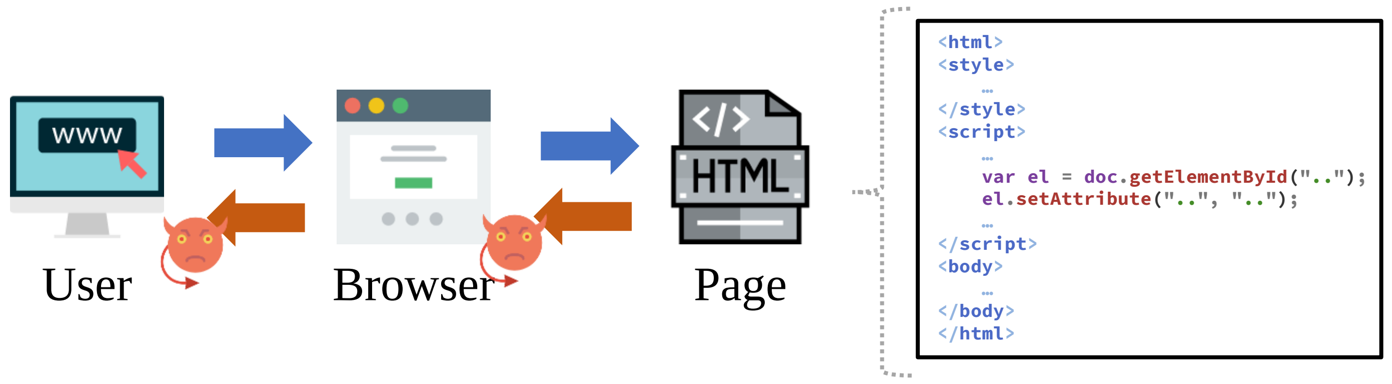
Minerva: Browser API Fuzzing with Dynamic Mod-Ref Analysis

Chijin Zhou[1], Quan Zhang[1], Mingzhe Wang[1],

Lihua Guo[1], Jie Liang[1], Zhe Liu[2], Mathias Payer[3], Yu Jiang[1]

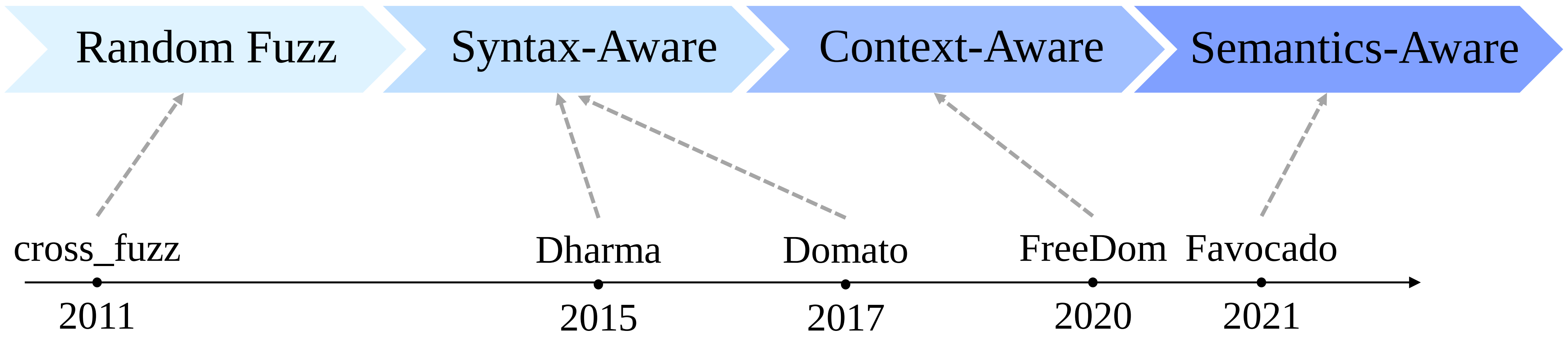
[1]Tsinghua University, Beijing, China
[2]NUAA, Nanjing, China
[3]EPFL, Lausanne, Switzerland

Browser Fuzzing: A Decade of Research

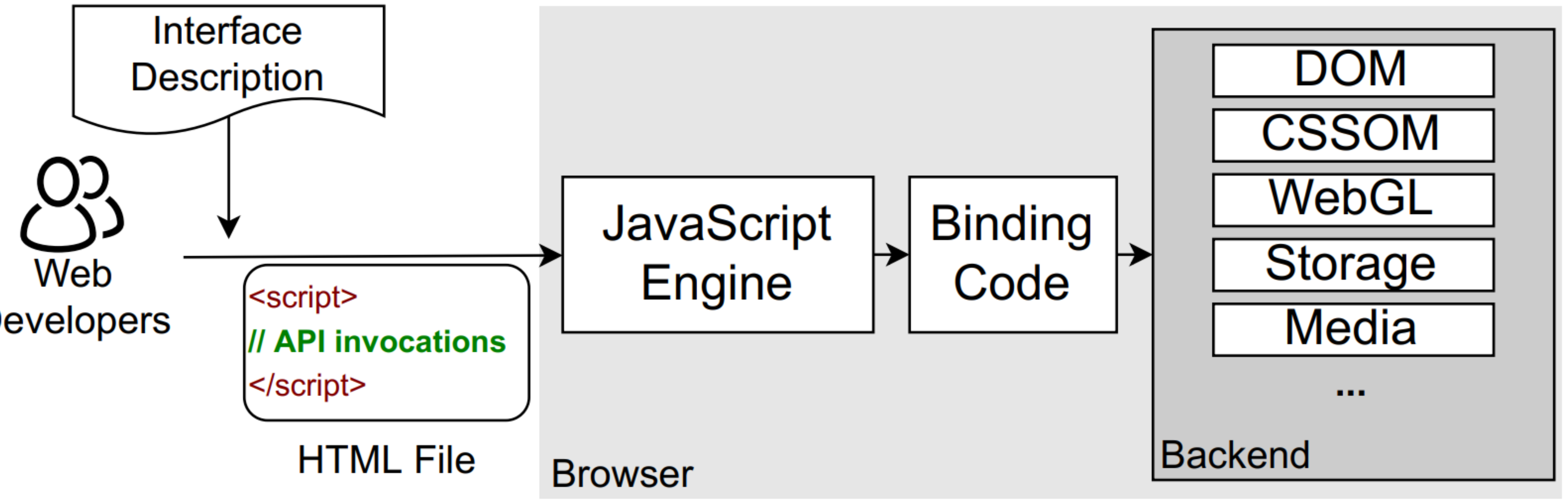


```
<html>
<style>
  …
</style>
<script>
  …
    var el = doc.getElementById("..");
    el.setAttribute("..", "..");
  …
</script>
<body>
  …
</body>
</html>
```

Goal of fuzzers: Generate html files that explore **browser states** and, with luck, trigger **memory corruptions**.



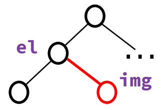
Browser APIs: Manipulating Browser State Transitions



Browser APIs: Manipulating Browser State Transitions

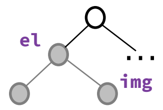
API invocations **Browser's backend logic**

```
el.appendChild(img)
```



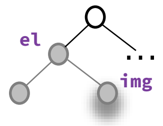
update the rendering tree
recalculate the layouts and geometries of all elements

```
el.style.visibility = "hidden"
```

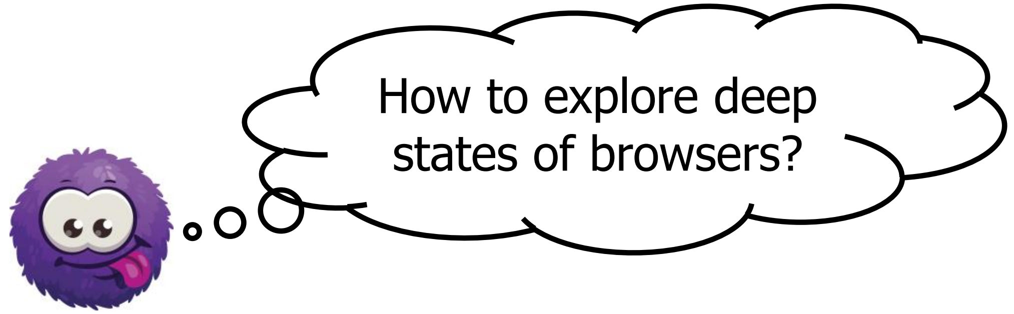


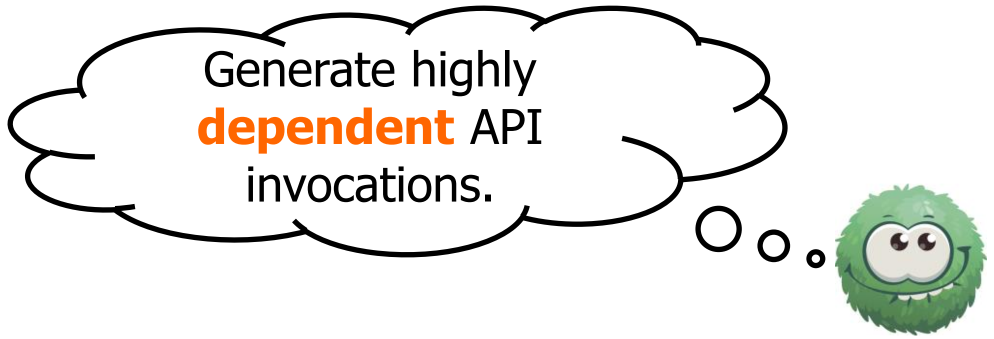
inform others of style changes
repaint relevant elements

```
img.height = 10
```



recalculate the layouts and geometries of all elements





Type Relation v.s. Memory Relation

Would the two APIs
interact with each other?

```
void SVGSVGElement::setCurrentTime(float)

void SVGSVGElement::setCurrentScale(float)
```

From type relation view:

Yes, they would. Because they belong to the same element object and accept identical input types.

From memory relation view:

No, they would not. Because they do not access any common memory locations when handling their backend logic.

Type Relation v.s. Memory Relation

Would the two APIs
interact with each other?

```
void SVGSVGElement::setCurrentTime(float)
```

Why is the memory relation important?

Because putting two memory-irrelevant APIs into a test case will not trigger any interesting behaviors.

```
void SVGSVGElement::setCurrentScale(float)
```

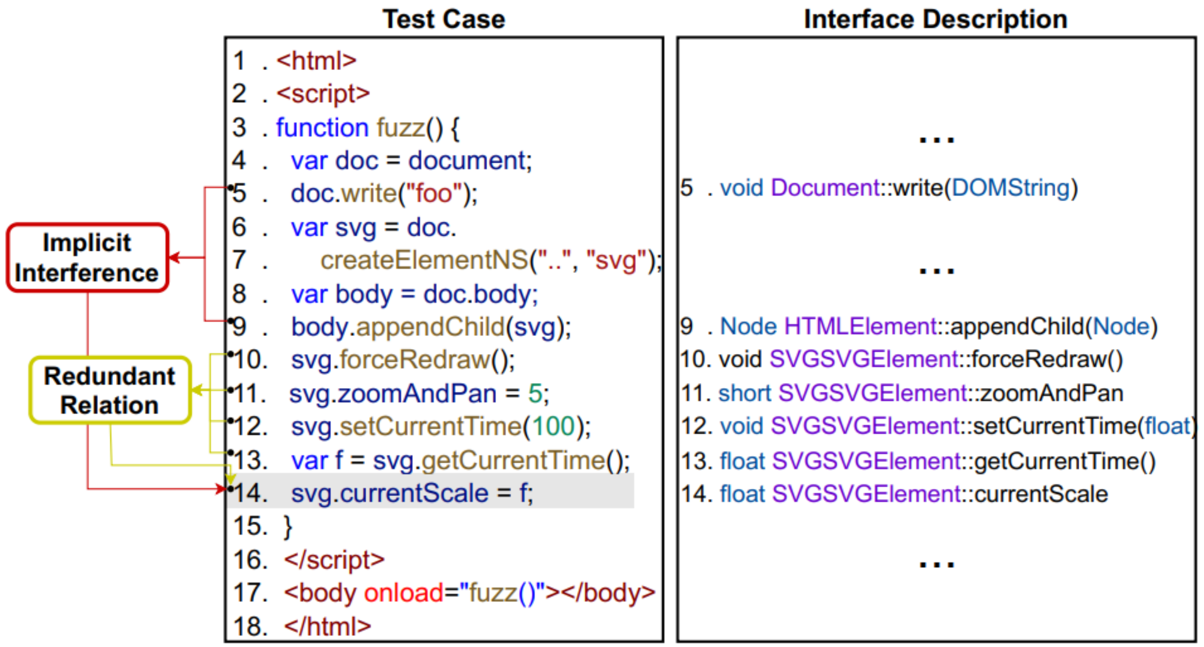
From type relation view:

Yes, they would. Because they belong to the same element object and accept identical input types.

From memory relation view:

No, they would not. Because they do not access any common memory locations when handling their backend logic.

Motivating Example



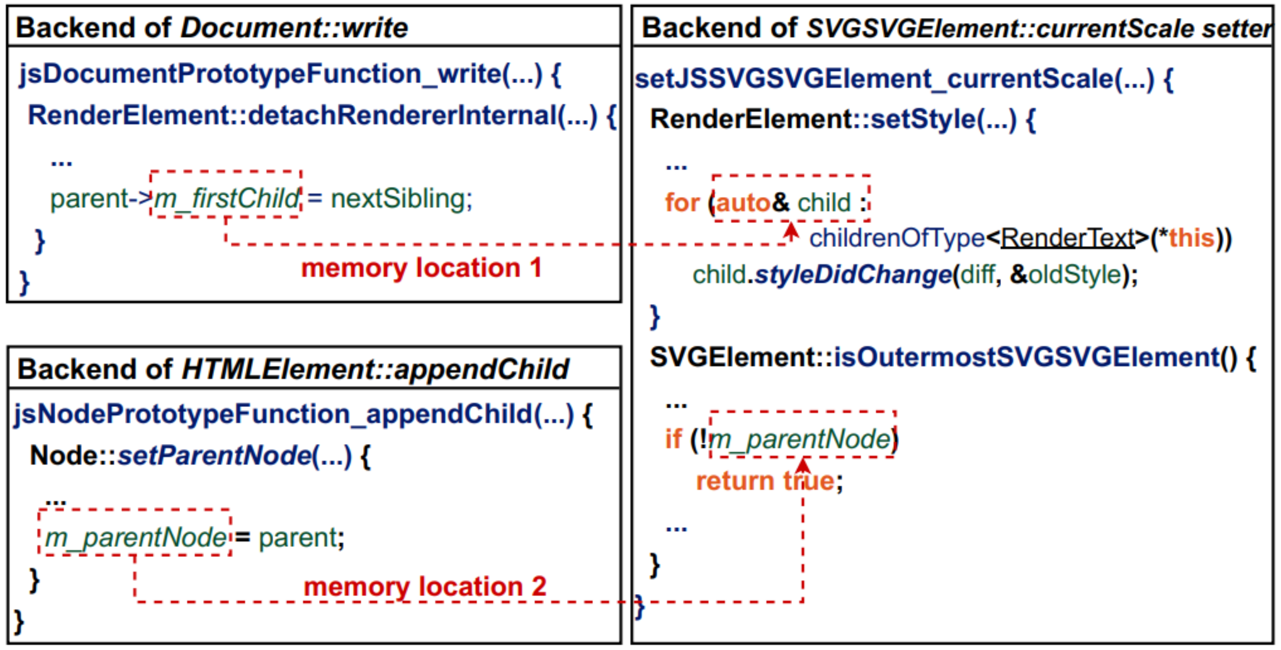
Test Case

```
1 . <html>
2 . <script>
3 . function fuzz() {
4 .   var doc = document;
5 .   doc.write("foo");
6 .   var svg = doc.
7 .     createElementNS("..", "svg");
8 .   var body = doc.body;
9 .   body.appendChild(svg);
10.   svg.forceRedraw();
11.   svg.zoomAndPan = 5;
12.   svg.setCurrentTime(100);
13.   var f = svg.getCurrentTime();
14.   svg.currentScale = f;
15. }
16. </script>
17. <body onload="fuzz()"></body>
18. </html>
```

Interface Description

```
...
5 . void Document::write(DOMString)
...
9 . Node HTMLElement::appendChild(Node)
10. void SVGSVGElement::forceRedraw()
11. short SVGSVGElement::zoomAndPan
12. void SVGSVGElement::setCurrentTime(float)
13. float SVGSVGElement::getCurrentTime()
14. float SVGSVGElement::currentScale
...
```

(a) A test case and its corresponding interface descriptions



Backend of *Document::write*

```
jsDocumentPrototypeFunction_write(...) {
 RenderElement::detachRendererInternal(...) {
   ...
   parent->m_firstChild = nextSibling;
 }
}
```

Backend of *HTMLElement::appendChild*

```
jsNodePrototypeFunction_appendChild(...) {
 Node::setParentNode(...) {
   ...
  m_parentNode = parent;
 }
}
```

Backend of *SVGSVGElement::currentScale setter*

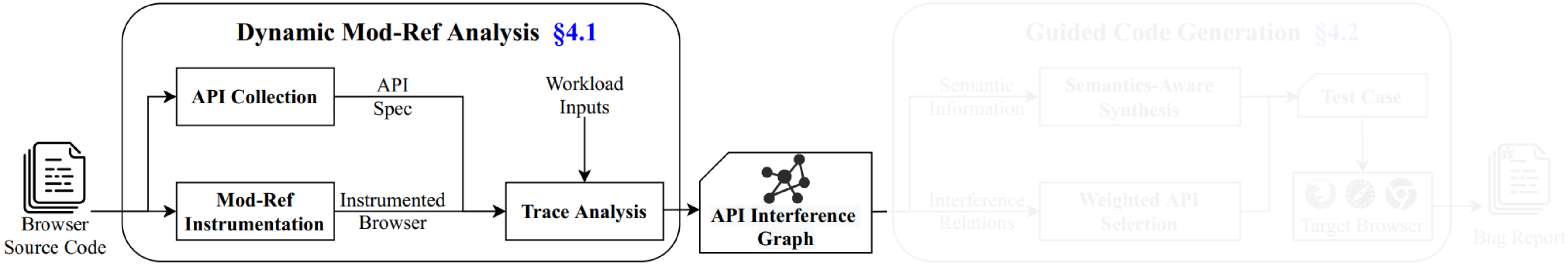
```
setJSSVGSVGElement_currentScale(...) {
 RenderElement::setStyle(...) {
   ...
   for (auto& child :
        childrenOfType<RenderText>(*this))
      child.styleDidChange(diff, &oldStyle);
 }
 SVGElement::isOutermostSVGSVGElement() {
   ...
   if (!m_parentNode)
     return true;
   ...
 }
}
```

(b) Memory access interference hidden in browser's backend logic

Observation: Memory mod-ref relations are implicitly present in API combinations

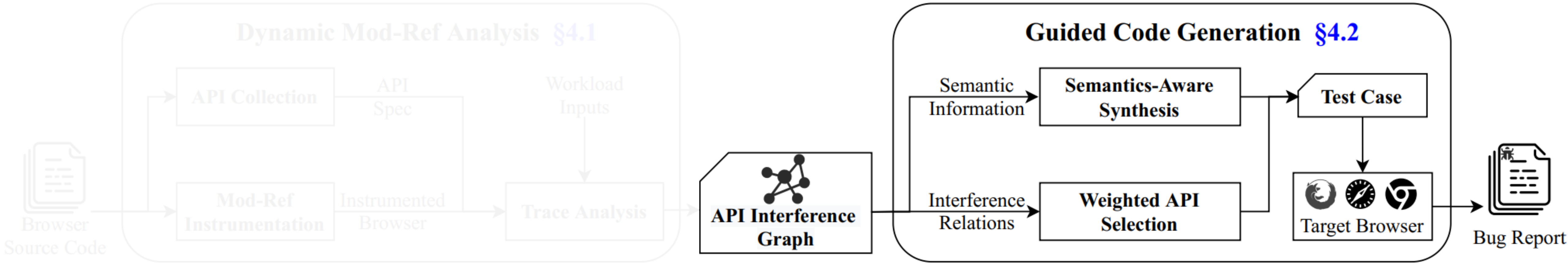
Minerva: A New Solution for Browser Fuzzing

- Insight: Fuzzing guided by memory mod-ref relations of APIs
- A two-stage design:
  1. **Analyze the mod-ref relations through dynamic traces during preparation**
  2. Select highly relevant APIs based on the relations during fuzzing

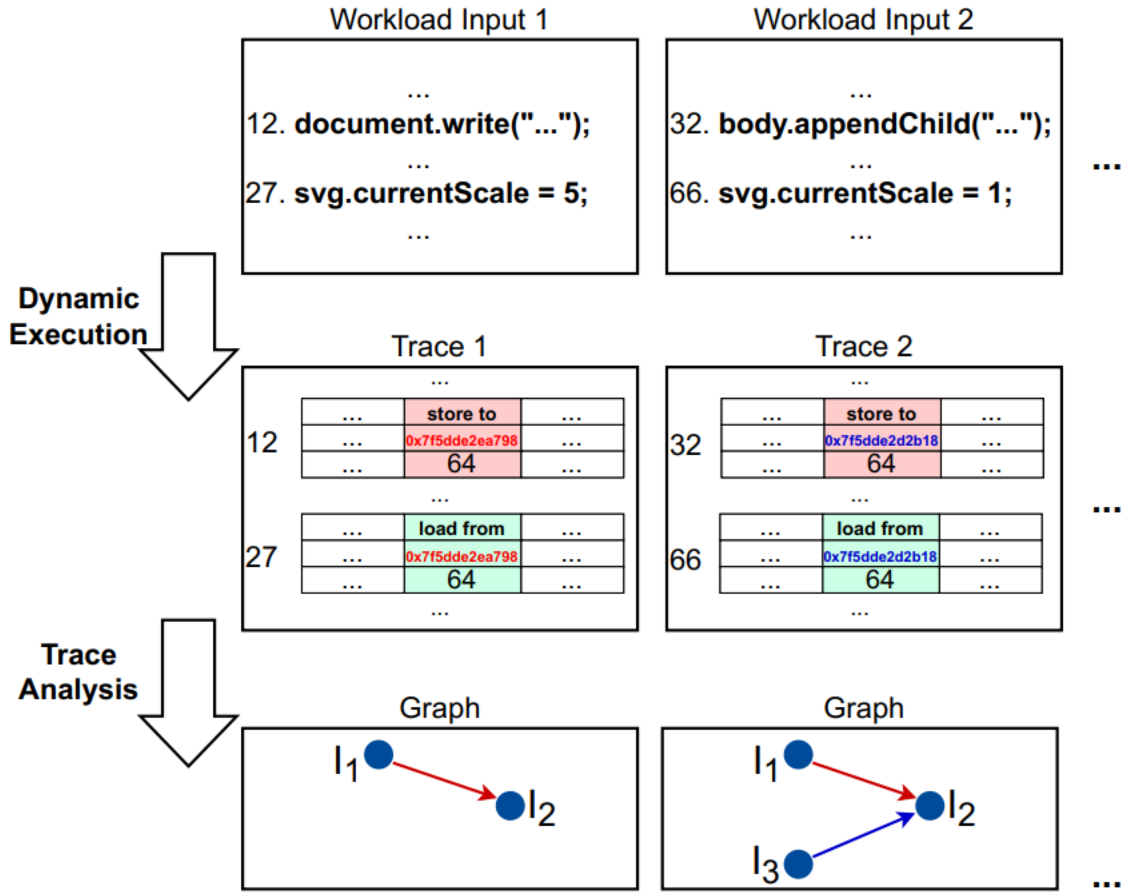


Minerva: A New Solution for Browser Fuzzing

- Insight: Fuzzing guided by memory mod-ref relations of APIs
- A two-stage design:
    1. Analyze the mod-ref relations through dynamic traces during preparation
    2. **Select highly relevant APIs based on the relations during fuzzing**



Design (1/2): Dynamic Mod-Ref Analysis



```
...
12. document.write("...");
...
27. svg.currentScale = 5;
...
```

```
...
32. body.appendChild("...");
...
66. svg.currentScale = 1;
...
```

Trace 1

12	...	store to	...
	...	0x7f5dde2ea798	...
	...	64	...
27	...	load from	...
	...	0x7f5dde2ea798	...
	...	64	...

Trace 2

32	...	store to	...
	...	0x7f5dde2d2b18	...
	...	64	...
66	...	load from	...
	...	0x7f5dde2d2b18	...
	...	64	...

l_1: <Document>.write(<DOMString>)
l_2: <SVGSVGElement>.currentScale = <float>
l_3: <new Node> = <Node>.appendChild(<Node>)

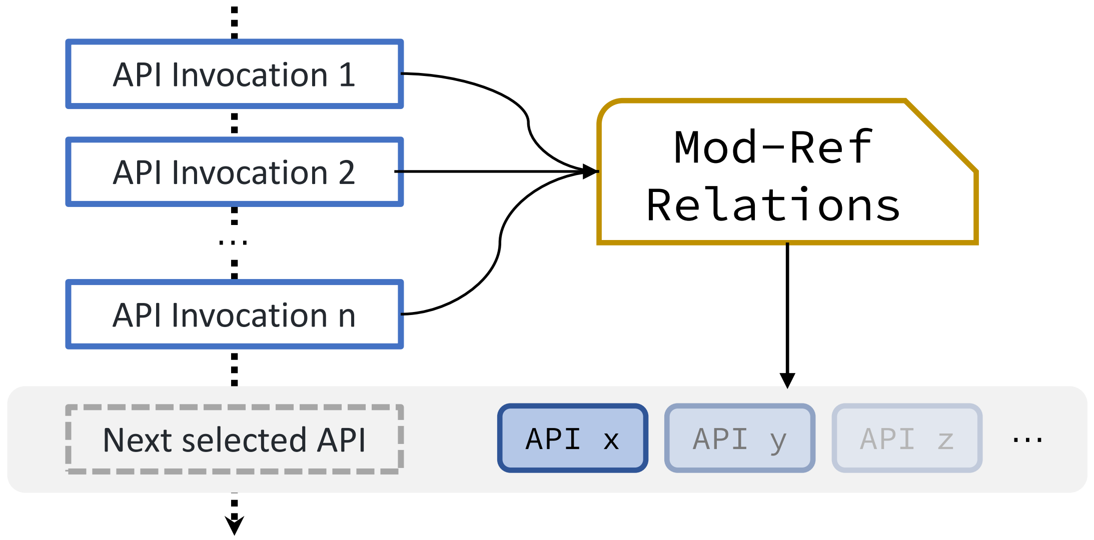
Sound

- Every relation has a concrete proof
- Unlikely to over-approximate relations due to dynamic analysis

Efficient

- Only focus on memory locations commonly visited by multiple APIs
- Analyze dataflow based on Andersen's pointer analysis

Design (2/2): Guided Input Generation



Relation-Guided

- Weighted selection for the next generated API invocation

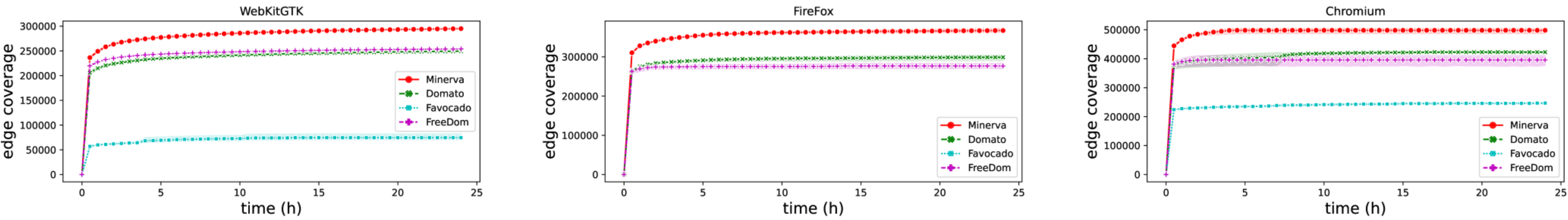
Context-Aware

- Maintain context of DOM objects (reusing Domato code)

Semantics-Aware

- API declarations are extracted from code base of browsers
- Generate inputs following type-correctness

Evaluation (1/3): Comparison to Existing Fuzzers



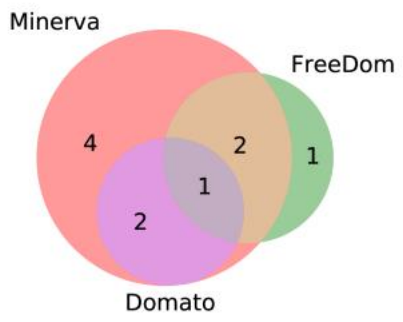
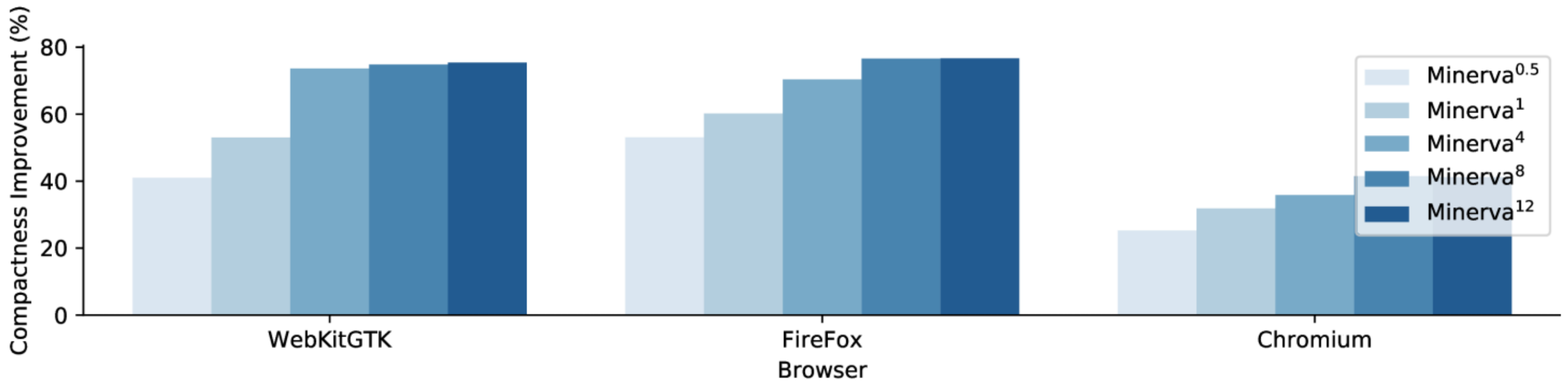


Figure 7: The overlapping relations of the unique bugs found by each fuzzer on WebKitGTK in 24 hours.

- ✓ 19.63% ~ 229.62% more coverage
- ✓ 39.18% ~ 68.67% more compactness
- ✓ More unique bugs

Evaluation (2/3): Effectiveness of Redundancy Reduction

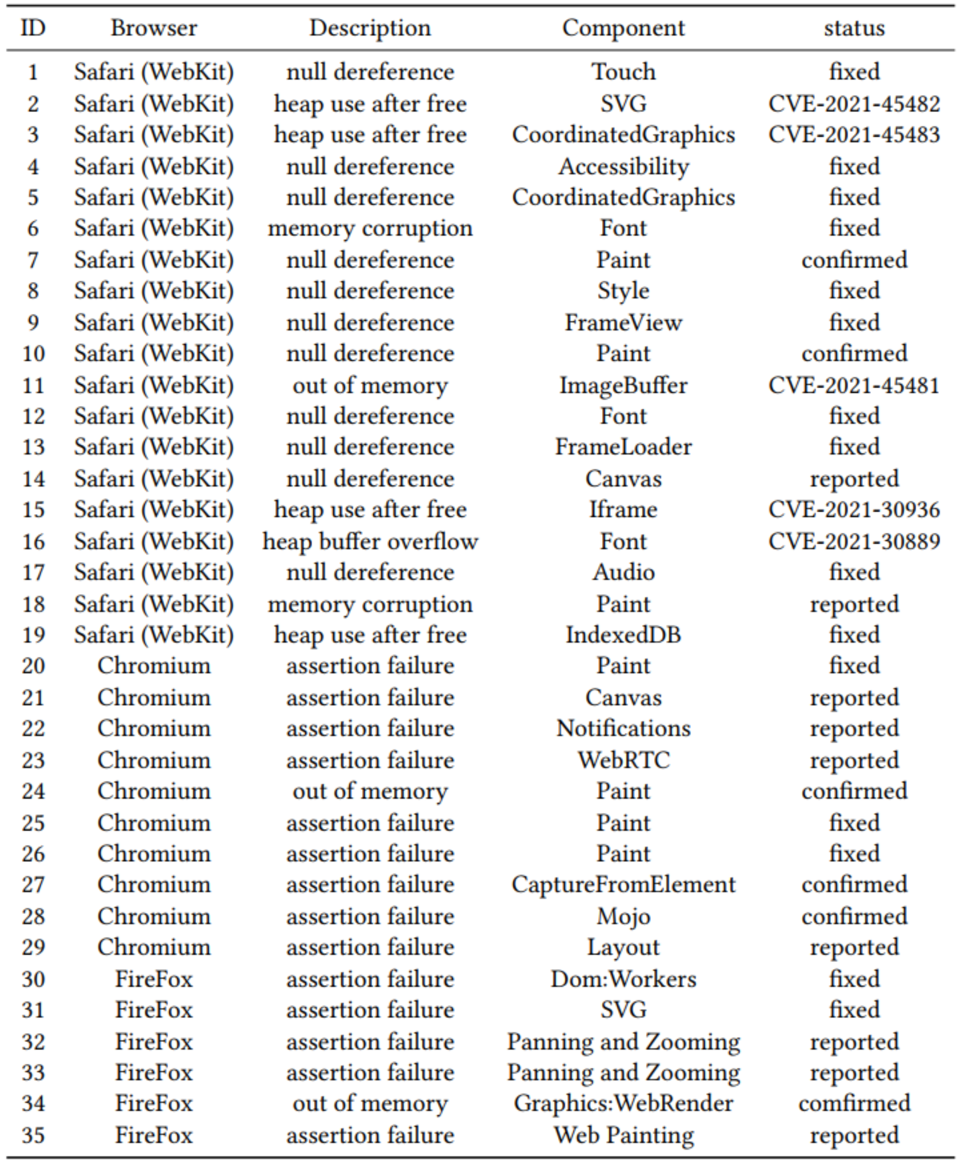


✓ API mod-ref relations are helpful to reduce redundancy

Evaluation (3/3): Discovering Unknown Browser Bug

35 new bugs:

- 26 have been confirmed;
- 20 have been fixed;
- 5 new CVEs.

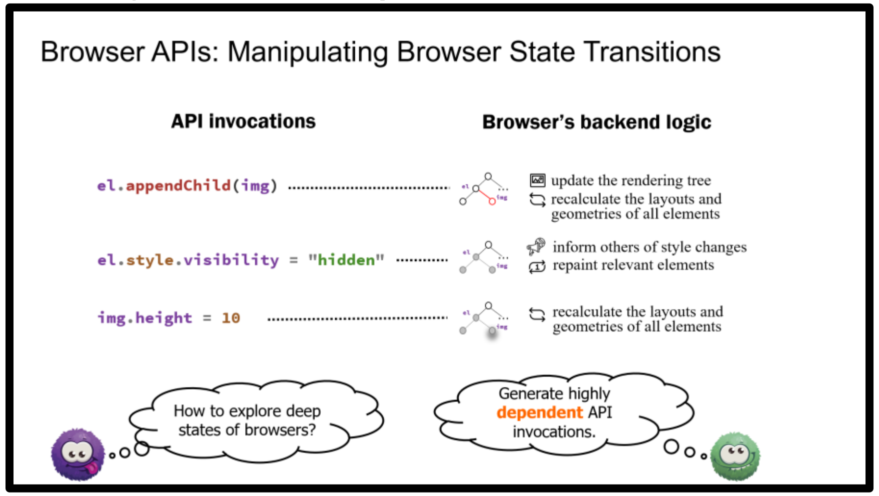


ID	Browser	Description	Component	status
1	Safari (WebKit)	null dereference	Touch	fixed
2	Safari (WebKit)	heap use after free	SVG	CVE-2021-45482
3	Safari (WebKit)	heap use after free	CoordinatedGraphics	CVE-2021-45483
4	Safari (WebKit)	null dereference	Accessibility	fixed
5	Safari (WebKit)	null dereference	CoordinatedGraphics	fixed
6	Safari (WebKit)	memory corruption	Font	fixed
7	Safari (WebKit)	null dereference	Paint	confirmed
8	Safari (WebKit)	null dereference	Style	fixed
9	Safari (WebKit)	null dereference	FrameView	fixed
10	Safari (WebKit)	null dereference	Paint	confirmed
11	Safari (WebKit)	out of memory	ImageBuffer	CVE-2021-45481
12	Safari (WebKit)	null dereference	Font	fixed
13	Safari (WebKit)	null dereference	FrameLoader	fixed
14	Safari (WebKit)	null dereference	Canvas	reported
15	Safari (WebKit)	heap use after free	Iframe	CVE-2021-30936
16	Safari (WebKit)	heap buffer overflow	Font	CVE-2021-30889
17	Safari (WebKit)	null dereference	Audio	fixed
18	Safari (WebKit)	memory corruption	Paint	reported
19	Safari (WebKit)	heap use after free	IndexedDB	fixed
20	Chromium	assertion failure	Paint	fixed
21	Chromium	assertion failure	Canvas	reported
22	Chromium	assertion failure	Notifications	reported
23	Chromium	assertion failure	WebRTC	reported
24	Chromium	out of memory	Paint	confirmed
25	Chromium	assertion failure	Paint	fixed
26	Chromium	assertion failure	Paint	fixed
27	Chromium	assertion failure	CaptureFromElement	confirmed
28	Chromium	assertion failure	Mojo	confirmed
29	Chromium	assertion failure	Layout	reported
30	FireFox	assertion failure	Dom:Workers	fixed
31	FireFox	assertion failure	SVG	fixed
32	FireFox	assertion failure	Panning and Zooming	reported
33	FireFox	assertion failure	Panning and Zooming	reported
34	FireFox	out of memory	Graphics:WebRender	comfirmed
35	FireFox	assertion failure	Web Painting	reported

Summary

Minerva: https://github.com/ChijinZ/Minerva

Goal: generate dependent API invocations

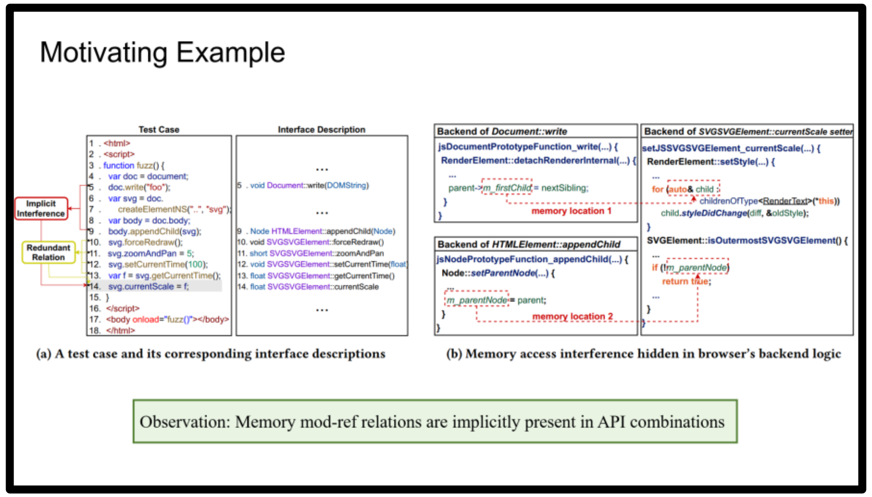


Browser APIs: Manipulating Browser State Transitions

API invocations	**Browser's backend logic**
`el.appendChild(img)`	update the rendering tree; recalculate the layouts and geometries of all elements
`el.style.visibility = "hidden"`	inform others of style changes; repaint relevant elements
`img.height = 10`	recalculate the layouts and geometries of all elements

How to explore deep states of browsers?
Generate highly dependent API invocations.

Motivation: implicit relations are overlooked



Motivating Example

Test Case

```
1 . <html>
2 . <script>
3 . function fuzz() {
4 .   var doc = document;
5 .   doc.write("foo");
6 .   var svg = doc.
7 .     createElementNS("..", "svg");
8 .   var body = doc.body;
9 .   body.appendChild(svg);
10.   svg.forceRedraw();
11.   svg.zoomAndPan = 5;
12.   svg.setCurrentTime(100);
13.   var f = svg.getCurrentTime();
14.   svg.currentScale = f;
15. }
16. </script>
17. <body onload="fuzz()"></body>
18. </html>
```

Interface Description

```
...
5 . void Document::write(DOMString)
...
9 . Node HTMLElement::appendChild(Node)
10. void SVGSVGElement::forceRedraw()
11. short SVGSVGElement::zoomAndPan
12. void SVGSVGElement::setCurrentTime(float)
13. float SVGSVGElement::getCurrentTime()
14. float SVGSVGElement::currentScale
...
```

(a) A test case and its corresponding interface descriptions

Backend of *Document::write*

```
jsDocumentPrototypeFunction_write(...) {
 RenderElement::detachRendererInternal(...) {
   ...
   parent->m_firstChild = nextSibling;
 }
}
```

Backend of *HTMLElement::appendChild*

```
jsNodePrototypeFunction_appendChild(...) {
 Node::setParentNode(...) {
   ...
   m_parentNode = parent;
 }
}
```

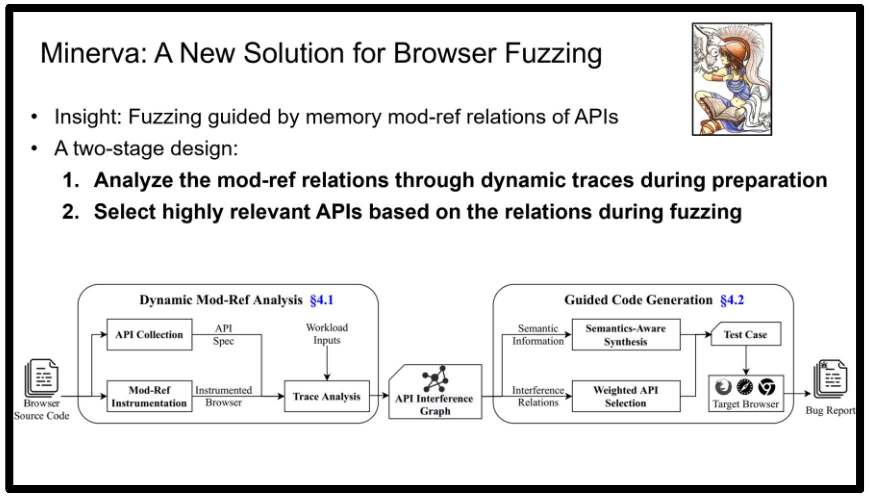
Backend of *SVGSVGElement::currentScale setter*

```
setJSSVGSVGElement_currentScale(...) {
 RenderElement::setStyle(...) {
   ...
   for (auto& child :
       childrenOfType<RenderText>(*this))
     child.styleDidChange(diff, &oldStyle);
 }
 SVGElement::isOutermostSVGSVGElement() {
   ...
   if (!m_parentNode)
     return true;
   ...
 }
}
```

(b) Memory access interference hidden in browser's backend logic

Observation: Memory mod-ref relations are implicitly present in API combinations

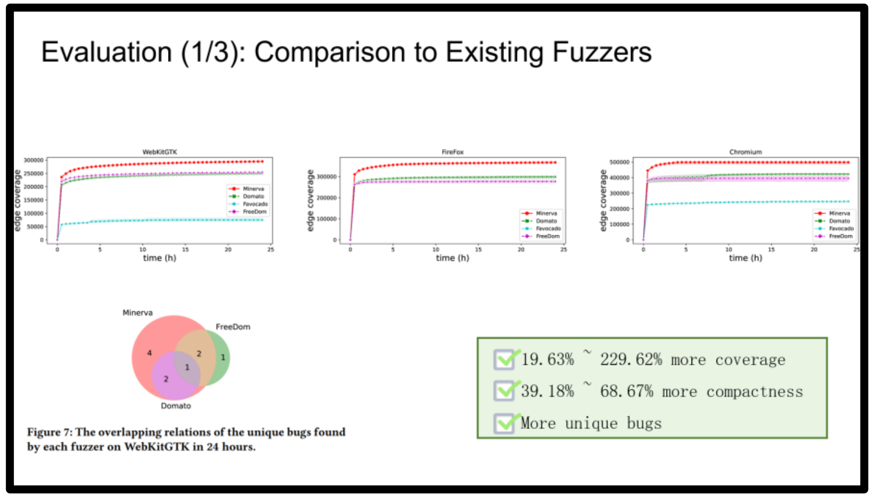
Method: fuzzing guided by mod-ref relations



Minerva: A New Solution for Browser Fuzzing

- Insight: Fuzzing guided by memory mod-ref relations of APIs
- A two-stage design:
  1. **Analyze the mod-ref relations through dynamic traces during preparation**
  2. **Select highly relevant APIs based on the relations during fuzzing**

Evaluation: more coverage, more bugs



Evaluation (1/3): Comparison to Existing Fuzzers

Figure 7: The overlapping relations of the unique bugs found by each fuzzer on WebKitGTK in 24 hours.

- ✓ 19.63% ~ 229.62% more coverage
- ✓ 39.18% ~ 68.67% more compactness
- ✓ More unique bugs