Differential Slicing: Identifying Causal Execution Differences for Security Applications

Noah M. Johnson\textsuperscript{1}, Juan Caballero\textsuperscript{2}, Kevin Zhijie Chen\textsuperscript{1}, Stephen McCamant\textsuperscript{1}, Pongsin Poosankam\textsuperscript{1,3}, Daniel Reynaud\textsuperscript{1}, and Dawn Song\textsuperscript{1}

\textsuperscript{1}University of California, Berkeley
\textsuperscript{2}IMDEA Software Institute
\textsuperscript{3}Carnegie Mellon University
Outline

• Introduction
• Problem Definition and Overview
• Trace Alignment
• Slice-Align
• Evaluation
• Related Work
• Conclusion
Introduction

• Why does the program crash?

• At what situation does the malware do malicious behaviors?

• How do you solve above problems if you don’t have the source code?
  • Static analysis
  • Dynamic analysis
  • ...
  • Too much time spent

• Static analysis
• Dynamic analysis
• ...
Introduction

• This paper, proposes “Differential Slicing”
  • Given 2 execution traces of a program with a target difference
  • Automatically finds the input and environment differences that caused the target difference
  • Generates a causal difference graph
    • Simply expressed what happened
Problem Definition and Overview

- The goal is to “understand” the target difference
  - To identify the input differences that caused the target difference.
  - To understand the sequence of events that led from the input differences to the target difference.

⇒ To build the causal difference graph
Problem Definition and Overview

Passing trace
$ \texttt{vuln\_cmp\ bar\ bazaar}$
Strings are not equal

Failing trace
$ \texttt{vuln\_cmp\ "\"\ foo}$
<<crashed at line 11>>

Then the passing trace and the failing trace can be used for Trace Alignment.

Figure 1: Motivating example program, vuln\_cmp.c.
Figure 1: Motivating example
Divergence point

Flow difference

Value difference

Flow differences = disaligned statements

Passing run

/* argc = 3 */
3: if(argc<3)
/* argv[1] = "bar" */
V 5: int len1 = strlen(argv[1])
/* argv[2] = "bazaar" */
V 6: int len2 = strlen(argv[2])
/* len1 = 3 */
V 7: if (len1)

/* len1 = 3 */
8: s1 = (char *)malloc(len1)
V 9: if (len2)
/* len2 = 6 */
V 10: s2 = (char *)malloc(len2)
/* s1 = (ptr to 3-byte buffer), argv[1] = "bar", len1 = 3 */
V 11: strncpy(s1, argv[1], len1)

/* s1 = (ptr to 6-byte buffer), argv[2] = "bazaar", len2 = 6 */
F 12: strncpy(s2, argv[2], len2)

F 15: return 0

Failing run

/* argc = 3 */
3: if(argc<3)
/* argv[1] = "" */
5: int len1 = strlen(argv[1])
/* argv[2] = "foo" */
6: int len2 = strlen(argv[2])
/* len1 = 0 */
7: if (len1)

/* len2 = 3 */
9: if (len2)
/* len2 = 3 */
10: s2 = (char *)malloc(len2)
/* s1 = NULL, argv[1] = "", len1 = 0 */
11: strncpy(s1, argv[1], len1)

crash
Problem Definition and Overview

• Causal difference graph
  • The causal difference graph contains the sequences of execution differences leading from the input differences to the target differences.
Problem Definition and Overview

- 6k lines of **Objective Caml** code
  - Trace alignment and post-dominator module: 4k lines
  - Slice-Align module: 2k lines

Figure 4: System architecture. The darker box was previously available.
Trace Alignment

- **Dominate**
  - A node $d$ dominates node $n$ iff every path from entry node to $n$ passes through $d$. (node $d$ is a dominator of node $n$)
  - Node $id$ immediately dominates $n$ if $id$ dominates $n$, and no other node $p$ such that $id$ dominates $p$ and $p$ dominates $n$. ($id$ is the only immediate dominator of $n$)

- **Post Dominate**
  - Same as dominate, from node $n$ to the exit node
  - Immediate post dominator
Execution Indexing

- Execution Indexing captures the structure of the program at any given point in the execution, identifying the execution point, and uses that structure to establish a correspondence between execution points across multiple executions of the program.
- Xin et al. use an indexing stack to deal with branch or method call.
Trace Alignment

Input: $A_0, A_1$ // anchor points
Output: $RL$ // list of aligned and disaligned regions
$EI_0, EI_1$: execution index stacks $\leftarrow$ Stack.empty();
$insn_0, insn_1 \leftarrow A_0, A_1$; // current instructions
$RL \leftarrow \emptyset$;

while $insn_0, insn_1 \neq \bot$ do
  $cr \leftarrow \text{regionBegin}(insn_0, insn_1, \text{aligned})$
  // Aligned-Loop: Traces aligned. Walk until disaligned
  while $EI_0 = EI_1$ do
    foreach $i \in 0, 1$ do
      $EI_i \leftarrow \text{updateIndex}(EI_i, insn_i)$;
      $cr \leftarrow \text{regionExtend}(insn_i, cr)$;
      $insn_i++$;
    end
  end
  $RL \leftarrow RL \cup cr$;
  $cr \leftarrow \text{regionBegin}(insn_0, insn_1, \text{disaligned})$
  // Disaligned-Loop: Traces disaligned. Walk until realigned
  while $EI_0 \neq EI_1$ do
    while $|EI_0| \neq |EI_1|$ do
      $j \leftarrow (|EI_0| > |EI_1|) ? 0 : 1$;
      while $|EI_j| \geq |EI_{1-j}|$ do
        $EI_j \leftarrow \text{updateIndex}(EI_j, insn_j)$;
        $cr \leftarrow \text{regionExtend}(insn_j, cr)$;
        $insn_j++$;
      end
    end
  end
  $RL \leftarrow RL \cup cr$;
end

Figure 5: Algorithm for trace alignment.
Slice-Align

- worklist
  - A pool of instructions to be operated

Figure 6: Algorithm for Basic graph.
Slice-Align

```
worklist

11: strncpy(s1, argv[1], len1);
8: s1 = (char*)malloc(len1)
5: int len1 = strlen(argv[1]);

argv[1]
7: if (len1) ...
```

Input difference

---

**Execution Omission**

8: s1 = (char*)malloc(len1);

**Passing run**

- argv[1] = "bar"
- len1 = 3

**Divergence Point**

```
7: if (len1) ...
```

**Failing run**

- argv[1] = ""
- len1 = 0

```
11: strncpy(s1, argv[1], len1);
```

s1 = (valid address)

**Execution Omission**

s1 = NULL
**Slice-Align**

- **Edge pruning and address normalization**
  - Pruning edges in the graph when an operand of an aligned instruction has the same value in both execution traces.

- **Heap pointer pruning**
  - The pointer is pruned if
    1. The allocation site for the live buffers that contain the pointed-to addresses are aligned
    2. The offset of those pointed-to addresses, with respect to the start address of the live buffer they belong to, is the same

- **Stack pointer pruning**
  - (in the thread stack range) normalized by subtracting the stack base address

- **Data section pointer pruning**
  - (in the same module) normalized by subtracting the module base address
Table II: Programs and vulnerabilities in the evaluation.
Evaluation

- Evaluating the Causal Difference Graph

<table>
<thead>
<tr>
<th>Name</th>
<th>Total instructions</th>
<th>Disaligned instructions</th>
<th>Disaligned regions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Passing</td>
<td>Failing</td>
<td>Passing</td>
</tr>
<tr>
<td>reader-e1</td>
<td>2,800,163</td>
<td>1,819,714</td>
<td>1,307,465</td>
</tr>
<tr>
<td>reader-e2</td>
<td>1,616,642</td>
<td>1,173,531</td>
<td>446,273</td>
</tr>
<tr>
<td>reader-u1</td>
<td>2,430,400</td>
<td>1,436,993</td>
<td>2,034,582</td>
</tr>
<tr>
<td>reader-u2</td>
<td>1,921,514</td>
<td>1,053,840</td>
<td>656,183</td>
</tr>
<tr>
<td>reader-u10</td>
<td>408,618</td>
<td>272,994</td>
<td>144,517</td>
</tr>
<tr>
<td>reader-u11</td>
<td>1,868,942</td>
<td>1,112,828</td>
<td>1,504,189</td>
</tr>
<tr>
<td>reader-u14</td>
<td>1,194,053</td>
<td>155,906</td>
<td>601,789</td>
</tr>
<tr>
<td>tftp</td>
<td>626,622</td>
<td>350,323</td>
<td>415,086</td>
</tr>
<tr>
<td>firebird</td>
<td>6,698</td>
<td>1,282</td>
<td>5,551</td>
</tr>
<tr>
<td>gdi-2008</td>
<td>42,124</td>
<td>4,310</td>
<td>38,743</td>
</tr>
<tr>
<td>gdi-2007</td>
<td>36,792</td>
<td>4,310</td>
<td>33,508</td>
</tr>
</tbody>
</table>

Table III: Total disaligned instructions and regions compared with disaligned regions in graph.
Evaluation

- Graph size
  - \#IDiff = number of input differences

<table>
<thead>
<tr>
<th>Name</th>
<th>Basic pruning</th>
<th></th>
<th>Extended pruning</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pass</td>
<td>Fail</td>
<td># IDiff</td>
<td>Pass</td>
</tr>
<tr>
<td>reader-e1</td>
<td>3,651</td>
<td>3,616</td>
<td>7</td>
<td>2,324</td>
</tr>
<tr>
<td>reader-e2</td>
<td>4,854</td>
<td>4,853</td>
<td>21</td>
<td>81</td>
</tr>
<tr>
<td>reader-u1</td>
<td>2,753</td>
<td>2,751</td>
<td>13</td>
<td>204</td>
</tr>
<tr>
<td>reader-u2</td>
<td>135</td>
<td>135</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>reader-u10</td>
<td>45</td>
<td>43</td>
<td>1</td>
<td>36</td>
</tr>
<tr>
<td>reader-u11</td>
<td>1,584</td>
<td>1,562</td>
<td>1</td>
<td>1,158</td>
</tr>
<tr>
<td>reader-u14</td>
<td>1,714</td>
<td>1,695</td>
<td>6</td>
<td>425</td>
</tr>
<tr>
<td>tftp</td>
<td>254</td>
<td>254</td>
<td>1</td>
<td>254</td>
</tr>
<tr>
<td>firebird</td>
<td>45</td>
<td>46</td>
<td>1</td>
<td>45</td>
</tr>
<tr>
<td>gdi-2008</td>
<td>100</td>
<td>101</td>
<td>1</td>
<td>96</td>
</tr>
<tr>
<td>gdi-2007</td>
<td>11</td>
<td>12</td>
<td>1</td>
<td>7</td>
</tr>
</tbody>
</table>
evaluation

- Performance
  - Less than 1 hour to generate a graph

<table>
<thead>
<tr>
<th>Name</th>
<th>Trace size (MB)</th>
<th>Tracing (sec.)</th>
<th>Trace align (sec.)</th>
<th>Slice-Align (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pass Fail</td>
<td>Pass Fail</td>
<td></td>
<td></td>
</tr>
<tr>
<td>reader-e1</td>
<td>202 106</td>
<td>482 365</td>
<td>1,684</td>
<td>3,510</td>
</tr>
<tr>
<td>reader-e2</td>
<td>143 67</td>
<td>345 337</td>
<td>1,180</td>
<td>1,291</td>
</tr>
<tr>
<td>reader-u1</td>
<td>200 133</td>
<td>403 406</td>
<td>714</td>
<td>101</td>
</tr>
<tr>
<td>reader-u2</td>
<td>110 61</td>
<td>208 295</td>
<td>152</td>
<td>208</td>
</tr>
<tr>
<td>reader-u10</td>
<td>24 16</td>
<td>267 275</td>
<td>39</td>
<td>24</td>
</tr>
<tr>
<td>reader-u11</td>
<td>152 101</td>
<td>155 161</td>
<td>462</td>
<td>364</td>
</tr>
<tr>
<td>reader-u14</td>
<td>160 107</td>
<td>195 192</td>
<td>837</td>
<td>239</td>
</tr>
<tr>
<td>tftpd</td>
<td>3.6 2.0</td>
<td>13 12</td>
<td>50</td>
<td>12</td>
</tr>
<tr>
<td>firebird</td>
<td>2.5 0.1</td>
<td>1 1</td>
<td>1</td>
<td>0.2</td>
</tr>
<tr>
<td>gdi-2008</td>
<td>2.4 0.4</td>
<td>2 0.8</td>
<td>2</td>
<td>0.5</td>
</tr>
<tr>
<td>gdi-2007</td>
<td>2.1 0.4</td>
<td>2 0.8</td>
<td>2</td>
<td>0.3</td>
</tr>
</tbody>
</table>

Table V: Performance evaluation.
Evaluation

• User Study (informal)
  • Subject A: an analyst at a commercial security research company
  • Subject B: a research scientist

<table>
<thead>
<tr>
<th>Subj.</th>
<th>Sample 1 (no graph)</th>
<th>Sample 2 (Causal difference graph)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>sample</td>
<td>time (hr)</td>
</tr>
<tr>
<td>A</td>
<td>reader-e2</td>
<td>13</td>
</tr>
<tr>
<td>B</td>
<td>reader-u10</td>
<td>3</td>
</tr>
</tbody>
</table>

Table VI: Results for user study.
Evaluation

- Identifying input differences in malware analysis
- **W32/Conficker.A**
  - Keyboard layout: Ukrainian (failing trace), US-English (passing trace)
  - Target difference: `CreateThread` API call
  - Result:
    - Input difference: `user32.dll::GetKeyboardLayoutList` function return value
- **W32/Netsky.C**
  - Makes the computer speaker beep continuously if the system time between 6am and 9pm on Feb. 26, 2004
  - Target Difference: Beep function call
  - Result:
    - Input difference: `kernel32.dll::GetLocalTime` system call
Conclusion

- Producing causal difference graph
  - Input difference information
  - Execution difference from input difference to target difference
- Reducing the graph size
- Reducing the input difference candidates