Reassembleable Disassembling

Shuai Wang, Pei Wang, Dinghao Wu

Pennsylvania State University

24th USENIX Security Symposium, August 2015

Motivation

Analysing and *retrofitting* COTS binaries with. . .

- software fault isolation
- control-flow integrity
- symbolic taint analysis
- elimination of ROP gadgets

Motivation

Analysing and *retrofitting* COTS binaries with…

- software fault isolation
- control-flow integrity
- symbolic taint analysis
- elimination of ROP gadgets

Binary rewriting comes with major drawbacks/limitations

- runtime overhead from patching due to control-flow transfers
- patching requires PIC if code is relocated
- instrumentation significantly increases binary size
- binary reuse only works for small binaries (coverage)

Goal

Produce *reassembleable* assembly code from *stripped COTS binaries* in a *fully automated* manner.

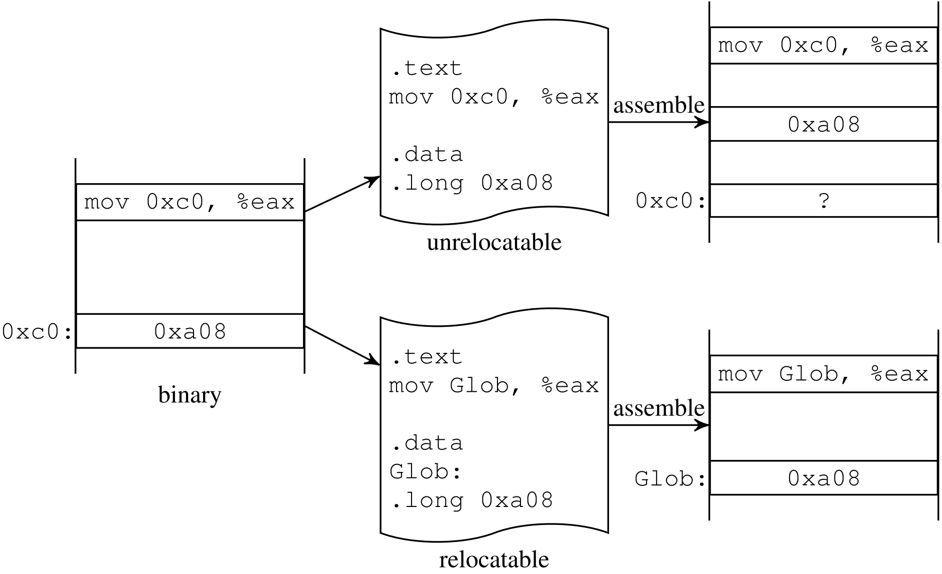
- Allows binary-based whole program transformations
- Requires relocatable assembly code → symbolization of immediate values
- Complementary to existing work

Symbolization

Given an immediate value in assembly code,
is it a constant or a memory address?

- Reassembling transformed program changes binary layout
- Address changes invalidate memory references
- x86
  - No distinction between code and data
  - Variable-length instruction encoding

(Un)Relocatable Assembly Code



```
mov 0xc0, %eax

0xc0: 0xa08
```

binary

```
.text
mov 0xc0, %eax

.data
.long 0xa08
```

unrelocatable

```
mov 0xc0, %eax

0xa08

0xc0: ?
```

```
.text
mov Glob, %eax

.data
Glob:
.long 0xa08
```

relocatable

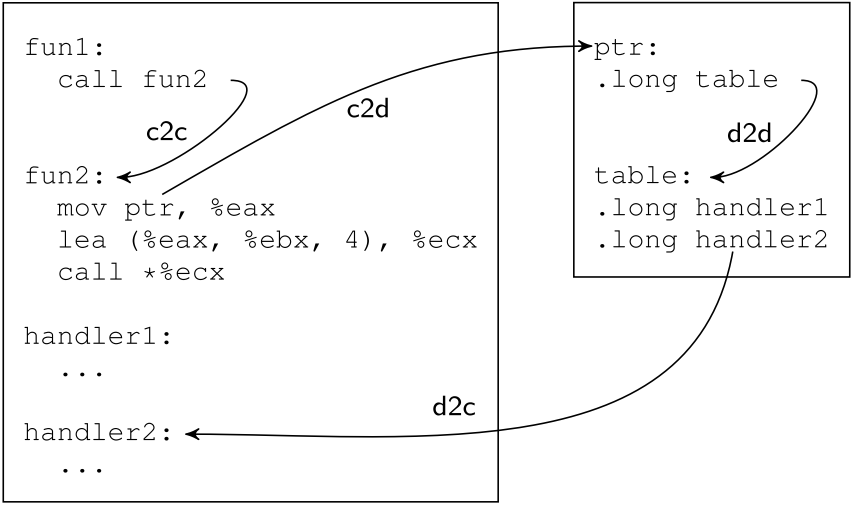
```
mov Glob, %eax

Glob: 0xa08
```

Types of Symbol References

Code Section

Data Section



```
fun1:
  call fun2

fun2:
  mov ptr, %eax
  lea (%eax, %ebx, 4), %ecx
  call *%ecx

handler1:
  ...

handler2:
  ...
```

```
ptr:
.long table

table:
.long handler1
.long handler2
```

c2c

c2d

d2d

d2c

Symbolization of c2c and c2d References

- Valid memory references point into code or data section
- Assume all immediates to be references and filter out invalid ones

Symbolization of d2c and d2d References

Assumption 1

"*All symbol references stored in data sections are n-byte aligned, where n is 4 for 32-bit binaries and 8 for 64-bit binaries.*"

→ Consider only n-byte values which are n-byte aligned

Symbolization of d2c and d2d References

Assumption 1

"*All symbol references stored in data sections are n-byte aligned, where n is 4 for 32-bit binaries and 8 for 64-bit binaries.*"

→ Consider only n-byte values which are n-byte aligned

Assumption 2

"*Users do not need to perform transformation on the original binary data.*"

→ Keep start addresses of data sections during reassembly and ignore d2d references

Symbolization of d2c and d2d References

Assumption 1

"*All symbol references stored in data sections are n-byte aligned, where n is 4 for 32-bit binaries and 8 for 64-bit binaries.*"

→ Consider only n-byte values which are n-byte aligned

Assumption 2

"*Users do not need to perform transformation on the original binary data.*"

→ Keep start addresses of data sections during reassembly and ignore d2d references

Assumption 3

"*d2c symbol references are only used as function pointers or jump table entries.*"

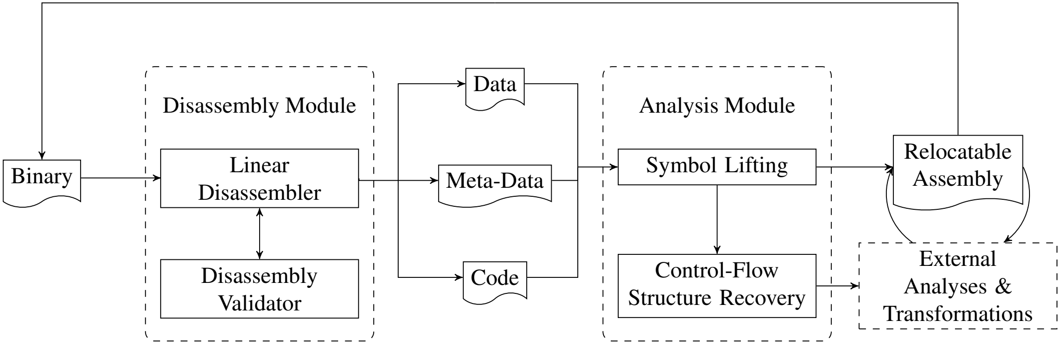
→ References need to point to start of a function or form a jump table

Evaluation

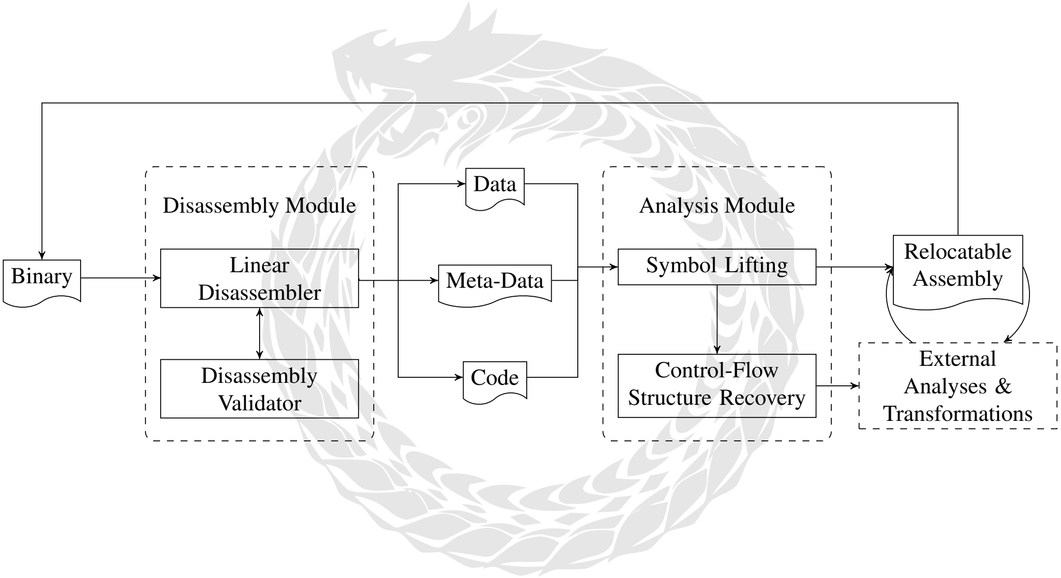
- Uroboros: 13,209 SLOC in OCaml and Python; works with x86/x64 ELF binaries
- Intel Core i7-3770 @ 3.4GHz with 8GiB RAM running Ubuntu 12.04
- 122 programs compiled for 32- and 64-bit targets
- gcc 4.6.3 with default configuration and optimization of each program
- `stripped` before testing

Collection	**Size**	**Content**
Coreutils	103	GNU Core Utilities
Real	7	bc, ctags, gzip, mongoose, nweb, oftpd, thttpd
Spec	12	C programs in SPEC2006

Architecture of UROBOROS



Architecture of UROBOROS



https://openclipart.org/detail/215030/

Correctness

Test input shipped with programs or custom test of major functionality (some of REAL)

Assumption Set	Binaries Failing Functionality Tests	
	32-bit	64-bit
{}	h264ref, gcc, gobmk, hmmer	perlbench, gcc, gobmk, hmmer, sjeng, h264ref, lbm, sphinx3
{A1}	h264ref, gcc, gobmk	perlbench, gcc, gobmk
{A1, A2}	h264ref, gcc, gobmk	perlbench, gcc, gobmk
{A1, A3}	gobmk	gcc, gobmk
{A1, A2, A3}	gobmk	

Symbolization Errors

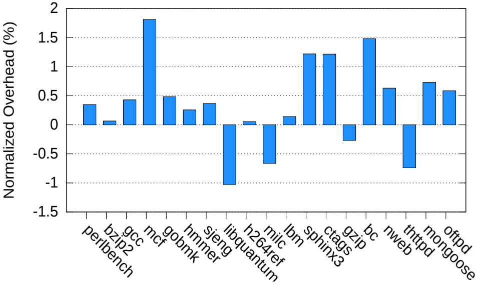
Table 4: Symbolization false positives of 32-bit SPEC, REAL and COREUTILS (Others have zero false positive)

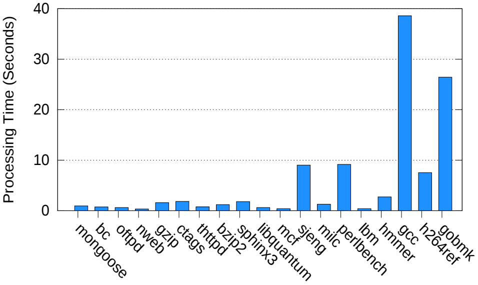
Benchmark	# of Ref.	Assumption Set									
		{}		{A1}		{A1, A2}		{A1, A3}		{A1, A2, A3}	
		FP	FP Rate	FP	FP Rate	FP	FP Rate	FP	FP Rate	FP	FP Rate
perlbench	76538	2	0.026‰	0	0.000‰	0	0.000‰	0	0.000‰	0	0.000‰
hmmer	13127	12	0.914‰	0	0.000‰	0	0.000‰	0	0.000‰	0	0.000‰
h264ref	20600	27	1.311‰	1	0.049‰	1	0.049‰	0	0.000‰	0	0.000‰
gcc	262698	49	0.187‰	32	0.122‰	32	0.122‰	0	0.000‰	0	0.000‰
gobmk	65244	1348	20.661‰	985	15.097‰	912	13.978‰	78	1.196‰	5	0.077‰

Table 5: Symbolization false negatives of 32-bit SPEC, REAL and COREUTILS (Others have zero false negative)

Benchmark	# of Ref.	Assumption Set									
		{}		{A1}		{A1, A2}		{A1, A3}		{A1, A2, A3}	
		FN	FN Rate	FN	FN Rate	FN	FN Rate	FN	FN Rate	FN	FN Rate
perlbench	76538	2	0.026‰	0	0.000‰	0	0.000‰	0	0.000‰	0	0.000‰
hmmer	13127	12	0.914‰	0	0.000‰	0	0.000‰	0	0.000‰	0	0.000‰
h264ref	20600	27	1.311‰	0	0.000‰	0	0.000‰	0	0.000‰	0	0.000‰
gcc	262698	11	0.042‰	0	0.000‰	0	0.000‰	0	0.000‰	0	0.000‰
gobmk	65244	86	1.318‰	0	0.000‰	0	0.000‰	0	0.000‰	0	0.000‰

Overhead for REAL and SPEC





No increase in binary size after first disassemble-assemble cycle

Conclusion

- Heuristic-based symbolization of memory references
- UROBOROS[1] provides reassembleable disassembly
- Assumes availability of raw disassembly and function starting addresses
- Tested with gcc and Clang compiled binaries
- Limited support for C++ (need to parse DWARF)

[1] Available at `https://github.com/s3team/uroboros`