RamBlr

Making Reassembly Great Again

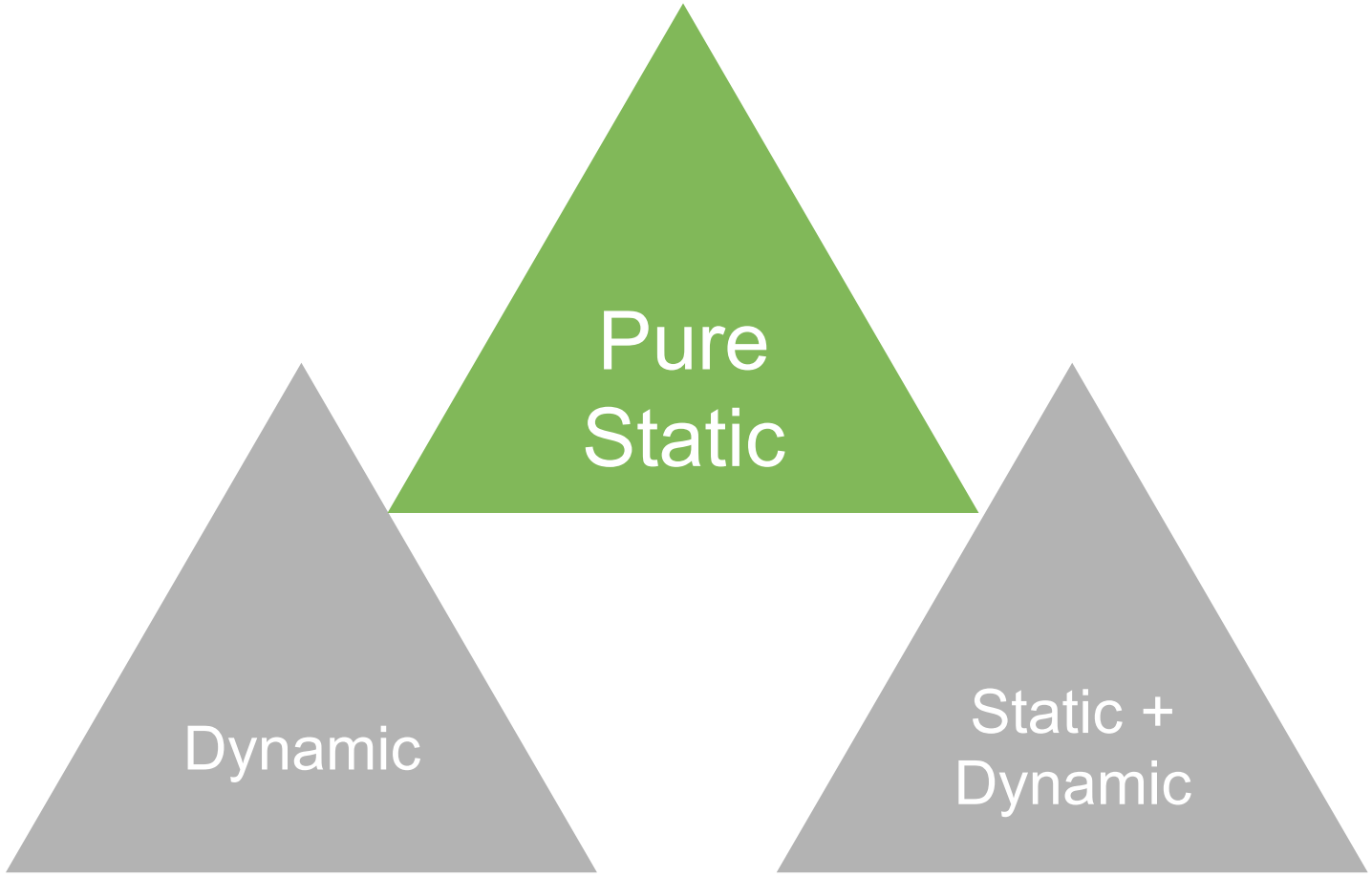
Ruoyu “Fish” Wang, Yan Shoshitaishvili, Antonio Bianchi,
Aravind Machiry, John Grosen, Paul Grosen,
Christopher Kruegel, Giovanni Vigna

Motivation


lenovo


00000
I-210+
I211
000 000 000

Available Solutions



What is Binary Reassembly?

Disassemble

```
          .text
400100    mov   [6000a0], eax
400105    jmp   0x40020d
          ...
40020d    mov   [6000a4], 1

          .data
6000a0    .long 0xc0deb4be
6000a4    .long 0x0
```

Disassemble

```
                .text
                mov   [data_0], eax
                jmp   target
                ...
target          mov   [data_1], 1

                .data
data_0          .long 0xc0deb4be
data_1          .long 0x0
```

Patch & Assemble

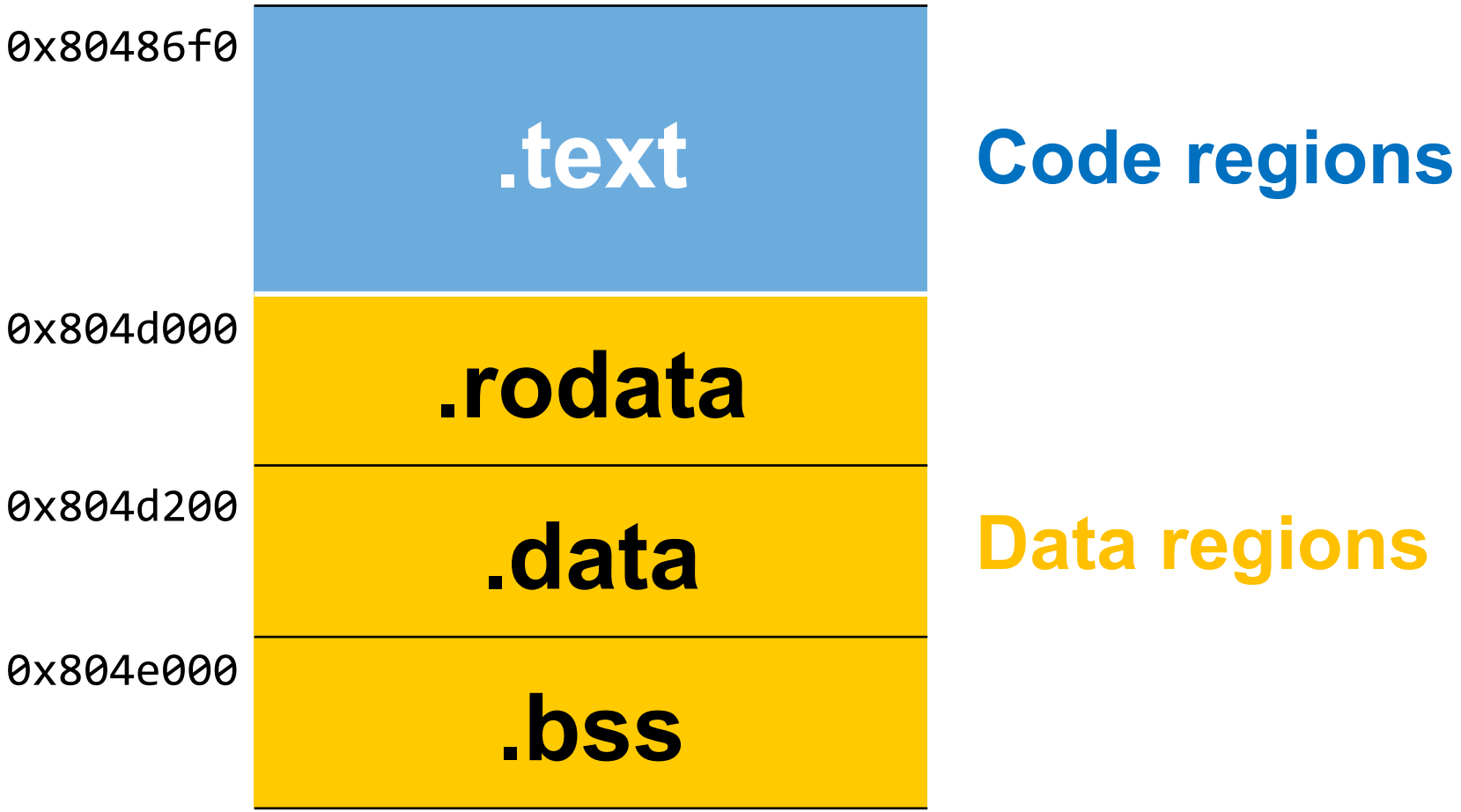
```
                .text
400100          mov   [6000a0], eax
400105          jmp   40020d
40020d                CRASH!
40020f          mov   [6000a4], 1

                .data
6000a0          “cat\x00”
6000a4          .long 0x0
6000a8
```

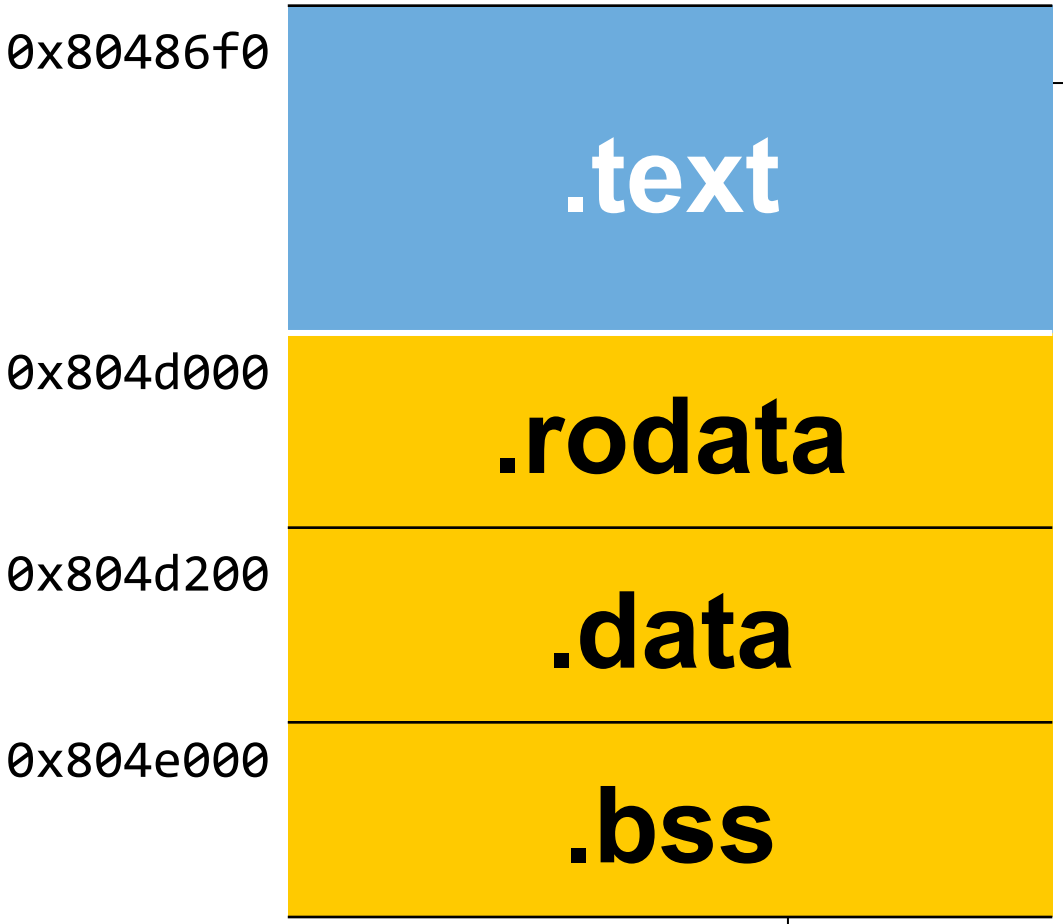
Non-relocatable Assembly

Patch & Assemble

```
                .text
                mov   [data_0], eax
                jmp   target
                ...
                mov   [data_1], 1
target          mov   [data_1], 1
                .data
data_0          .long 0xc0deb4be
data_1          .long 0x0
```

Relocatable Assembly


0x80486f0
.text
0x804d000
.rodata
0x804d200
.data
0x804e000
.bss
Code regions
Data regions



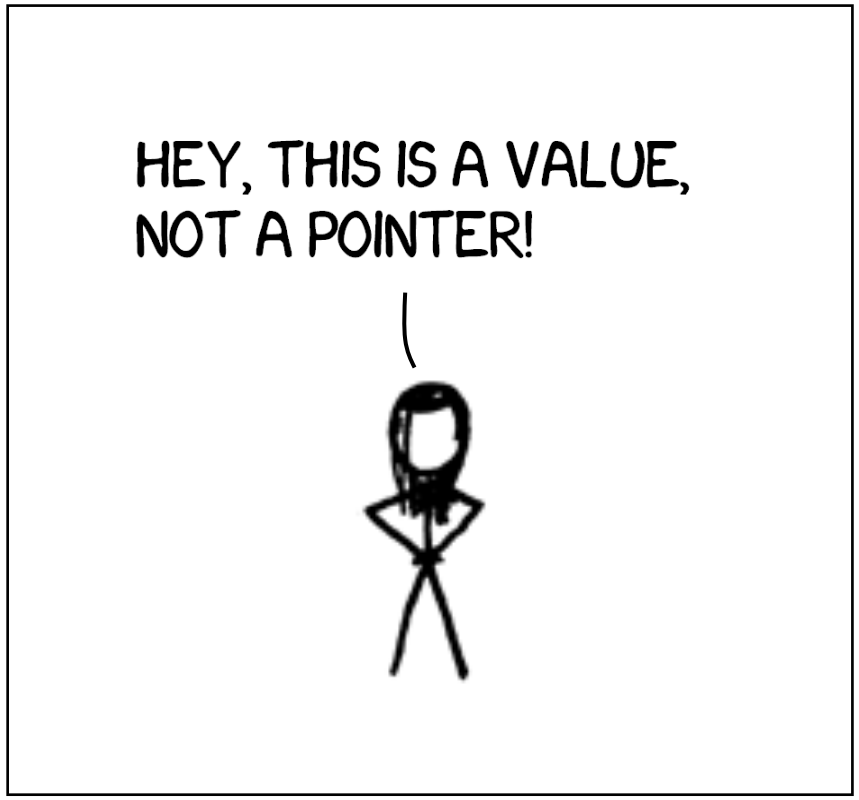

```
push   ebp
mov    ebp, esp
sub    esp, 0x48
mov    DWORD PTR [ebp-0x10], 0x0
mov    DWORD PTR [ebp-0xc], 0x0
mov    DWORD PTR [ebp-0xc], 0x80540a0
mov    eax, 0xfb7
mov    WORD PTR [ebp-0x10], ax
mov    eax, ds:0x805be60
test   eax, eax
jne    0x804895b
mov    eax, ds:0x805be5c
```

```
...

.data:
804d538:    0x8048eec
804d53c:    0x8048f05
804d540:    0x8048f1e
```

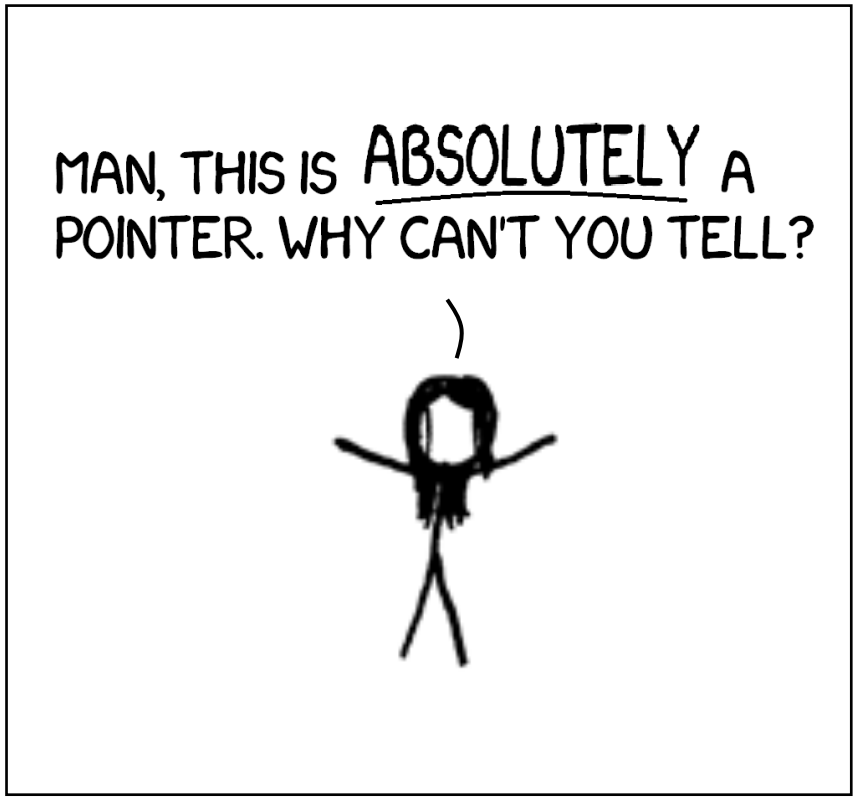
Uroboros

USENIX Sec '15

Problems



False Positives



False Negatives

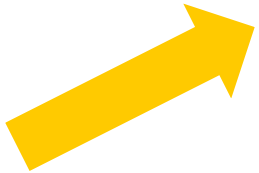
Thanks **xkcd :-)**

False Positives

Problem: Value Collisions

```
/* stored at 0x8060080 */
static float a = 4e-34;
```

A Floating-point Variable *a*



```
8060080       .db 3d
8060081       .db ec
8060082       .db 04
8060083       .db 08
```

Byte Representation

```
8060080       label_804ec3d
```

Interpreted as a Pointer

False Negatives

Problem: Compiler Optimization

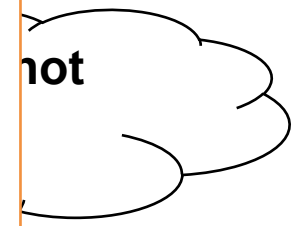
```
int ctrs[2] = {0};

int main()
{
    int input = getchar();
    switch (input - 'A')
    {
        case 0:
            ctrs[input - 'A']++;
            break;
        ...
    }
}
```

A code snippet allows **constant folding**

False Negatives

Problem: Compiler Optimization

```
int ct                                                    a034

int ma
{                                                         ], 1
    in
    sw
    {
        0x804a034 – ‘A’ * sizeof(int) = 0x8049f30

    }
}
```



A code snippet allows **constant folding** Compiled in Clang with –O1

Our Approach

Naïve Strategy

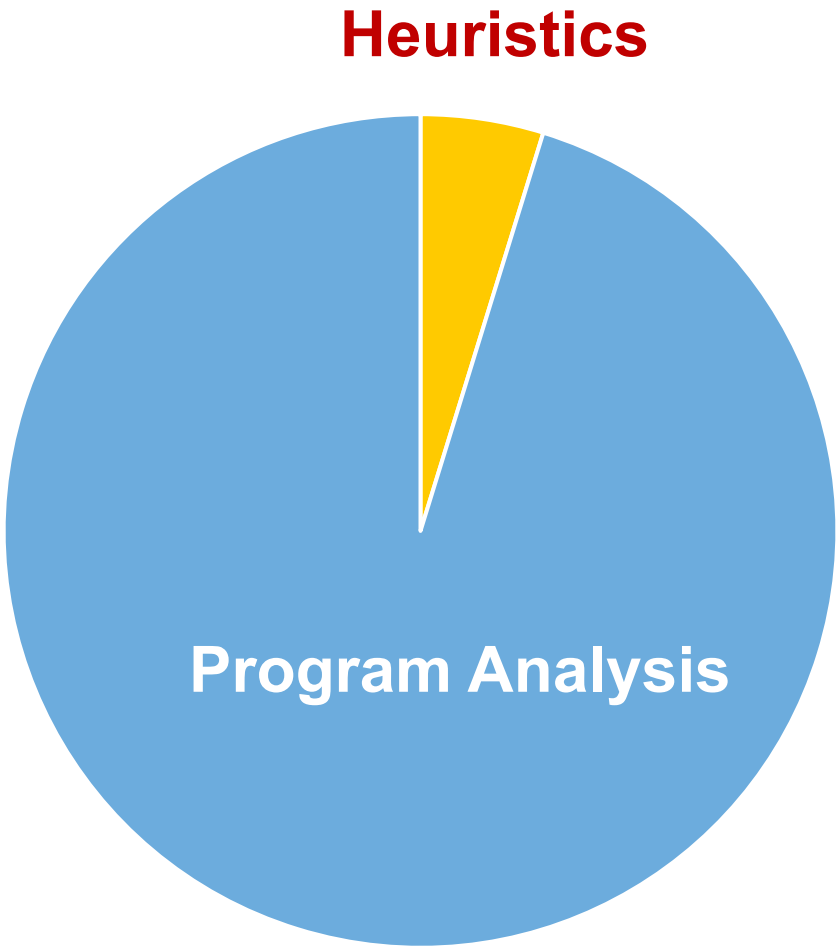
False Positives

False Negatives

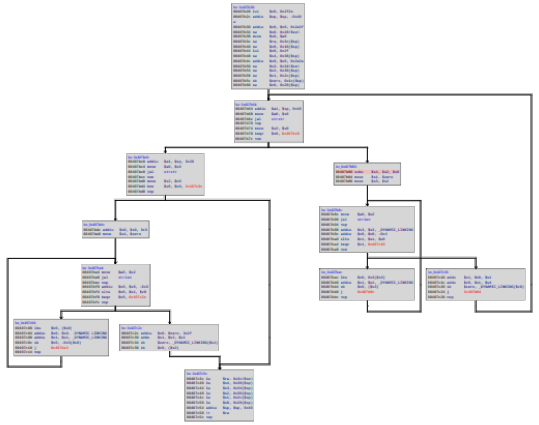
Ramblr

False Positives

False Negatives



Pipeline



CFG
Recovery

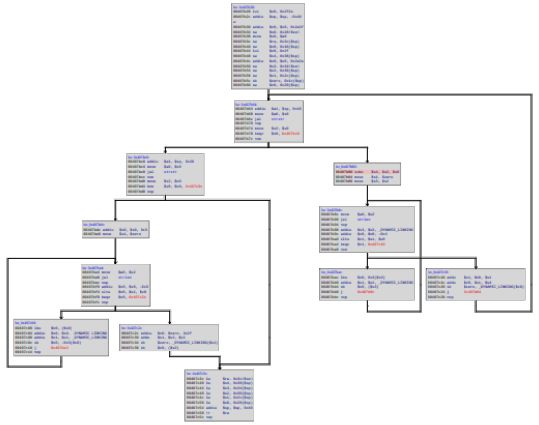
0x804850b	Pointer
0xa	Integer
0xdc5	Integer
63 61 74 00	String
0x80484a2	Pointer
0x804840b	Pointer
0xa0000	Integer

Content Classification

```
push    offset label_34
push    offset label_35
cmp     eax, ecx
jne     label_42

.label_42:
mov     eax, 0x12fa9e5
...
```

Symbolization
&
Reassembly

Pipeline



CFG
Recovery

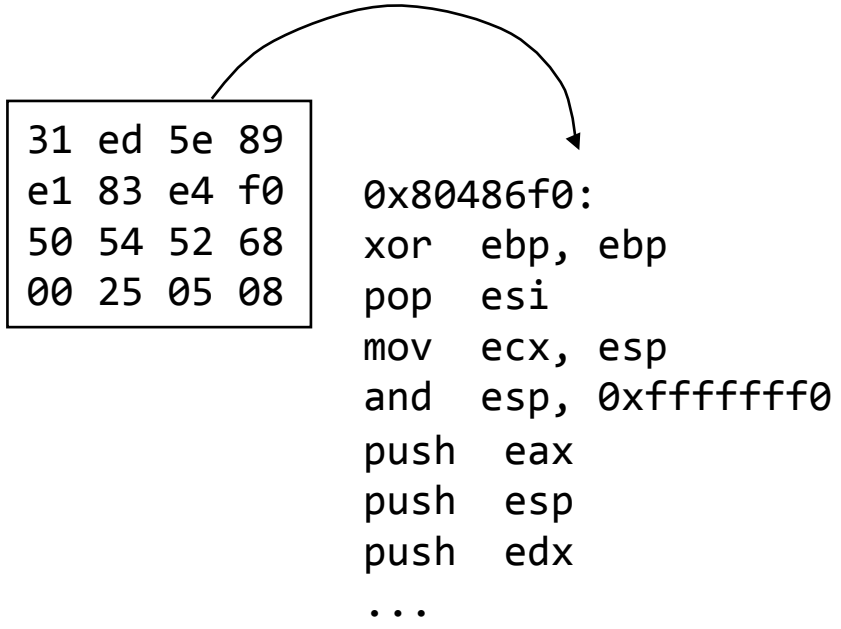
0x804850b	Pointer
0xa	Integer
0xdc5	Integer
63 61 74 00	String
0x80484a2	Pointer
0x804840b	Pointer
0xa0000	Integer

Content Classification

```
push    offset label_34
push    offset label_35
cmp     eax, ecx
jne     label_42

.label_42:
mov     eax, 0x12fa9e5
...
```

Symbolization
&
Reassembly

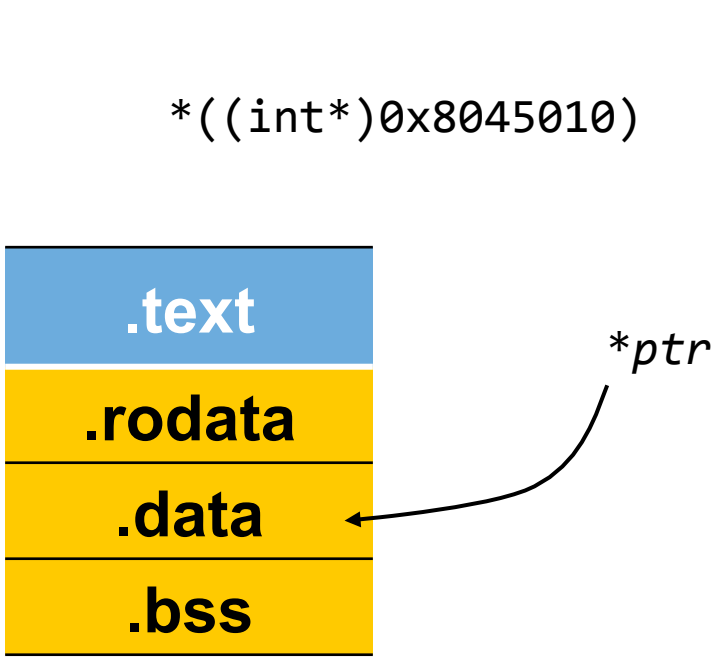
CFG Recovery



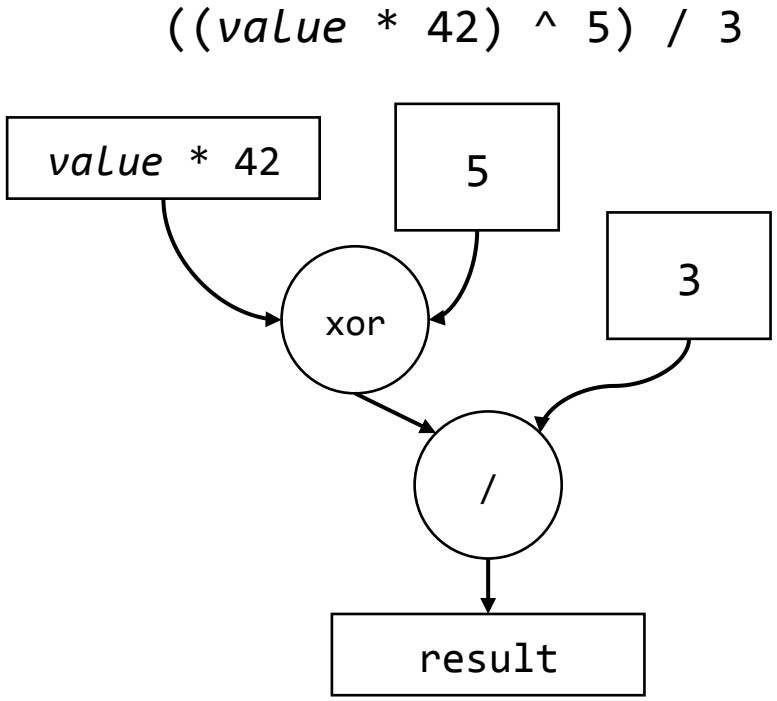
Recursive Disassembly

Iterative Refinement

Content Classification



A Typical Pointer



A Typical Value

Content Classification

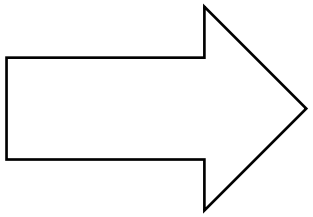
Type Category	Examples
Primitive types	Pointers, shorts, DWORDs, QWORDs, Floating-point values, etc.
Strings	Null-terminated ASCII strings, Null-terminated UTF-16 strings
Jump tables	A list of jump targets
Arrays of primitive types	An array of pointers, a sequence of integers

Data Types that Ramblr Recognizes

Content Classification

MOVe **S**calar **D**ouble-precision
floating-point value

```
movsd   xmm0, ds:0x804d750
movsd   xmm1, ds:0x804d758
```

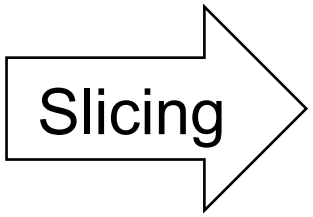


Two floating-points	
804d750	Floating point integer
804d758	Floating point integer

Recognizing Types during CFG Recovery

Content Classification

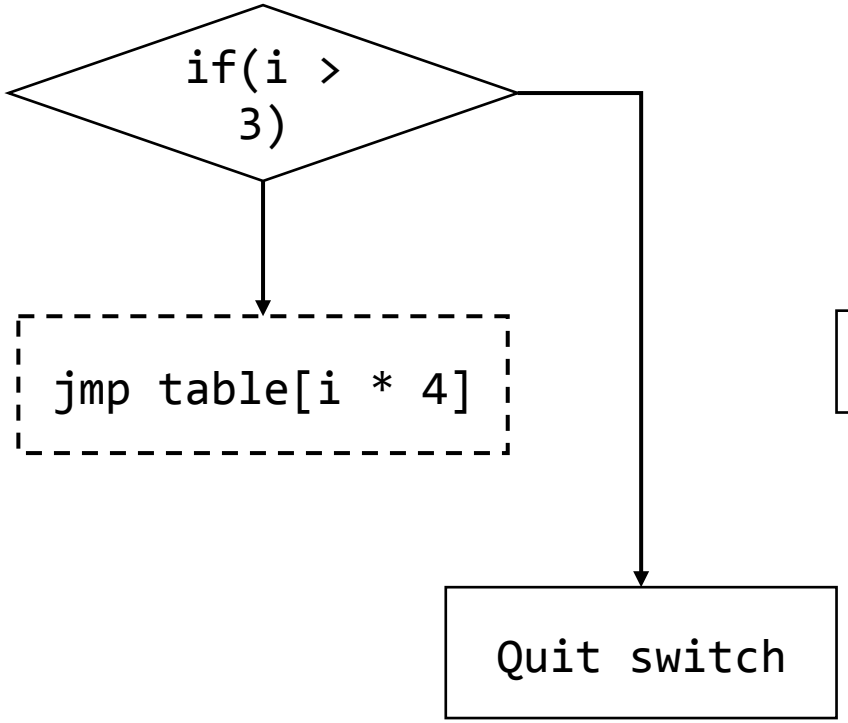
```
chr = _getch();
switch (i)
{
    case 1:
        a += 2; break;
    case 2:
        b += 4; break;
    case 3:
        c += 6; break;
    default:
        a = 0; break;
}
```




```
switch (i)
{
    case 1:
        ...
    case 2:
        ...
    case 3:
        ...
    default:
        ...
}
```

Recognizing Types with Slicing & VSA

Content Classification



VSA

i = [0, 2] with a stride of 1

A jump table of 3 entries	
table[0]	Pointer, jump target
table[1]	Pointer, jump target
table[2]	Pointer, jump target

Recognizing Types with Slicing & VSA

False Negatives

Base Pointer Reattribution

```
int ctrs[2] = {0};

int main()
{
    int input = getchar();
    switch (input – 'A')
    {
        case 0:
            ctrs[input – 'A']++;
            break;
        ...
    }
}
```

```
; Assuming ctrs is stored at 0x804a034
; eax holds the input character
; ctrs[input – 'A']++;
            add    0x8049f30[eax * 4], 1
...

.bss
804a034:  ctrs[0]
804a038:  ctrs[1]
```

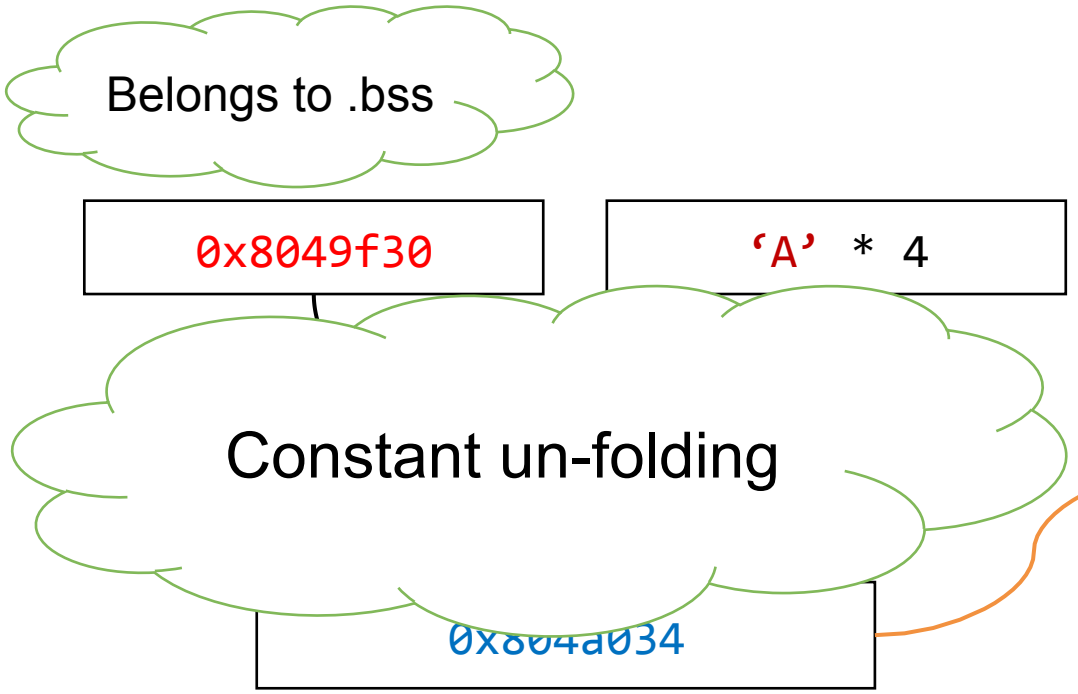
0x8049f30 does not belong to any section

A code snippet allows **constant folding**

Compiled in Clang with –O1

False Negatives

Base Pointer Reattribution



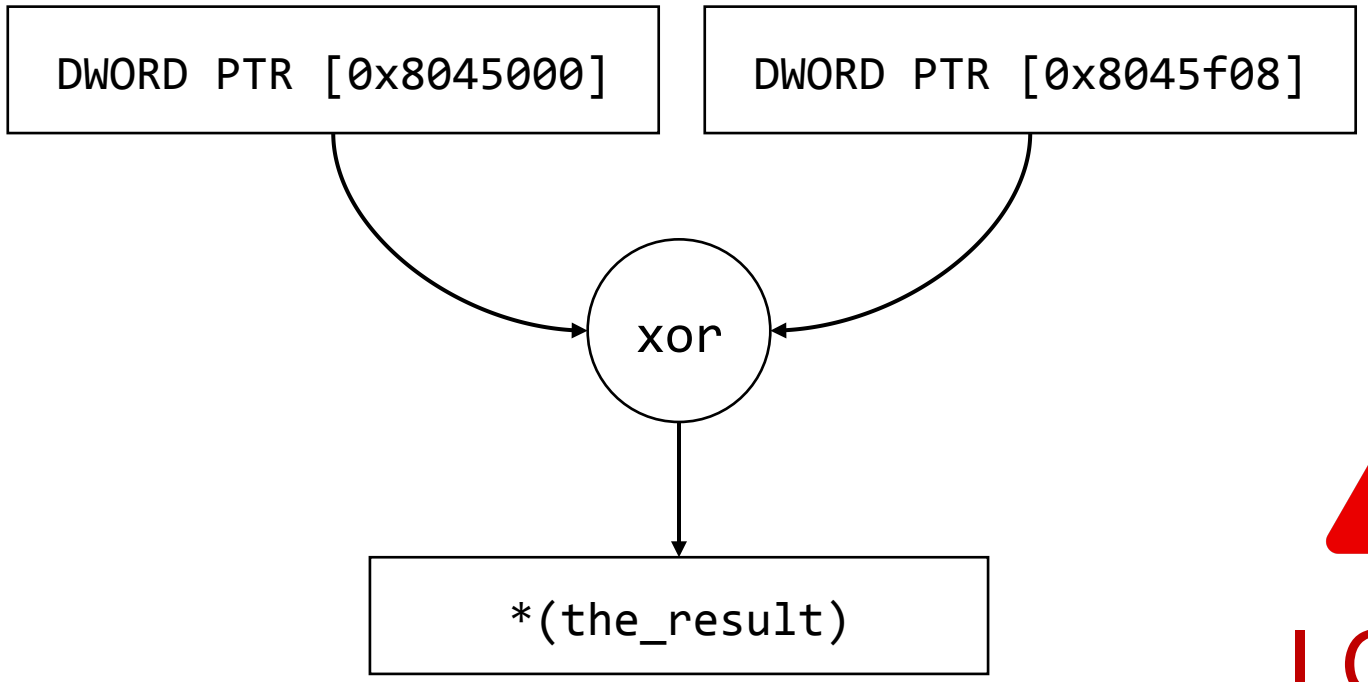

```
; Assuming ctrs is stored at 0x804a034
; eax holds the input character
; ctrs[input – 'A']++;
          add    0x8049f30[eax * 4], 1
...

.bss
804a034:  ctrs[0]
804a038:  ctrs[1]
```

0x8049f30 does not belong to any section

The Slicing Result

Compiled in Clang with –O1

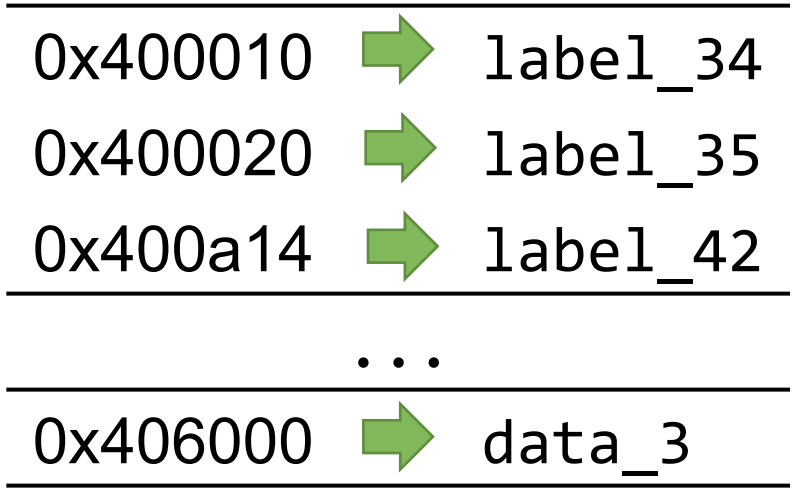
Safety Heuristics: Data Consumer Check



I GIVE UP

Unusual Behaviors Triggering the Opt-out Rule

Symbolization & Reassembly



Symbolization

```
push    offset label_34
push    offset label_35
cmp     eax, ecx
jne     label_42

.label_42:
mov     eax, 0x12fa9e5
...
```

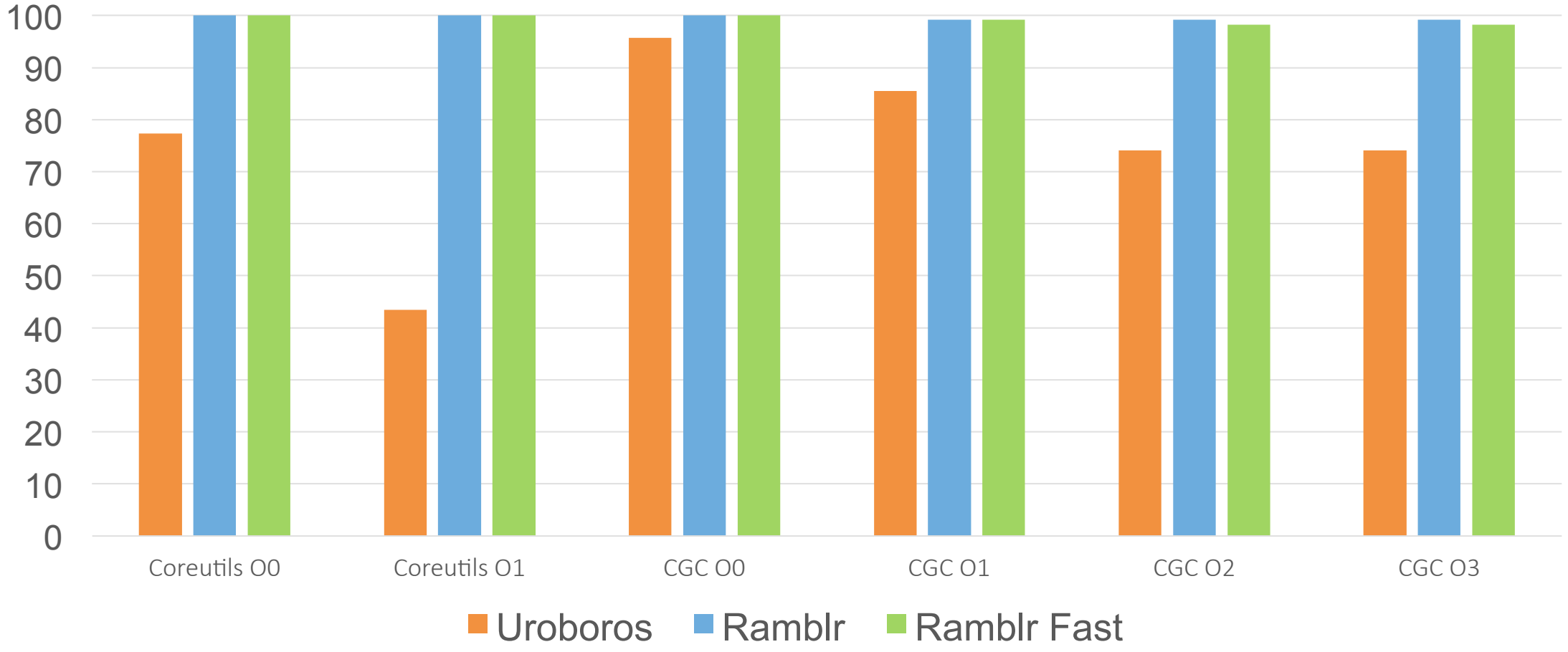
Assembly Generation

Evaluation

Data sets

	Coreutils 8.25.55	Binaries from CGC
Programs	106	143
Compiler	CGC 5	Clang 4.4
Optimization levels	O0/O1/O2/O3/Os/Ofast	
Architectures	X86/AMD64	X86
Test cases	Yes	Yes
Total binaries	**1272**	**725**

Brief Results: Success Rate



CYBER
GRAND_CHALLENGE

SHELLPHISH

Ramblr is the foundation of ...

- Patching Vulnerabilities
- Obfuscating Control Flows
- Optimizing Binaries
- Hardening Binaries

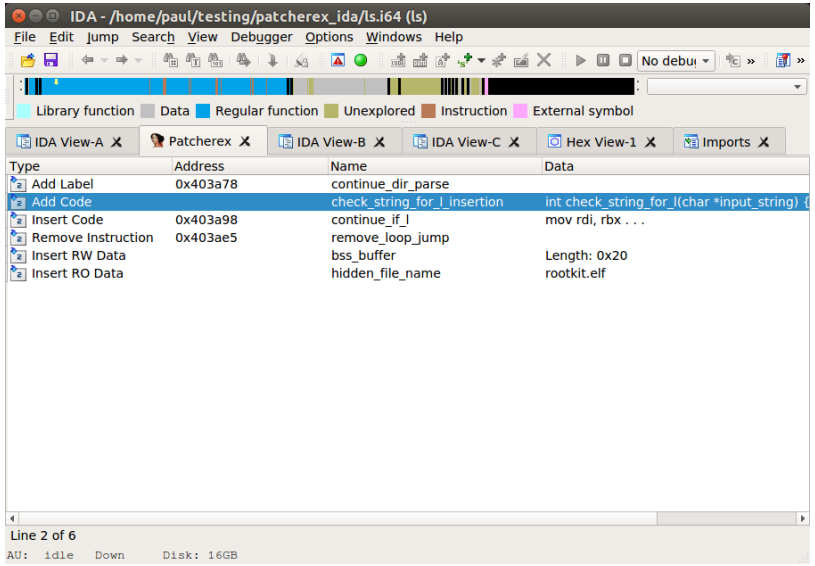
Conclusion

Conclusion

- Identified challenges in reassembling
- Proposed a novel composition of static analysis techniques
- Developed a systematic approach to reassemble stripped binaries

- ✓ Ramblr is open-sourced
- ✓ Extra data-sets and usable tools will be released soon

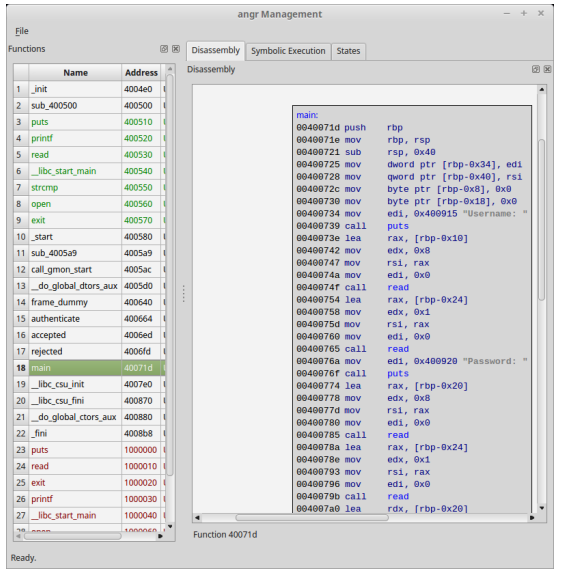
Tools



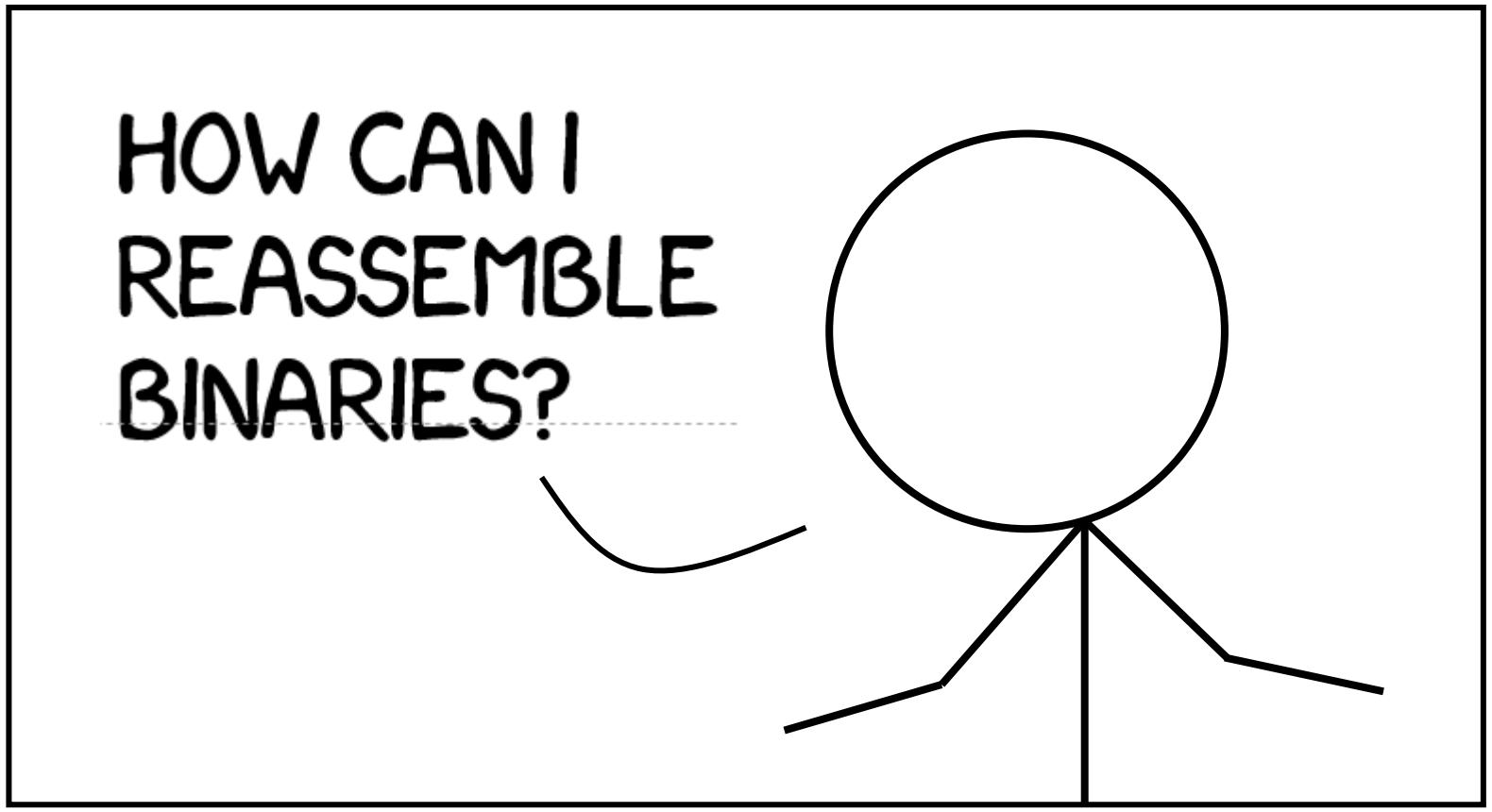
Ramblr IDA Plugin

CYBER GRAND_CHALLENGE

Patcherex



Patching support in angr Management



Thanks **xkcd :-)**

Limitations

- The **infeasibility** of static content classification
- The lack of guarantee of our approaches
- The “80% versus 20%” problem

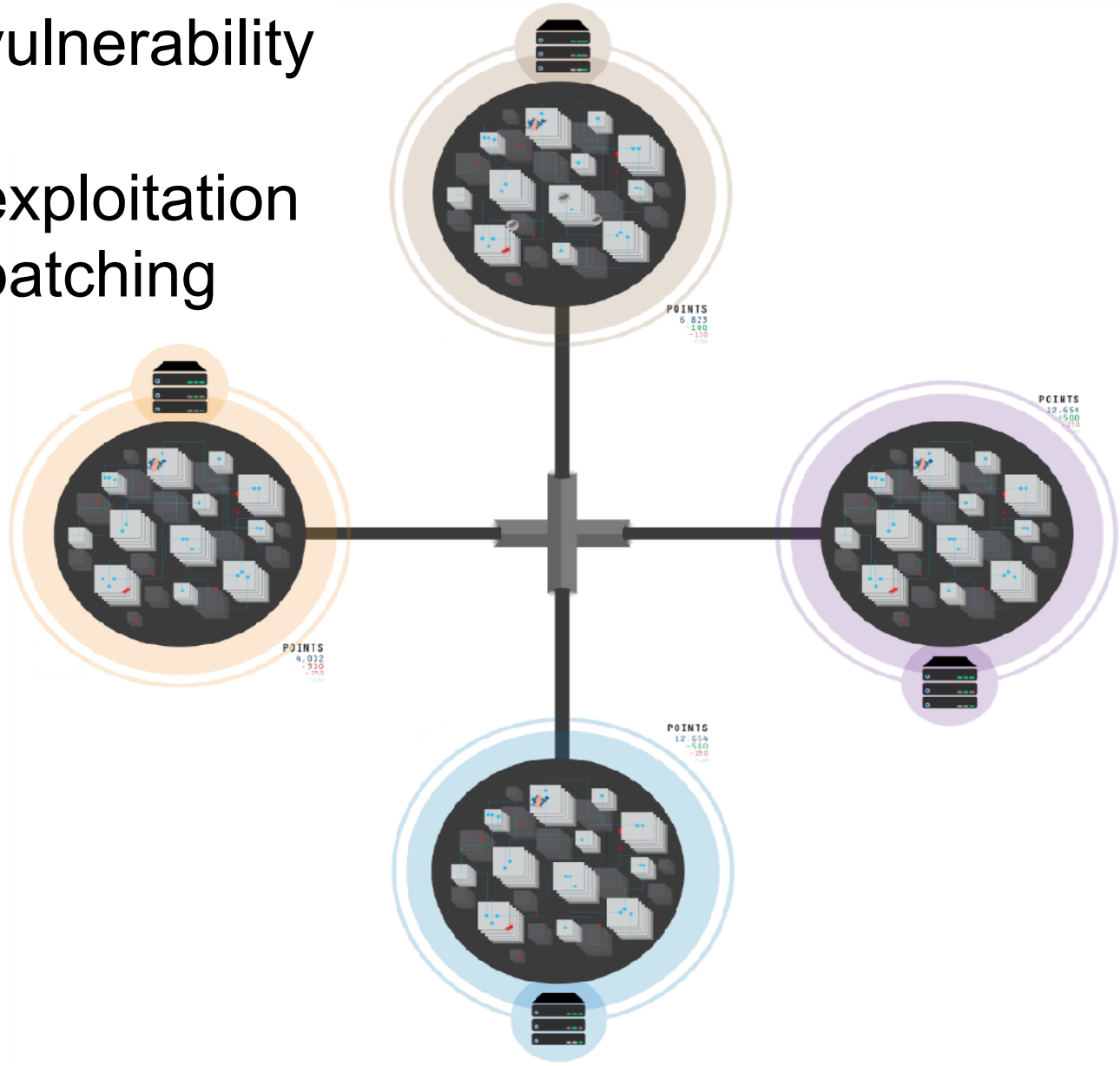
Brief Results: Success Rate (cont.)

- Emphasis

We reproduced Uroboros' results on Coreutils 8.15 compiled with GCC 4.6 on Ubuntu 12.04

- Changes in Coreutils > 8.15 makes it harder for Uroboros
- Optimizations in GCC 5 yields new challenges for Uroboros

- Autonomous vulnerability discovery
- Autonomous exploitation
- Autonomous patching



- Autonomous vulnerability discovery
- Autonomous exploitation
- Autonomous **patching**

Requires

a low memory overhead

and

an **EXTREMELY** low execution overhead