%	0
mctoll	0.89%	0.89%	0.89%	0.89%	0.89%
multiverse		26.31%	10.82%	0	0
reopt	89.92%	76.43%	33.91%	0	0
retrowrite	24.43%	9.98%	9.24%	9.86%	7.59%
revng		26.46%	23.50%	0	0
uroboros	10.88%	6.45%	2.87%	6.27%	0
zipr		100.00%	100.00%	80.98%	48.11%

Aggregate Rewriting Success

Table: Percentage of successfully rewritten x86-64 Linux binaries.

Tool	IR	Null EXE	Null Func.	AFL EXE	AFL Func.
ddisasm	98.14%	88.87%	85.91%	90.31%	70.15%
e9patch		100.00%	78.34%	100.00%	36.24%
egalito		98.50%	29.39%	74.55%	0
mctoll	0.89%	0.89%	0.89%	0.89%	0.89%
multiverse		26.31%	10.82%	0	0
reopt	89.92%	76.43%	33.91%	0	0
retrowrite	24.43%	9.98%	9.24%	9.86%	7.59%
revng		26.46%	23.50%	0	0
uroboros	10.88%	6.45%	2.87%	6.27%	0
zipr		100.00%	100.00%	80.98%	48.11%

The cp command would get 100% in “EXE” and “Null Func.”

Aggregate Rewriting Success

Table: Percentage of successfully rewritten x86-64 Linux binaries.

Tool	IR	Null EXE	Null Func.	AFL EXE	AFL Func.
ddisasm	98.14%	88.87%	85.91%	90.31%	70.15%
e9patch		100.00%	78.34%	100.00%	36.24%
egalito		98.50%	29.39%	74.55%	0
mctoll	0.89%	0.89%	0.89%	0.89%	0.89%
multiverse		26.31%	10.82%	0	0
reopt	89.92%	76.43%	33.91%	0	0
retrowrite	24.43%	9.98%	9.24%	9.86%	7.59%
revng		26.46%	23.50%	0	0
uroboros	10.88%	6.45%	2.87%	6.27%	0
zipr		100.00%	100.00%	80.98%	48.11%

Uroboros only works with `no-pie` binaries.

Aggregate Rewriting Success

Table: Percentage of successfully rewritten x86-64 Linux binaries.

Tool	IR	Null EXE	Null Func.	AFL EXE	AFL Func.
ddisasm	98.14%	88.87%	85.91%	90.31%	70.15%
e9patch		100.00%	78.34%	100.00%	36.24%
egalito		98.50%	29.39%	74.55%	0
mctoll	0.89%	0.89%	0.89%	0.89%	0.89%
multiverse		26.31%	10.82%	0	0
reopt	89.92%	76.43%	33.91%	0	0
retrowrite	24.43%	9.98%	9.24%	9.86%	7.59%
revng		26.46%	23.50%	0	0
uroboros	10.88%	6.45%	2.87%	6.27%	0
zipr		100.00%	100.00%	80.98%	48.11%

E9patch, egalito, and retrowrite don't work with `no-pie` binaries.

Aggregate Rewriting Success

Table: Percentage of successfully rewritten x86-64 Linux binaries.

Tool	IR	Null EXE	Null Func.	AFL EXE	AFL Func.
ddisasm	98.14%	88.87%	85.91%	90.31%	70.15%
e9patch		100.00%	78.34%	100.00%	36.24%
egalito		98.50%	29.39%	74.55%	0
mctoll	0.89%	0.89%	0.89%	0.89%	0.89%
multiverse		26.31%	10.82%	0	0
reopt	89.92%	76.43%	33.91%	0	0
retrowrite	24.43%	9.98%	9.24%	9.86%	7.59%
revng		26.46%	23.50%	0	0
uroboros	10.88%	6.45%	2.87%	6.27%	0
zipr		100.00%	100.00%	80.98%	48.11%

McToll requires prototypes for all external function.

Aggregate Rewriting Success

Table: Percentage of successfully rewritten x86-64 Linux binaries.

Tool	IR	Null EXE	Null Func.	AFL EXE	AFL Func.
ddisasm	98.14%	88.87%	85.91%	90.31%	70.15%
e9patch		100.00%	78.34%	100.00%	36.24%
egalito		98.50%	29.39%	74.55%	0
mctoll	0.89%	0.89%	0.89%	0.89%	0.89%
multiverse		26.31%	10.82%	0	0
reopt	89.92%	76.43%	33.91%	0	0
retrowrite	24.43%	9.98%	9.24%	9.86%	7.59%
revng		26.46%	23.50%	0	0
uroboros	10.88%	6.45%	2.87%	6.27%	0
zipr		100.00%	100.00%	80.98%	48.11%

Multiverse only supports Ubuntu 16 binaries (not binaries built on newer Ubuntus).

Binary Rewriter Runtime and Memory High-Water Mark

Table: Runtime.

Tool	Runtime (seconds)
ddisasm	72.81
e9patch	2.74
egalito	454.40
mctoll	0.00
multiverse	1195.72
reopt	169.89
retrowrite	114.57
revng	703.74
uroboros	19.17
zipr	233.61

Table: Memory high-water mark.

Tool	Memory (gigabytes)
ddisasm	0.509
e9patch	0.105
egalito	10.433
mctoll	0.001
multiverse	0.688
reopt	4.062
retrowrite	1.967
revng	2.244
uroboros	0.094
zipr	1.016

Functionality and Performance against Full Test Suite

Table: Functionality.

Tool	lighttpd	nginx	redis
original	30/30	60/60	26/30
ddisasm	0/30	60/60	26/30
e9patch	30/30	60/60	26/30
egalito	18/18	18/18	10/10
multiverse	0/0	0/0	0/0
reopt	2/19	4/60	2/8
retrowrite	0/9	16/26	0/0
revng	0/0	0/0	0/0
uroboros	0/0	0/0	0/0
zipr	28/30	58/58	22/30

Table: Performance.

Tool	Runtime	Memory High-water
ddisasm	109%	100%
e9patch	120%	99%
egalito	104%	100%
reopt	1325%	51937%
retrowrite	104%	100%
zipr	103%	102%

Size of Rewritten Binaries

Table: Percent rewritten binary size increase.

Tool	Percent size change
ddisasm	91.90%
e9patch	114.45%
egalito	169.17%
mctoll	128.22%
multiverse	870.71%
reopt	99.61%
retrowrite	83.78%
revng	1581.95%
uroboros	148.97%
zipr	140.05%

Size of Rewritten Binaries by Section

As measured by Bloaty

Table: Percent rewritten binary size increase by section.

Section	ddisasm	e9patch	egalito	mctoll	multiverse	reopt	retrowrite	revng	uroboros	zipr
.got.plt	100.08%	100.00%	100.67%	89.60%	100.00%	100.29%	100.00%	NA	99.24%	NA
.data	102.94%	100.00%	100.24%	NA	99.99%	103.29%	100.42%	1014.94%	101.87%	NA
.dynamic	97.48%	100.00%	66.53%	97.70%	100.00%	99.82%	101.83%	NA	99.91%	NA
.rela.dyn	87.27%	114.67%	823.91%	54.35%	100.00%	102.65%	97.30%	355.43%	91.19%	NA
.strtab	104.90%	100.00%	77.54%	95.27%	99.26%	76.69%	105.45%	407.41%	485.39%	NA
.dynsym	75.44%	100.00%	101.86%	75.24%	100.00%	110.20%	77.96%	821.27%	99.43%	NA
.dynstr	72.78%	100.00%	101.42%	73.18%	99.99%	104.62%	74.56%	1075.44%	99.75%	NA
.symtab	118.87%	100.00%	84.73%	93.93%	100.00%	107.44%	95.09%	270.77%	1675.22%	NA
.eh_frame_hdr	103.76%	100.00%	NA	110.43%	100.00%	106.36%	93.36%	25.33%	108.05%	NA
.plt	100.05%	100.00%	100.45%	89.60%	100.00%	100.28%	99.81%	990.99%	99.75%	NA
.rela.plt	99.93%	100.00%	99.35%	NA	100.00%	100.30%	99.81%	757.03%	100.00%	NA
.eh_frame	109.75%	100.00%	NA	101.88%	100.00%	125.52%	15.14%	520.42%	111.34%	NA
[Prg. Hdrs.]	96.37%	100.00%	94.29%	105.53%	144.44%	94.00%	104.79%	NA	97.77%	119.81%
[Sct. Hdrs.]	97.66%	100.00%	71.70%	94.93%	116.13%	103.62%	92.86%	148.37%	97.57%	98.55%
.rodata	100.10%	100.00%	100.43%	NA	100.00%	99.96%	100.06%	480.80%	100.03%	NA
.text	146.85%	100.00%	161.64%	100.89%	100.00%	240.13%	120.45%	3817.02%	110.17%	NA
[Unmapped]	128.93%	13162.97%	670.47%	350.40%	2285.56%	181.98%	225.89%	22384.87%	165.31%	10.40%

Size of Rewritten Binaries by Section

As measured by Bloaty

Table: Percent rewritten binary size increase by section.

Section	ddisasm	e9patch	egalito	mctoll	multiverse	reopt	retrowrite	revng	uroboros	zipr
.got.plt	100.08%	100.00%	100.67%	89.60%	100.00%	100.29%	100.00%	NA	99.24%	NA
.data	102.94%	100.00%	100.24%	NA	99.99%	103.29%	100.42%	1014.94%	101.87%	NA
.dynamic	97.48%	100.00%	66.53%	97.70%	100.00%	99.82%	101.83%	NA	99.91%	NA
.rela.dyn	87.27%	114.67%	823.91%	54.35%	100.00%	102.65%	97.30%	355.43%	91.19%	NA
.strtab	104.90%	100.00%	77.54%	95.27%	99.26%	76.69%	105.45%	407.41%	485.39%	NA
.dynsym	75.44%	100.00%	101.86%	75.24%	100.00%	110.20%	77.96%	821.27%	99.43%	NA
.dynstr	72.78%	100.00%	101.42%	73.18%	99.99%	104.62%	74.56%	1075.44%	99.75%	NA
.symtab	118.87%	100.00%	84.73%	93.93%	100.00%	107.44%	95.09%	270.77%	1675.22%	NA
.eh_frame_hdr	103.76%	100.00%	NA	110.43%	100.00%	106.36%	93.36%	25.33%	108.05%	NA
.plt	100.05%	100.00%	100.45%	89.60%	100.00%	100.28%	99.81%	990.99%	99.75%	NA
.rela.plt	99.93%	100.00%	99.35%	NA	100.00%	100.30%	99.81%	757.03%	100.00%	NA
.eh_frame	109.75%	100.00%	NA	101.88%	100.00%	125.52%	15.14%	520.42%	111.34%	NA
[Prg. Hdrs.]	96.37%	100.00%	94.29%	105.53%	144.44%	94.00%	104.79%	NA	97.77%	119.81%
[Sct. Hdrs.]	97.66%	100.00%	71.70%	94.93%	116.13%	103.62%	92.86%	148.37%	97.57%	98.55%
.rodata	100.10%	100.00%	100.43%	NA	100.00%	99.96%	100.06%	480.80%	100.03%	NA
.text	146.85%	100.00%	161.64%	100.89%	100.00%	240.13%	120.45%	3817.02%	110.17%	NA
[Unmapped]	128.93%	13162.97%	670.47%	350.40%	2285.56%	181.98%	225.89%	22384.87%	165.31%	10.40%

Most rewriters add sections which aren't properly in the program header table.

Size of Rewritten Binaries by Section

As measured by Bloaty

Table: Percent rewritten binary size increase by section.

Section	ddisasm	e9patch	egalito	mctoll	multiverse	reopt	retrowrite	revng	uroboros	zipr
.got.plt	100.08%	100.00%	100.67%	89.60%	100.00%	100.29%	100.00%	NA	99.24%	NA
.data	102.94%	100.00%	100.24%	NA	99.99%	103.29%	100.42%	1014.94%	101.87%	NA
.dynamic	97.48%	100.00%	66.53%	97.70%	100.00%	99.82%	101.83%	NA	99.91%	NA
.rela.dyn	87.27%	114.67%	823.91%	54.35%	100.00%	102.65%	97.30%	355.43%	91.19%	NA
.strtab	104.90%	100.00%	77.54%	95.27%	99.26%	76.69%	105.45%	407.41%	485.39%	NA
.dynsym	75.44%	100.00%	101.86%	75.24%	100.00%	110.20%	77.96%	821.27%	99.43%	NA
.dynstr	72.78%	100.00%	101.42%	73.18%	99.99%	104.62%	74.56%	1075.44%	99.75%	NA
.symtab	118.87%	100.00%	84.73%	93.93%	100.00%	107.44%	95.09%	270.77%	1675.22%	NA
.eh_frame_hdr	103.76%	100.00%	NA	110.43%	100.00%	106.36%	93.36%	25.33%	108.05%	NA
.plt	100.05%	100.00%	100.45%	89.60%	100.00%	100.28%	99.81%	990.99%	99.75%	NA
.rela.plt	99.93%	100.00%	99.35%	NA	100.00%	100.30%	99.81%	757.03%	100.00%	NA
.eh_frame	109.75%	100.00%	NA	101.88%	100.00%	125.52%	15.14%	520.42%	111.34%	NA
[Prg. Hdrs.]	96.37%	100.00%	94.29%	105.53%	144.44%	94.00%	104.79%	NA	97.77%	119.81%
[Sct. Hdrs.]	97.66%	100.00%	71.70%	94.93%	116.13%	103.62%	92.86%	148.37%	97.57%	98.55%
.rodata	100.10%	100.00%	100.43%	NA	100.00%	99.96%	100.06%	480.80%	100.03%	NA
.text	146.85%	100.00%	161.64%	100.89%	100.00%	240.13%	120.45%	3817.02%	110.17%	NA
[Unmapped]	128.93%	13162.97%	670.47%	350.40%	2285.56%	181.98%	225.89%	22384.87%	165.31%	10.40%

Most LLVM rewriters increase text size due to reified stack and memory.

Size of Rewritten Binaries by Section

As measured by Bloaty

Table: Percent rewritten binary size increase by section.

Section	ddisasm	e9patch	egalito	mctoll	multiverse	reopt	retrowrite	revng	uroboros	zipr
.got.plt	100.08%	100.00%	100.67%	89.60%	100.00%	100.29%	100.00%	NA	99.24%	NA
.data	102.94%	100.00%	100.24%	NA	99.99%	103.29%	100.42%	1014.94%	101.87%	NA
.dynamic	97.48%	100.00%	66.53%	97.70%	100.00%	99.82%	101.83%	NA	99.91%	NA
.rela.dyn	87.27%	114.67%	823.91%	54.35%	100.00%	102.65%	97.30%	355.43%	91.19%	NA
.strtab	104.90%	100.00%	77.54%	95.27%	99.26%	76.69%	105.45%	407.41%	485.39%	NA
.dynsym	75.44%	100.00%	101.86%	75.24%	100.00%	110.20%	77.96%	821.27%	99.43%	NA
.dynstr	72.78%	100.00%	101.42%	73.18%	99.99%	104.62%	74.56%	1075.44%	99.75%	NA
.symtab	118.87%	100.00%	84.73%	93.93%	100.00%	107.44%	95.09%	270.77%	1675.22%	NA
.eh_frame_hdr	103.76%	100.00%	NA	110.43%	100.00%	106.36%	93.36%	25.33%	108.05%	NA
.plt	100.05%	100.00%	100.45%	89.60%	100.00%	100.28%	99.81%	990.99%	99.75%	NA
.rela.plt	99.93%	100.00%	99.35%	NA	100.00%	100.30%	99.81%	757.03%	100.00%	NA
.eh_frame	109.75%	100.00%	NA	101.88%	100.00%	125.52%	15.14%	520.42%	111.34%	NA
[Prg. Hdrs.]	96.37%	100.00%	94.29%	105.53%	144.44%	94.00%	104.79%	NA	97.77%	119.81%
[Sct. Hdrs.]	97.66%	100.00%	71.70%	94.93%	116.13%	103.62%	92.86%	148.37%	97.57%	98.55%
.rodata	100.10%	100.00%	100.43%	NA	100.00%	99.96%	100.06%	480.80%	100.03%	NA
.text	146.85%	100.00%	161.64%	100.89%	100.00%	240.13%	120.45%	3817.02%	110.17%	NA
[Unmapped]	128.93%	13162.97%	670.47%	350.40%	2285.56%	181.98%	225.89%	22384.87%	165.31%	10.40%

Trampoline rewriters don't change most section sizes at all.

Utility Classes of Binary Rewriters

Practical Rewriters	**Immature Rewriters**	**Speculative Rewriters**
e9patch, Zipr, DDisasm	Uroboros, Egalito, Retrowrite, multiverse	*LLVM Rewriters*: mctoll, ReOpt, revng

Challenges of LLVM Binary Rewriting

Binary type analysis

LLVM IR must be typed but binary type recovery is an open research challenge.

Workarounds

- Heap explicitly reified as a byte array
- Stack explicitly reified as a byte array
- Code added to maintain the reified stack
- 2 stacks and 2 heaps

Results

- Baroque, complex, slow, and brittle rewritten binaries
- Most LLVM passes don't apply

Questions for the community

1. Should anyone work on LLVM lifters without addressing binary type analysis?
2. Are we working on the right problems?
3. Are new immature rewriters interesting?
4. How to interpret reported success rates?

Is it better to acknowledge this unsatisfied dependency or continue to ignore it.

Questions for the community

1. Should anyone work on LLVM lifters without addressing binary type analysis?
2. Are we working on the right problems?
3. Are new immature rewriters interesting?
4. How to interpret reported success rates?

Research Focuses

- Code / Data disambiguation
- Function boundary identification
- Symbolization

Practical Problems

- Symbolization
- Extra-code structures (e.g., exceptions)
- Robust decoding of instructions

Questions for the community

1. Should anyone work on LLVM lifters without addressing binary type analysis?
2. Are we working on the right problems?
3. Are new immature rewriters interesting?
4. How to interpret reported success rates?

Pro

- Explore new techniques
- Pedagogic value in implementing end-to-end lifter
- Grad students seem to enjoy writing rewriters

Con

- Most commonly abandoned
- Lessons might not scale to production-grade lifters
- Wasted effort

Questions for the community

1. Should anyone work on LLVM lifters without addressing binary type analysis?
2. Are we working on the right problems?
3. Are new immature rewriters interesting?
4. How to interpret reported success rates?

Tool	Original	lifter-eval
DDisasm	$\frac{2850}{2865}$	85.91%
e9patch		78.34%
egalito	$\frac{90}{149}$	29.39%
mctoll		0.89%
multiverse	$\frac{16}{16}$	10.82%
reopt		33.91%
retrowrite	$\frac{28}{28}$	9.24%
revng		23.50%
Uroboros	$\frac{243}{244}$	2.87%
zipr		100.00%

Suggestions

- Standardized benchmark set.
- Development and evaluation benchmarks.
- Stop using coreutils and SPEC.

Quotes

totally practical for production deployment

we are confident that [tool] can rewrite arbitrary C binaries

Conclusion

Suggestions for research, development, and application

Research

- Binary type analysis
- Learning approaches to generalization

Development

Handle more binaries.

Stripped	no-PIE
Exceptions	Multi-threaded
ISAs (w/extensions)	File Formats
Source Languages	Compilers

Selective Application Today

- Production ready for many application environments
- Use rewritability classifiers

Reproduction & Extension

`https://gitlab.com/GrammaTech/lifter-eval`
`https://gitlab.com/GrammaTech/lifter-eval-artifacts`

Thanks!

Eric Schulte	schulte.eric@gmail.com
Michael Brown	michael.brown@trailofbits.com
Vlad Folts	vfolts@grammatech.com

Backup and References

Analysis of Binary Rewriter Success

Backup

Methodology

1. Aggregate binary rewriter success by binary features.
2. Train decision trees to predict rewriter success.
3. Inspect decision trees for most predictive features.

Retrowrite

```
def retrowrite_tree(note.gnu.build_id, pi, got.plt,
                    note.abi_tag, rela.plt,
                    data.rel.ro,interp):
  if note.gnu.build_id:
    if not pi:
      return {'FAIL': 531.0, 'PASS': 0.0}
    else: # pi
      if got.plt:
        return {'FAIL': 169.0, 'PASS': 0.0}
      else
        //...
  else: # not note.gnu.build_id
    return {'FAIL': 1166.0, 'PASS': 0.0}
```

DDisasm

```
def ddisasm_tree(note.abi_tag, interp, strip, rela.plt, pi):
  if not note.abi_tag:
    if not interp:
      if strip:
        if interp:
          return {'FAIL': 50.0, 'PASS': 112.0}
        else: # not interp
          return {'FAIL': 37.0, 'PASS': 33.0}
      else: # not strip
        return {'FAIL': 12.0, 'PASS': 0.0}
      //...
```

References I

- Dennis Andriesse, Xi Chen, Victor van der Veen, Asia Slowinska, and Herbert Bos. 2016.
  An In-Depth Analysis of Disassembly on Full-Scale x86/x64 Binaries. In 25th USENIX Security Symposium (USENIX Security 16). USENIX Association, Austin, TX, 583–600.
  https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/andriesse
- Erick Bauman, Zhiqiang Lin, and Kevin W. Hamlen. 2018.
  Superset Disassembly: Statically Rewriting x86 Binaries Without Heuristics. In NDSS.
  https://doi.org/10.14722/ndss.2018.23304

References II

Sandeep Dasgupta, Sushant Dinesh, Deepan Venkatesh, Vikram S. Adve, and Christopher W. Fletcher. 2020.
Scalable Validation of Binary Lifters. In Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation (London, UK) (PLDI 2020). Association for Computing Machinery, New York, NY, USA, 655–671.
https://doi.org/10.1145/3385412.3385964

Alessandro Di Federico, Mathias Payer, and Giovanni Agosta. 2017.
rev.ng: a unified binary analysis framework to recover CFGs and function boundaries. In Proceedings of the 26th International Conference on Compiler Construction. 131–141.

References III

Artem Dinaburg and Andrew Ruef. 2014.
McSema: Static translation of x86 instructions to llvm. In ReCon 2014 Conference, Montreal, Canada.

Sushant Dinesh. 2019.
RetroWrite: Statically Instrumenting COTS Binaries for Fuzzing and Sanitization.

Ph. D. Dissertation. figshare.

Gregory J Duck, Xiang Gao, and Abhik Roychoudhury. 2020.
Binary rewriting without control flow recovery. In Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation. 151–163.

References IV

- Antonio Flores-Montoya and Eric Schulte. 2020.
  Datalog Disassembly. In 29th USENIX Security Symposium (USENIX Security 20). USENIX Association, 1075–1092.
  https://www.usenix.org/conference/usenixsecurity20/presentation/flores-montoya
- Inc. Galois. 2021.
  ReOpt.
  https://github.com/GaloisInc/reopt.

References V

- Jason D Hiser, Anh Nguyen-Tuong, Michele Co, Benjamin Rodes, Matthew Hall, Clark L Coleman, John C Knight, and Jack W Davidson. 2014. A Framework for Creating Binary Rewriting Tools (Short Paper). In Dependable Computing Conference (EDCC), 2014 Tenth European. IEEE, 142–145.
- Pantea Kiaei, Cees-Bart Breunesse, Mohsen Ahmadi, Patrick Schaumont, and Jasper van Woudenberg. 2020. Rewrite to Reinforce: Rewriting the Binary to Apply Countermeasures against Fault Injection. arXiv preprint arXiv:2011.14067 (2020).

References VI

- M. A. Laurenzano, M. M. Tikir, L. Carrington, and A. Snavely. 2010. PEBIL: Efficient static binary instrumentation for Linux. In 2010 IEEE International Symposium on Performance Analysis of Systems Software (ISPASS). 175–183. https://doi.org/10.1109/ISPASS.2010.5452024
- Kaiyuan Li, Maverick Woo, and Limin Jia. 2020. On the Generation of Disassembly Ground Truth and the Evaluation of Disassemblers. In Proceedings of the 2020 ACM Workshop on Forming an Ecosystem Around Software Transformation. 9–14.

References VII

Xiaozhu Meng and Barton P. Miller. 2016.
Binary Code is Not Easy. In Proceedings of the 25th International Symposium on Software Testing and Analysis (Saarbrücken, Germany) (ISSTA 2016). ACM, New York, NY, USA, 24–35.
https://doi.org/10.1145/2931037.2931047

Microsoft. 2022.
mctoll.
https://github.com/microsoft/llvm-mctoll.

References VIII

- Chengbin Pang, Ruotong Yu, Yaohui Chen, Eric Koskinen, Georgios Portokalidis, Bing Mao, and Jun Xu. 2021.
  Sok: All you ever wanted to know about x86/x64 binary disassembly but were afraid to ask. In 2021 IEEE Symposium on Security and Privacy (SP). IEEE, 833–851.
- Chenxiong Qian, Hong Hu, Mansour Alharthi, Pak Ho Chung, Taesoo Kim, and Wenke Lee. 2019.
  {RAZOR}: A framework for post-deployment software debloating. In 28th {USENIX} Security Symposium ({USENIX} Security 19). 1733–1750.

References IX

Ted Romer, Geoff Voelker, Dennis Lee, Alec Wolman, Wayne Wong, Hank Levy, Brian Bershad, and Brad Chen. 1997.
Instrumentation and optimization of Win32/Intel executables using Etch. In Proceedings of the USENIX Windows NT Workshop, Vol. 1997. 1–8.

Benjamin Schwarz, Saumya Debray, Gregory Andrews, and Matthew Legendre. 2001.
Plto: A link-time optimizer for the Intel IA-32 architecture. In Proc. 2001 Workshop on Binary Translation (WBT-2001).

References X

- Matthew Smithson, Khaled ElWazeer, Kapil Anand, Aparna Kotha, and Rajeev Barua. 2013.
  Static binary rewriting without supplemental information: Overcoming the tradeoff between coverage and correctness. In Reverse Engineering (WCRE), 2013 20th Working Conference on. IEEE, 52–61.
- Software Engineering Institute. 2022.
  Automated static analysis tools for binary programs.
  https://github.com/cmu-sei/pharos.
- Eli Tilevich and Yannis Smaragdakis. 2005.
  Binary refactoring: Improving code behind the scenes. In Proceedings of the 27th international conference on Software engineering. ACM, 264–273.

References XI

- Ludo Van Put, Dominique Chanet, Bruno De Bus, Bjorn De Sutter, and Koen De Bosschere. 2005.
  Diablo: a reliable, retargetable and extensible link-time rewriting framework.
  In Proceedings of the Fifth IEEE International Symposium on Signal Processing and Information Technology, 2005. IEEE, 7–12.

- Ruoyu Wang, Yan Shoshitaishvili, Antonio Bianchi, Aravind Machiry, John Grosen, Paul Grosen, Christopher Kruegel, and Giovanni Vigna. 2017.
  Ramblr: Making Reassembly Great Again. In NDSS.

References XII

- Shuai Wang, Pei Wang, and Dinghao Wu. 2015.
  Reassembleable Disassembling. In 24th USENIX Security Symposium (USENIX Security 15). USENIX Association, Washington, D.C., 627–642.
  https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/wang-shuai
- Richard Wartell, Vishwath Mohan, Kevin W Hamlen, and Zhiqiang Lin. 2012.

  Securing untrusted code via compiler-agnostic binary rewriting. In Proceedings of the 28th Annual Computer Security Applications Conference. ACM, 299–308.

References XIII

Matthias Wenzl, Georg Merzdovnik, Johanna Ullrich, and Edgar Weippl. 2019.
From hack to elaborate technique—a survey on binary rewriting.
ACM Computing Surveys (CSUR) 52, 3 (2019), 1–37.

David Williams-King, Hidenori Kobayashi, Kent Williams-King, Graham Patterson, Frank Spano, Yu Jian Wu, Junfeng Yang, and Vasileios P Kemerlis. 2020.
Egalito: Layout-Agnostic Binary Recompilation. In Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems. 133–147.

References XIV

William Woodruff, Niki Carroll, and Sebastiaan Peters. 2021.
Differential analysis of x86-64 instruction decoders. In Proceedings of the Seventh Language-Theoretic Security Workshop (LangSec) at the IEEE Symposium on Security and Privacy.

Xiaoyang Xu, Masoud Ghaffarinia, Wenhao Wang, Kevin W Hamlen, and Zhiqiang Lin. 2019.
{CONFIRM}: Evaluating compatibility and relevance of control-flow integrity protections for modern software. In 28th USENIX Security Symposium (USENIX Security 19). 1805–1821.

References XV

Mingwei Zhang, Rui Qiao, Niranjan Hasabnis, and R Sekar. 2014.
A platform for secure static binary instrumentation. In Proceedings of the 10th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments. ACM, 129–140.

Mingwei Zhang and R Sekar. 2013.
Control Flow Integrity for COTS Binaries.. In USENIX Security. 337–352.