GLeeFuzz: Fuzzing WebGL Through Error Message Guided Mutation

Hui Peng (Purdue University, Google)

Zhihao “Zephyr” Yao (UC Irvine, NJIT)

Ardalan Amiri Sani (UC Irvine)

Dave (Jing) Tian (Purdue University)

Mathias Payer (EPFL)
WebGL enables 3D graphics for web apps

WebGL was released in 2011

WebGL is increasingly popular

The top 100 most visited websites are almost all using WebGL
WebGL enables 3D graphics for web apps

WebGL was released in 2011

WebGL is increasingly popular

The top 100 most visited websites are almost all using WebGL

https://www.apple.com/macos/sierra/
WebGL enables 3D graphics for web apps

WebGL was released in 2011

WebGL is increasingly popular

The top 100 most visited websites are almost all using WebGL
How does WebGL work?
Traditionally, native apps are trusted.
Traditionally, native apps are trusted.
WebGL contents are not trusted

Browser

Kernel mode GPU driver

GPU hardware

user space

kernel space

Web app

GPU Process

Checks

GL libs

Web app

Web app
WebGL contents are not trusted
We want to analyze WebGL security through fuzzing

- WebGL exposes low level graphic code to attackers
- Fuzzing is a battle-tested technique to find vulnerabilities
Strawman solution: fuzzing WebGL with code coverage

- web app
- Browser components
- GL libs
- Kernel with GPU driver
- GPU hardware

user space
kernel space
hardware
Strawman solution: fuzzing WebGL with code coverage
Challenges faced by coverage-guided fuzzing on WebGL

Challenge 1: Collect coverage across processes
Challenge 2: Collect coverage from close-sourced binaries
Challenge 3: Collect coverage across user/kernel mode
Challenge 1: Collect coverage across processes

Random WebGL inputs -> Inter-Process Call -> web app

Coverage

Input Mutator

Kernel with GPU driver

GL libs

Browser components

web app

user space

kernel space

hardware
Challenge 2: Collect coverage from close-sourced binaries

Browser components

GL libs

web app

Kernel with GPU driver

GPU hardware

Random WebGL inputs

Inter-Process Call

Coverage

Input Mutator

Some browsers are close-sourced

GL libs are often close-sourced

user space

kernel space

Hardware
Challenge 3: Collect coverage across user/kernel mode

- Kernel with GPU driver
  - GPU hardware
  - kernel space
  - kernel space

- Inter-Process Call
  - syscall

- Browser components
  - GL libs
  - user space
  - user space

- web app

- Input Mutator

- Random WebGL inputs

- Coverage

- ?
Observation: WebGL has excellent error feedbacks

Browser components

GL libs

Kernel with GPU driver

GPU hardware

“lower your vertex index”
GLeeFuzz Workflow

Type 1 error messages

Type 2 error messages

LLVM bitcode of WebGL in Chrome

WebGL Spec (IDL)

Error Messages

input
corpus

Pre-Processing

Fuzzing

browsers
canvas = document.createElement("canvas");
gl = canvas.getContext("webgl");
shader = gl.createShader(gl.VERTEX_SHADER);
buffer = gl.createBuffer();
// ....
ogl.bufferData(gl.ALPHA, 100, gl.STATIC_DRAW);
program = gl.createProgram();

"invalid target"
Type 2 error message: indicating invalidity of internal state

```javascript
canvas = document.createElement("canvas");
gl = canvas.getContext("webgl");
shader = gl.createShader(gl.VERTEX_SHADER);
buffer = gl.createBuffer();
// ....
// ....
// ....
gl.useProgram(program);
gl.drawArrays(gl.POINTS, 100, gl.STATIC_DRAW);
```

"no valid shader program in use"
Build mutating rules based on error messages

Type-1 messages: find the arguments that cause the error

Type-2 messages: find the dependent APIs
Type-1 Message: Computing Target Arguments

Key idea

Error-emitting statement are tainted by certain internal variable, leading to the culprit API argument

Approach

Backward taint analysis on the internal variable of the error-emitting statement
Example

```cpp
void bufferData(GLenum target, int64_t size, GLenum usage) {
    BufferDataImpl(target, size, nullptr, usage);
}

void BufferDataImpl(GLenum target, int64_t size, const void* data, GLenum usage) {
    ValidateBufferDataTarget("bufferData", target);
    // ...
}

WebGLBuffer* ValidateBufferDataTarget(const char* function_name, GLenum target) {
    // ..... switch (target) {
    case GL_ARRAY_BUFFER:
        buffer = bound_array_buffer_.Get();
        break;
    default:
        SynthesizeGLError("invalid target");
    }
    // ...
}
```

```cpp
shader = gl.createShader(gl.VERTEX_SHADER);
buffer = gl.createBuffer();
// ....
gl.bufferData(gl.ALPHA, 100, gl.STATIC_DRAW);
```
Type-2 Message: Computing Dependent API Set

Key idea

Conditions of error-emitting statements are tainted by internal variables which are updated by other APIs (i.e., a dependent API set)
Additional benefit of error message guided fuzzing

Containing useful information about which part of the input is invalid
Evaluation: GLeeFuzz outperforms random mutation

# of unique WebGL API triggered
Evaluation: GLeeFuzz outperforms random mutation

# of unique WebGL API triggered

# of unique WebGL error messages triggered
So far, 7 new vulnerabilities in WebGL have been found

<table>
<thead>
<tr>
<th>Bug Descriptions</th>
<th>GPU</th>
<th>Platform</th>
<th>Browser</th>
<th>Bug Location</th>
<th>Severity</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU hang</td>
<td>Apple GPU</td>
<td>iOS</td>
<td>Safari</td>
<td>GPU Driver</td>
<td>Not set</td>
</tr>
<tr>
<td>GPU hang; X-Server freeze</td>
<td>Intel</td>
<td>Ubuntu</td>
<td>Chrome</td>
<td>GPU Driver</td>
<td>Medium</td>
</tr>
<tr>
<td>Nullptr dereference in GPU process</td>
<td>N/A</td>
<td>N/A</td>
<td>Chrome</td>
<td>Browser</td>
<td>Not set</td>
</tr>
<tr>
<td>Memory corruption in GPU process</td>
<td>N/A</td>
<td>N/A</td>
<td>Chrome</td>
<td>Browser</td>
<td>High</td>
</tr>
<tr>
<td>Assertion failure</td>
<td>N/A</td>
<td>N/A</td>
<td>Chrome</td>
<td>Browser</td>
<td>Low</td>
</tr>
<tr>
<td>OS memory leak</td>
<td>Intel</td>
<td>macOS; Ubuntu</td>
<td>Firefox</td>
<td>Browser</td>
<td>Low</td>
</tr>
<tr>
<td>Tab crash</td>
<td>N/A</td>
<td>macOS</td>
<td>Safari</td>
<td>Browser</td>
<td>Not set</td>
</tr>
</tbody>
</table>
Conclusion

● Fuzzing WebGL interface is challenging
● GLeeFuzz leverages error messages to fuzz WebGL
  ○ eliminates dependency on code coverage
  ○ performs meaningful mutation
  ○ has found 7 new vulnerabilities
  ○ Source: https://github.com/HexHive/GLeeFuzz

Thank you!