Ramblr
Making Reassembly Great Again
Ruoyu “Fish” Wang, Yan Shoshitaishvili, Antonio Bianchi, Aravind Machiry, John Grosen, Paul Grosen, Christopher Kruegel, Giovanni Vigna

SECLab
THE COMPUTER SECURITY GROUP AT UC SANTA BARBARA
Motivation
Available Solutions

- Dynamic
- Pure Static
- Static + Dynamic
What is Binary Reassembly?
Disassemble:

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

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

.text
mov [data_0], eax
jmp target...

.target mov [data_1], 1

.data
data_0 .long 0xc0deb4be
data_1 .long 0x0
Non-relocatable Assembly

BOOM!

Patch & Assemble

<table>
<thead>
<tr>
<th>.text</th>
<th>400100 mov [6000a0], eax</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>400105 jmp 40020d</td>
</tr>
<tr>
<td>40020d</td>
<td>CRASH!</td>
</tr>
<tr>
<td>40020f</td>
<td>mov [6000a4], 1</td>
</tr>
<tr>
<td>.data</td>
<td>6000a0 “cat\x00”</td>
</tr>
<tr>
<td></td>
<td>6000a4 .long 0x0</td>
</tr>
<tr>
<td></td>
<td>6000a8</td>
</tr>
</tbody>
</table>
mov [data_0], eax
jmp target
...

mov [CRASH!], 1

target mov [data_1], 1

.data

data_0 .long 0xc0deb4be
data_1 "cat\000"
data_0 .long 0xc0deb4be
data_1 .long 0x0

Relocatable Assembly

Patch & Assemble
Problems
Problem: Value Collisions

A Floating-point Variable $a$

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

Byte Representation

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

Interpreted as a Pointer

8060080 label_804ec3d

False Positives
Problem: Compiler Optimization

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

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

A code snippet allows constant folding
Problem: Compiler Optimization

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

A code snippet allows constant folding. Compiled in Clang with `-O1`.

False Negatives

\[0x804a034 - 'A' \times \text{sizeof(int)} = 0x8049f30\]
Our Approach
False Positives

False Negatives

Program Analysis

Heuristics

Ramblr
Pipeline

CFG
Recovery

Content Classification

Symbolization & Reassembly

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

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

CFG Recovery

Content Classification

Symbolization & Reassembly

0x804850b  Pointer
0xa      Integer
0xdc5    Integer
63 61 74 00  String
0x80484a2  Pointer
0x804840b  Pointer
0xa0000    Integer
push      offset label_34
push      offset label_35
cmp       eax, ecx
jne       label_42
.label_42:
mov       eax, 0x12fa9e5
...
CFG Recovery

Recursive Disassembly

Iterative Refinement

Example assembly code:

```
0x80486f0:
xor ebp, ebp
pop esi
mov ecx, esp
and esp, 0xfffffffff0
push eax
push esp
push esp
push edx
... 
```
Content Classification

A Typical Pointer

```
*(*(int*)0x8045010)
```

A Typical Value

```
((value * 42) ^ 5) / 3
```
## Content Classification

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

Data Types that Ramblr Recognizes
Content Classification

MOV· Scalar Double-precision floating-point value

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>movsd xmm0, ds:0x804d750</td>
<td></td>
</tr>
<tr>
<td>movsd xmm1, ds:0x804d758</td>
<td></td>
</tr>
</tbody>
</table>

Recognizing Types during CFG Recovery

Two floating-points

<table>
<thead>
<tr>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>804d750</td>
<td>Floating point integer</td>
</tr>
<tr>
<td>804d758</td>
<td>Floating point integer</td>
</tr>
</tbody>
</table>
Content Classification

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

Recognizing Types with Slicing & VSA
if(i > 3)
jmp table[i * 4]

Quit switch

\[
i = [0, 2] \text{ with a stride of 1}
\]

<table>
<thead>
<tr>
<th>A jump table of 3 entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>table[0]</td>
</tr>
<tr>
<td>table[1]</td>
</tr>
<tr>
<td>table[2]</td>
</tr>
</tbody>
</table>

Recognizing Types with Slicing & VSA
Base Pointer Reattribution

```c
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]

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

False Negatives
Safety Heuristics: Data Consumer Check

Unusual Behaviors Triggering the Opt-out Rule
Symbolization & Reassembly

<table>
<thead>
<tr>
<th>Address</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x400010</td>
<td>label_34</td>
</tr>
<tr>
<td>0x400020</td>
<td>label_35</td>
</tr>
<tr>
<td>0x400a14</td>
<td>label_42</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>0x406000</td>
<td>data_3</td>
</tr>
</tbody>
</table>

Symbolization

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Address</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>push offset label_34</td>
<td>0x400010</td>
<td></td>
</tr>
<tr>
<td>push offset label_35</td>
<td>0x400020</td>
<td></td>
</tr>
<tr>
<td>cmp eax, ecx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>jne label_42</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.label_42:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mov eax, 0x12fa9e5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Assembly Generation
Evaluation
## Data sets

<table>
<thead>
<tr>
<th></th>
<th>Coreutils 8.25.55</th>
<th>Binaries from CGC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programs</td>
<td>106</td>
<td>143</td>
</tr>
<tr>
<td>Compiler</td>
<td>CGC 5</td>
<td>Clang 4.4</td>
</tr>
<tr>
<td>Optimization levels</td>
<td>O0/O1/O2/O3/Os/Ofast</td>
<td></td>
</tr>
<tr>
<td>Architectures</td>
<td>X86/AMD64</td>
<td>X86</td>
</tr>
<tr>
<td>Test cases</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Total binaries</td>
<td>1272</td>
<td>725</td>
</tr>
</tbody>
</table>
Brief Results: Success Rate

- Coreutils O0
- Coreutils O1
- CGC O0
- CGC O1
- CGC O2
- CGC O3

Uroboros, Ramblr, Ramblr Fast
Ramblr is the foundation of ...

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

Patcherex

Patching support in angr Management
HOW CAN I REASSEMBLE BINARIES?
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