APICraft: Fuzz Driver Generation for Closed-source SDK Libraries

Cen Zhang, Xingwei Lin, Yuekang Li, Yinxing Xue, Jundong Xie, Hongxu Chen, Xinlei Ying, Jiashui Wang, Yang Liu
Motivation

- Closed-source SDKs contain huge attack surfaces
- Fuzzing them requires manual efforts to write fuzz drivers
- Existing methods cannot be applied for their fuzz drivers’ generation
Challenges

- Limited information can be used in binary-only scenario

- Semantic correctness of these complex API usages has to be kept
Practical Example

```
1 DataProvider* prov = ProviderCreateWithData(data);
2 Font* font = ExtractFont(prov);
3 DoubleLeadingSpace(font);

1 FontDescriptor* desc = CreateFontDescriptor(data);
2 Font* font = ExtractFont(desc);
3 CalcLeadingSpace(font);
```

Consumer 1

a) Directly Carved

b) Crossover

c) Desired
Collect & Combine Approach
Approach Detail – Data Dependency

➢ Extraction:

```
Algorithm 1 Basic Data Dependency Extraction

Input: T (An API function trace)
Output: R (Data dependency set)
1: R ← ∅
2: cache ← {}
3: for FB ∈ T
4:   for In ∈ IFB
5:     for (FA, Out) ∈ cache[In.value]
6:       if Out.type type In.type
7:         R ← (FA, Out, FB, In)
8: for Out ∈ OFB
9:   if Out.value ≠ 0
10:   cache ← {Out.value : (FB, Out)}
```

➢ Inference:

R1: Dependency-based transition
Fa.out => Fc.in
Fa.out => Fd.in => Fb.out => Fd.in
Fb.out => Fc.in

R2: Type-based transition
Fa.out type is T
Fb.in type is T => Fa.out => Fb.in

R3: Inter-thread data flow dependency
Approach Detail – Control Dependency

Error Handling Information:

- Pointer type output of a function
  Naive null pointer check

- Integer type output of a function
  A taint-based analysis to extract its check condition
Approach Detail – Combination – 1

Combine data dependencies via a multi-objective genetic algorithm (NSGA-II)

- Diversity (DIV)

  \[ DIV = E + CC \]  \hspace{2cm} (1)  

  \[ S_{\text{eff}}(b) = \begin{cases} 
    3 & \text{if basic block } b \text{ has call and in loop} \\
    2 & \text{if basic block } b \text{ either has call or in loop} \\
    1 & \text{otherwise} 
  \end{cases} \]

- Effectiveness (EFF)

  \[ EFF = \sum_{b \in B} S_{\text{eff}}(b) \]  \hspace{2cm} (2)  

  \[ S_{\text{comp}}(i) = \begin{cases} 
    2 & \text{if } i \text{ is in core dependency} \\
    1 - \frac{\text{min}(k,5)}{5} & \text{if } i \text{ is in non-core dependency} \\
    0 & \text{if } S_{\text{comp}}(i) \text{ has been counted} \\
    1 & \text{otherwise} 
  \end{cases} \]

- Compactness (COMP)

  \[ COMP = \frac{\sum_{f \in F} \sum_{i \in I_f} S_{\text{comp}}(i) - (F_{\text{num}} - 1)}{\sum_{f \in F} \sum_{i \in I_f} 1} \]  \hspace{2cm} (3)
Approach Detail – Combination – 2

Improving the stability of generated fuzz drivers:

- **Stability Test**
  Smoke testing with sanitizers under 3 – 5 distinct inputs

- **Error Handling**
  Generated from control dependencies
Evaluation – Applying to Real-world SDK

**MacOS SDK**
- Image, Font, PDF, Audio, RTF

**MacOS Applications**
- Preview, Safari, Messages, Mail, Quicktime Player, …

<table>
<thead>
<tr>
<th>Attack Surface</th>
<th>Tracer LoC</th>
<th>Trace Size</th>
<th># of APIs</th>
<th>Time (min)</th>
<th>Pre-Processing</th>
<th>Dependency Collection</th>
<th>Dependency Combination</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R1</td>
<td>R2</td>
<td>R3</td>
<td># of Data Deps</td>
<td># of Control Deps</td>
<td>Time (min)</td>
<td>Initial Score (EFF/DIV/COMP)</td>
</tr>
<tr>
<td>Image</td>
<td>26,775</td>
<td>2.90 GB</td>
<td>540</td>
<td>125</td>
<td>870</td>
<td>232</td>
<td>(124+0)/(124+5)</td>
</tr>
<tr>
<td>Font</td>
<td>33,904</td>
<td>7.70 GB</td>
<td>689</td>
<td>180</td>
<td>16,556</td>
<td>320</td>
<td>(60+0)/(60+5)</td>
</tr>
<tr>
<td>PDF</td>
<td>29,356</td>
<td>1.60 GB</td>
<td>595</td>
<td>95</td>
<td>908</td>
<td>233</td>
<td>(117+0)/(117+6)</td>
</tr>
<tr>
<td>Audio</td>
<td>18,822</td>
<td>0.13 GB</td>
<td>345</td>
<td>58</td>
<td>107</td>
<td>32</td>
<td>(2+68)/(2+68)</td>
</tr>
<tr>
<td>RTF</td>
<td>10,442</td>
<td>0.41 GB</td>
<td>191</td>
<td>15</td>
<td>40</td>
<td>24</td>
<td>(30+0)/(30+0)</td>
</tr>
</tbody>
</table>

APICraft can apply to real world closed-source SDKs like MacOS SDK.
Evaluation – Performance Comparison

In most targets, APICraft outperforms manually written fuzz drivers in metrics of coverage and crash with statistical significance.
Evaluation – Effectiveness of Each Component

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Font</th>
<th>PDF</th>
<th>Audio</th>
<th>RTF</th>
<th>Avg. Pct.</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>33/110</td>
<td>55/69</td>
<td>23/87</td>
<td>63/137</td>
<td>12/28</td>
<td>43%</td>
</tr>
<tr>
<td>R2</td>
<td>33/110</td>
<td>8/69</td>
<td>24/87</td>
<td>65/137</td>
<td>12/28</td>
<td>32%</td>
</tr>
<tr>
<td>R3</td>
<td>32/110</td>
<td>6/69</td>
<td>23/87</td>
<td>15/137</td>
<td>3/28</td>
<td>18%</td>
</tr>
<tr>
<td>R*</td>
<td>66/110</td>
<td>64/69</td>
<td>44/87</td>
<td>79/137</td>
<td>15/28</td>
<td>62%</td>
</tr>
</tbody>
</table>

Each component of APICraft significantly contributes to the overall performance.

Statistics of Used Dependencies from Inference Rules

Coverage Per Time

(a) Image  (b) Font  (c) PDF  (d) Audio  (e) RTF
Evaluation – Real World Fuzzing

- 8 months’ long-term fuzzing
- 142 unique vulnerabilities with 54 CVEs
- Affects popular Apple products and several OSes
- All found bugs are confirmed with fixed by vendor
Q & A

Cen Zhang
cen001@e.ntu.edu.sg