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

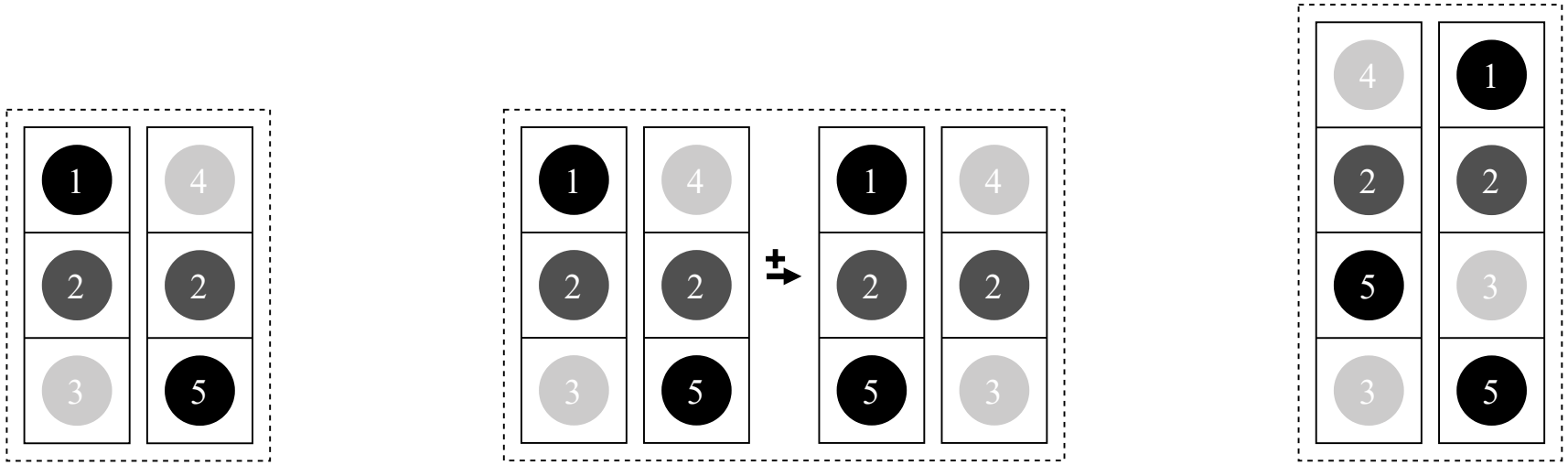
Practical Example

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

Consumer 1

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

Consumer 2

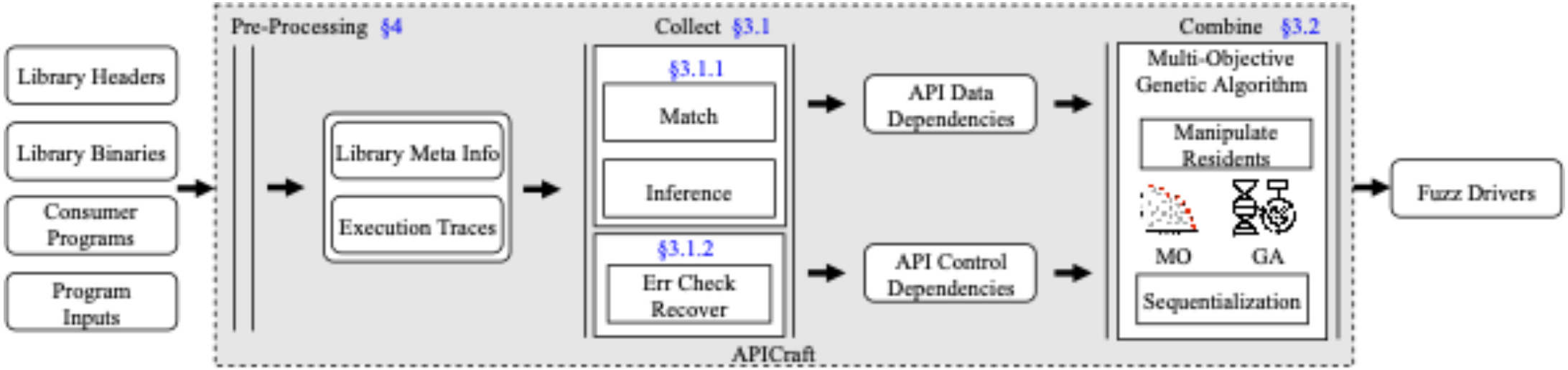


a) Directly Carved

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)

```
R ← ∅
cache ← { }
for F_B ∈ T
    for In ∈ I_{F_B}
        for ⟨F_A, Out⟩ ∈ cache[In.value]
            if Out.type =^{type} In.type
                R ←^{+} ⟨F_A, Out, F_B, In⟩
    for Out ∈ O_{F_B}
        if Out.value ≠ 0
            cache ←^{+} {Out.value : ⟨F_B, 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)
- Effectiveness (EFF)
- Compactness (COMP)

$$DIV = E + CC \tag{1}$$

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

$$EFF = \sum_{b \in B} S_{eff}(b) \tag{2}$$

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

$$COMP = \frac{\sum_{f \in F} \sum_{i \in I_f} S_{comp}(i) - (F_{num} - 1)}{\sum_{f \in F} \sum_{i \in I_f} 1} \tag{3}$$

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

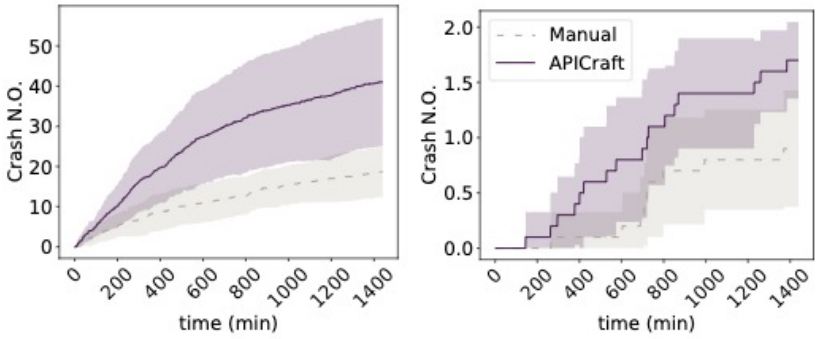
- Preview, Safari, Messages, Mail, Quicktime Player, …

Attack Surface	Pre-Processing				Dependency Collection						Dependency Combination		
	Tracer LoC	Trace Size	# of APIs	Time (min)	R1	R2	R3	# of Data Deps	# of Control Deps	Time (min)	Initial Score (EFF/DIV/COMP)	Final Score (EFF/DIV/COMP)	Time (min)
Image	26,775	2.90 GB	540	125	870	840	232	56,632	(124+0)/(124+5)	1035	23,211/ 5.30/ 1.07	32,795/ 43.30/ 1.06	1,075
Font	33,904	7.70 GB	689	180	16,556	1,350	320	192,388	(60+0)/(60+5)	1643	18,782/ 7.10/ 1.06	26,391/ 33.60/ 1.16	534
PDF	29,356	1.60 GB	595	95	908	905	233	66,689	(117+0)/(117+6)	371	13,214/ 6.00/ 0.98	19,080/ 43.50/ 1.04	484
Audio	18,822	0.13 GB	345	58	107	116	32	11,422	(2+68)/(2+68)	89	11,603/ 6.50/ 1.06	13,061/ 92.00/ 1.06	857
RTF	10,442	0.41 GB	191	15	40	40	24	1,396	(30+0)/(30+0)	25	43,721/ 3.00/ 1.00	45,001/ 13.40/ 0.96	723

APICraft can apply to real world closed-source SDKs like MacOS SDK.

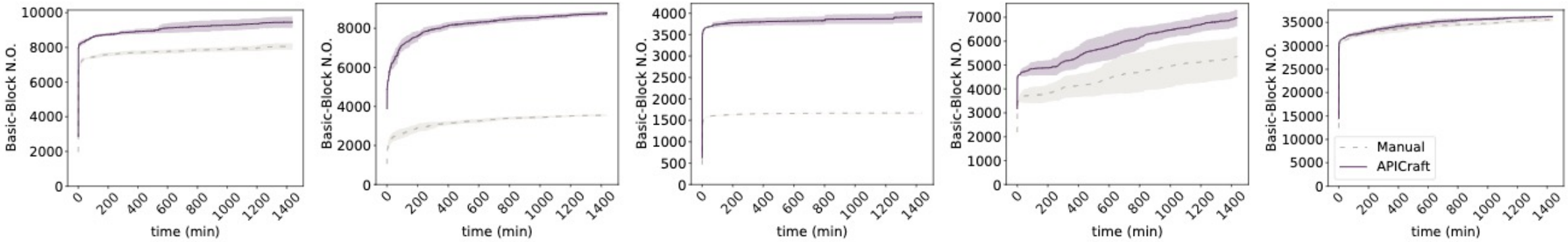
Evaluation – Performance Comparison

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



(a) Audio (b) Font

Crash Per Time



(a) Image (b) Font (c) PDF (d) Audio (e) RTF

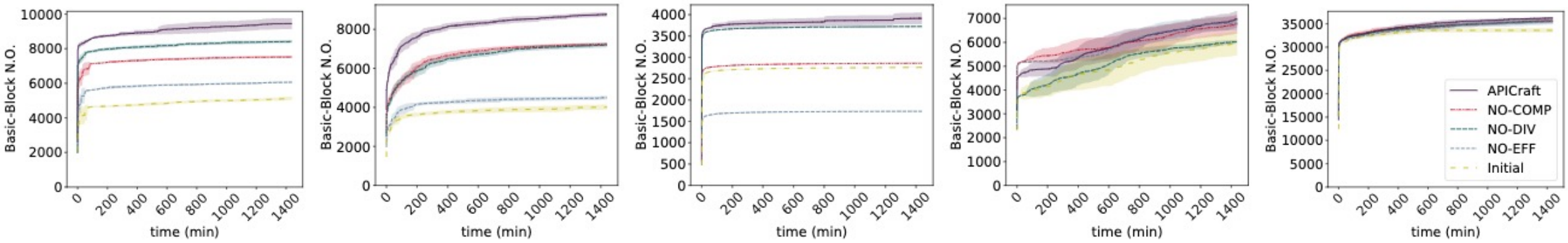
Coverage Per Time

Evaluation – Effectiveness of Each Component

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

Statistics of Used Dependencies from Inferrence Rules

Each component of APICraft significantly contributes to the overall performance.



(a) Image (b) Font (c) PDF (d) Audio (e) RTF

Coverage Per Time

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