GLeeFuzz: Fuzzing WebGL Through Error Message Guided Mutation

Hui Peng (Purdue University, Google)

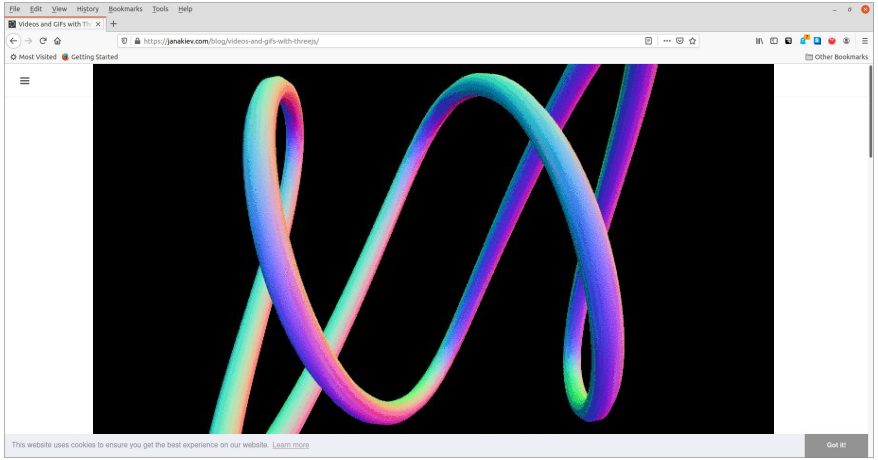
Zhihao “Zephyr” Yao (UC Irvine, NJIT)

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

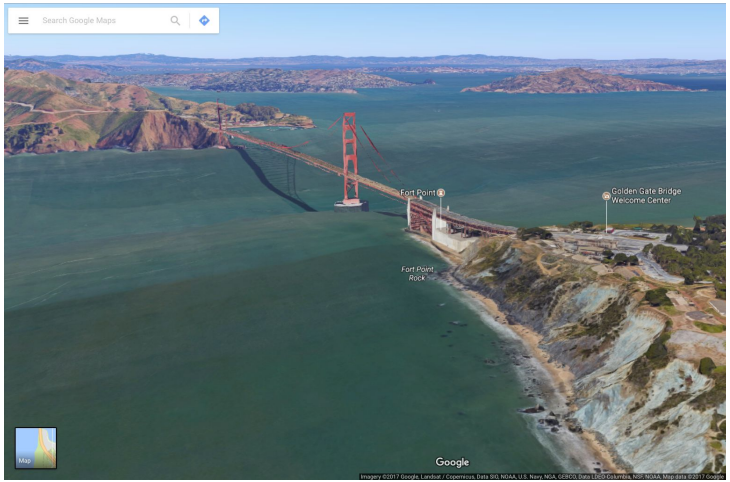
https://www.apple.com/macos/sierra/

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



https://www.google.com/map

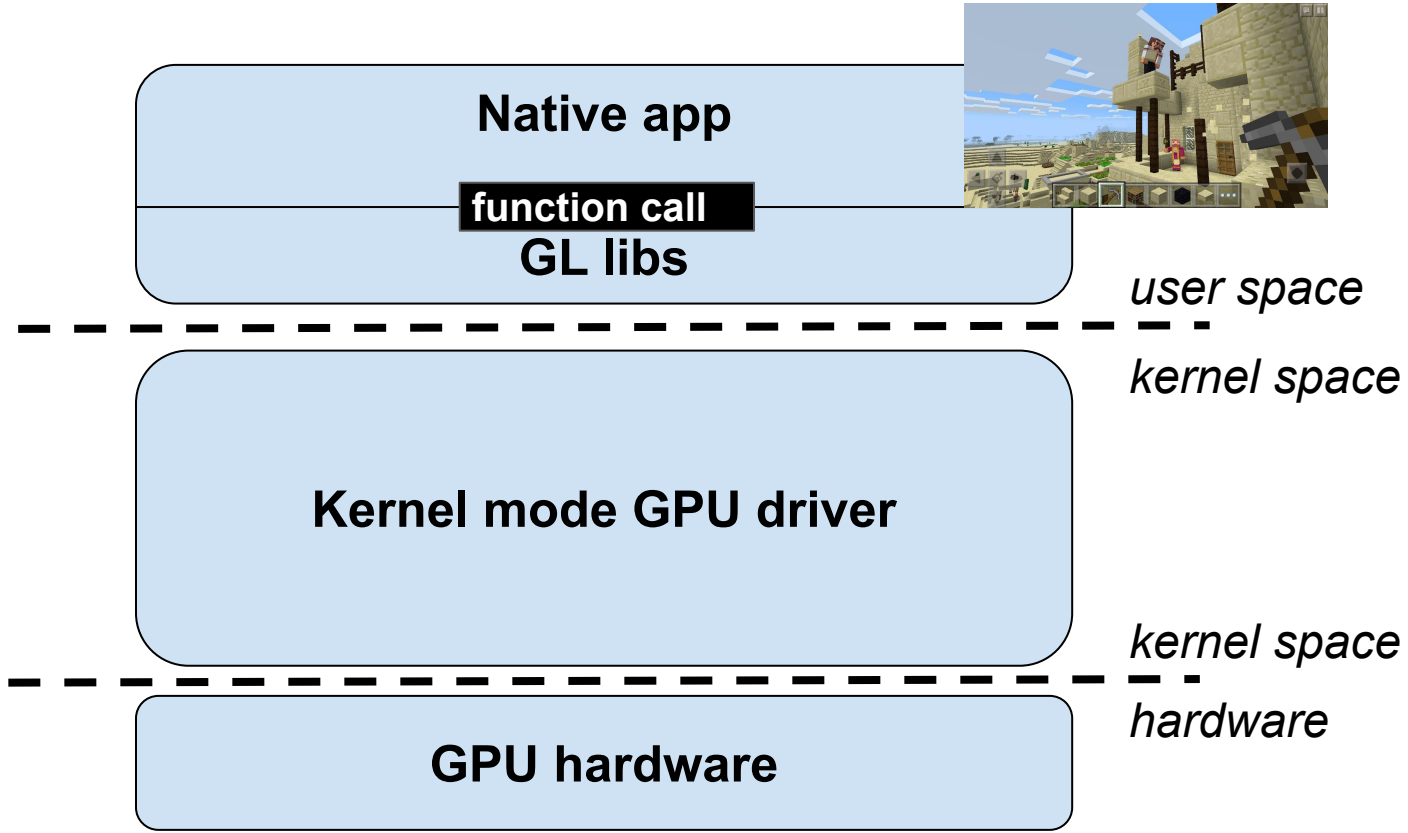
WebGL was released in 2011

WebGL is increasingly popular

The top 100 most visited websites are almost all using WebGL

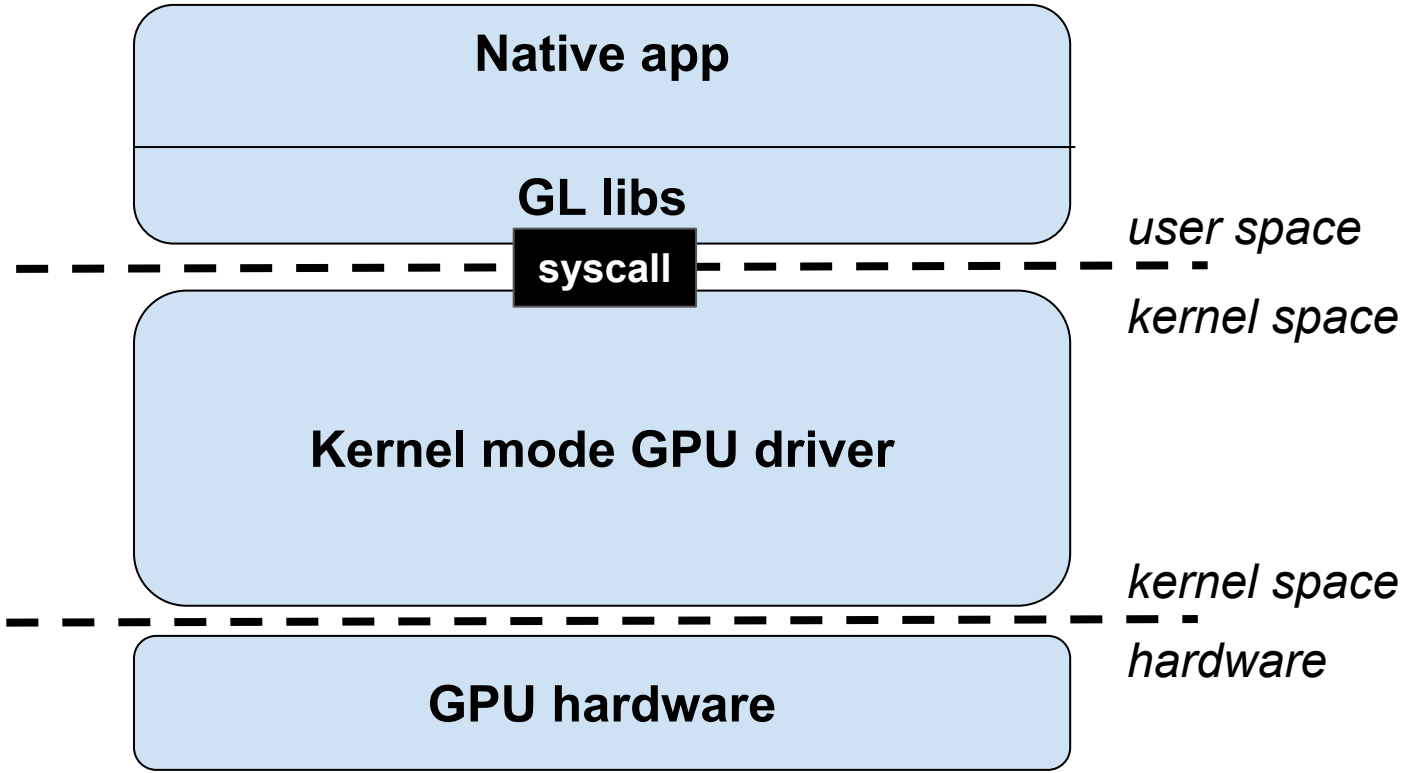
How does WebGL work?

Traditionally, native apps are trusted

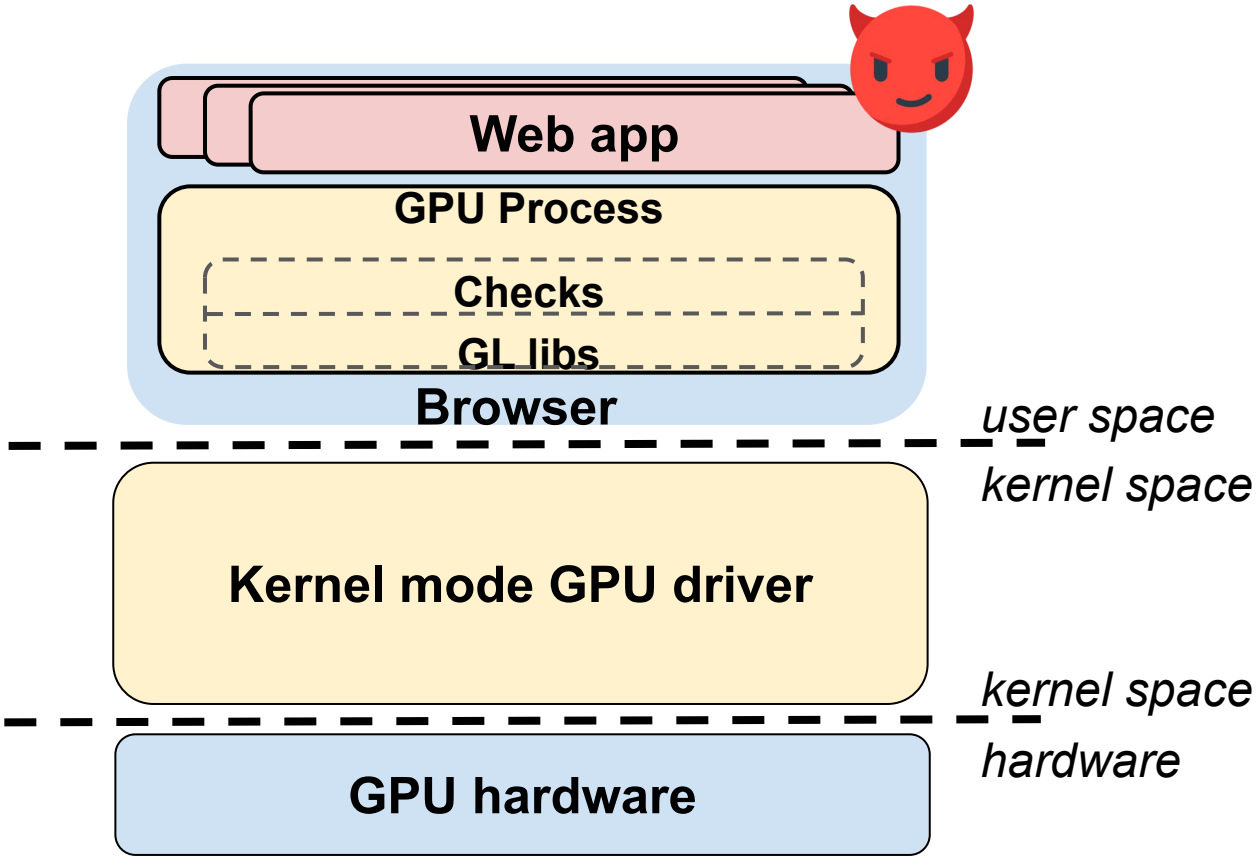


Minecraft on Google Play

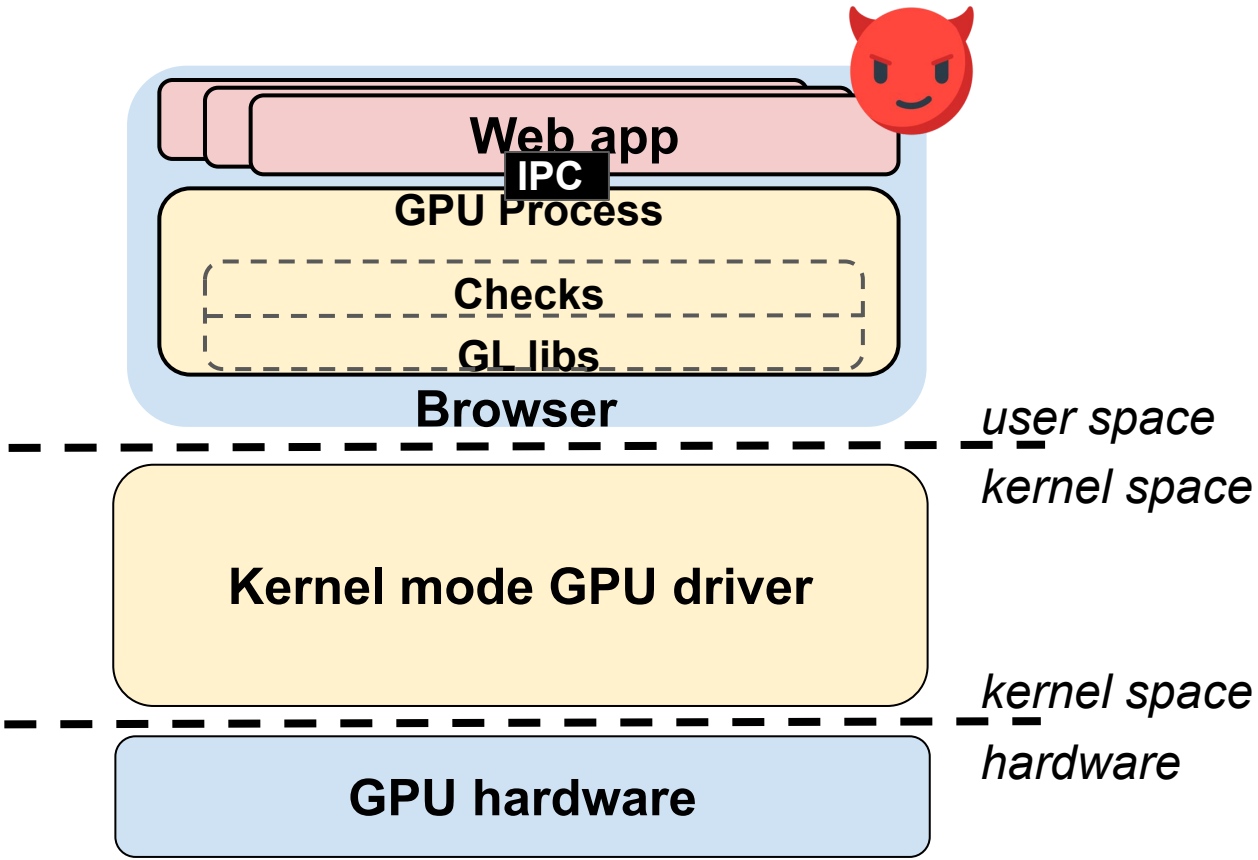
Traditionally, native apps are trusted



WebGL contents are not trusted



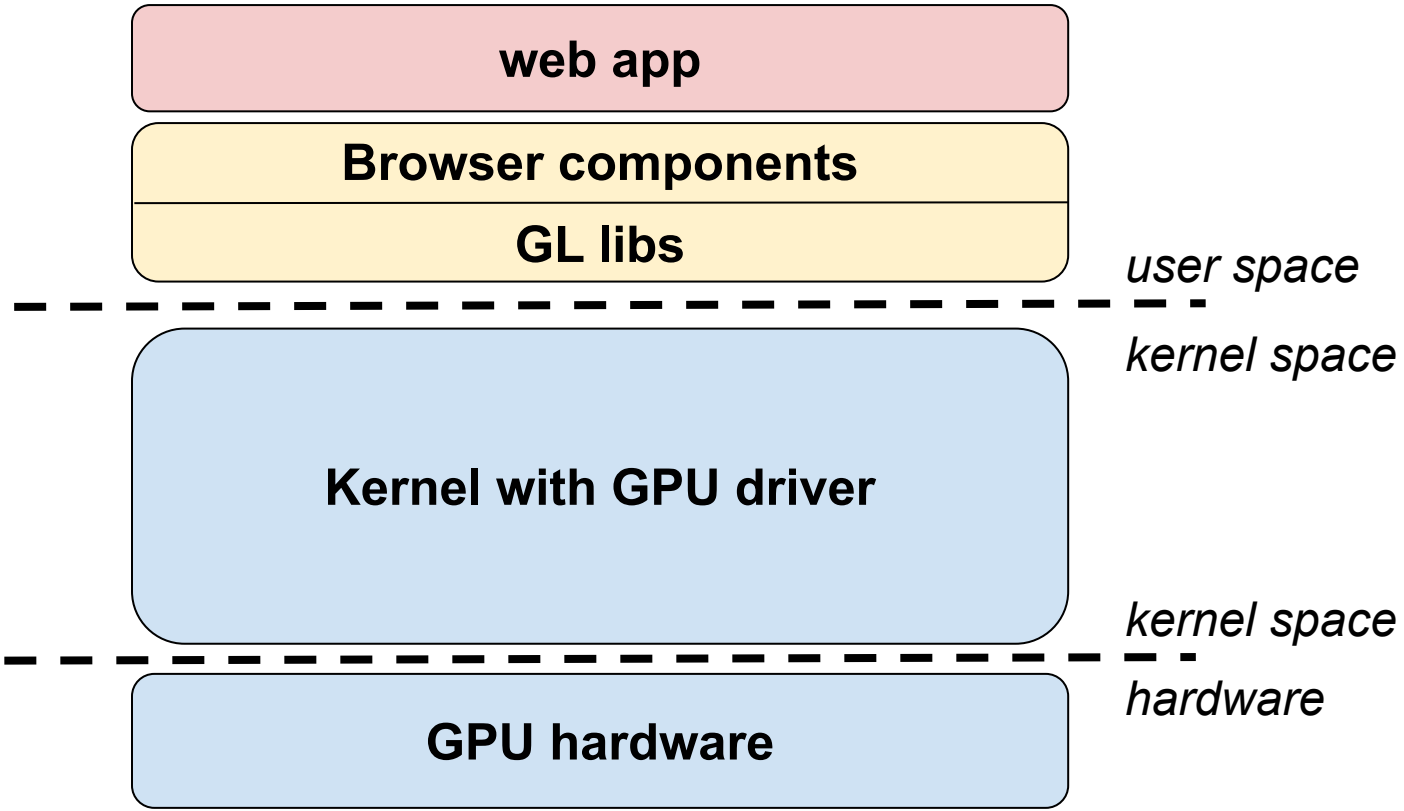
WebGL contents are not trusted



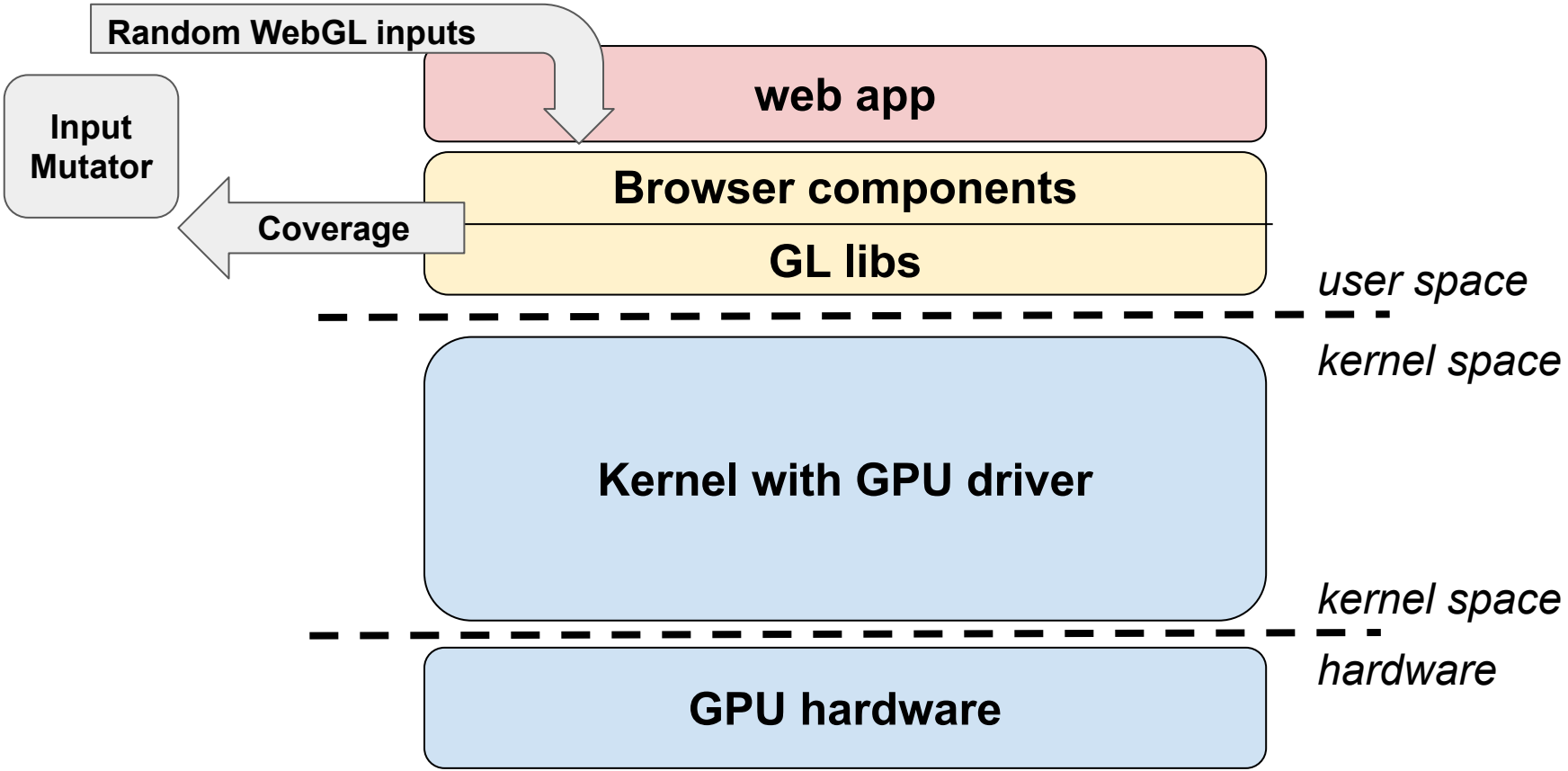
We want to analyze WebGL security through fuzzing

- WebGL exposes low level graphic code to attackers
- Fuzzing is a battle-tested technique to find vulnerabilities

Strawman solution: fuzzing WebGL with code coverage



Strawman solution: fuzzing WebGL with code coverage



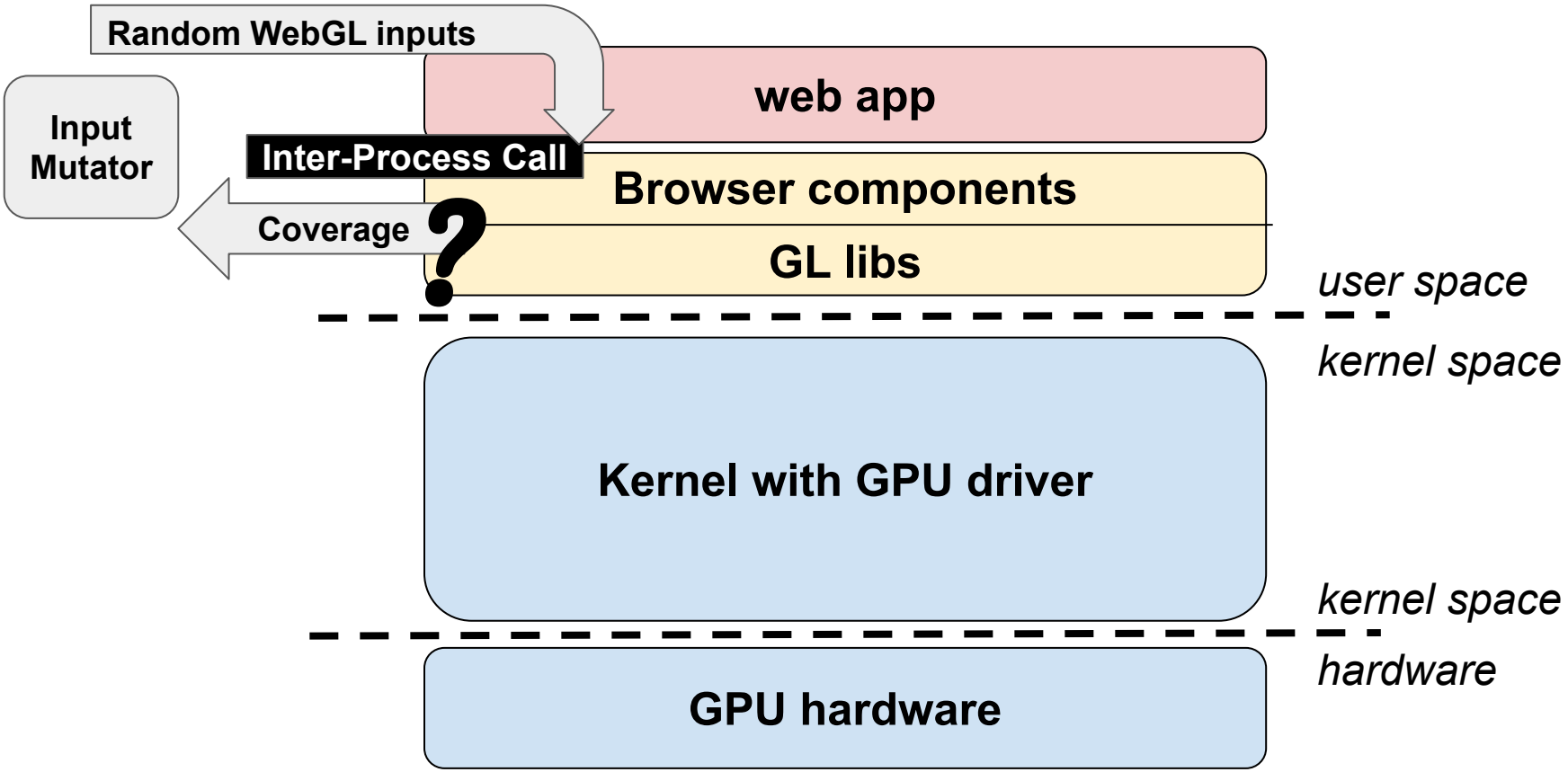
Challenges faced by coverage-guided fuzzing on WebGL

Challenge 1: Collect coverage across processes

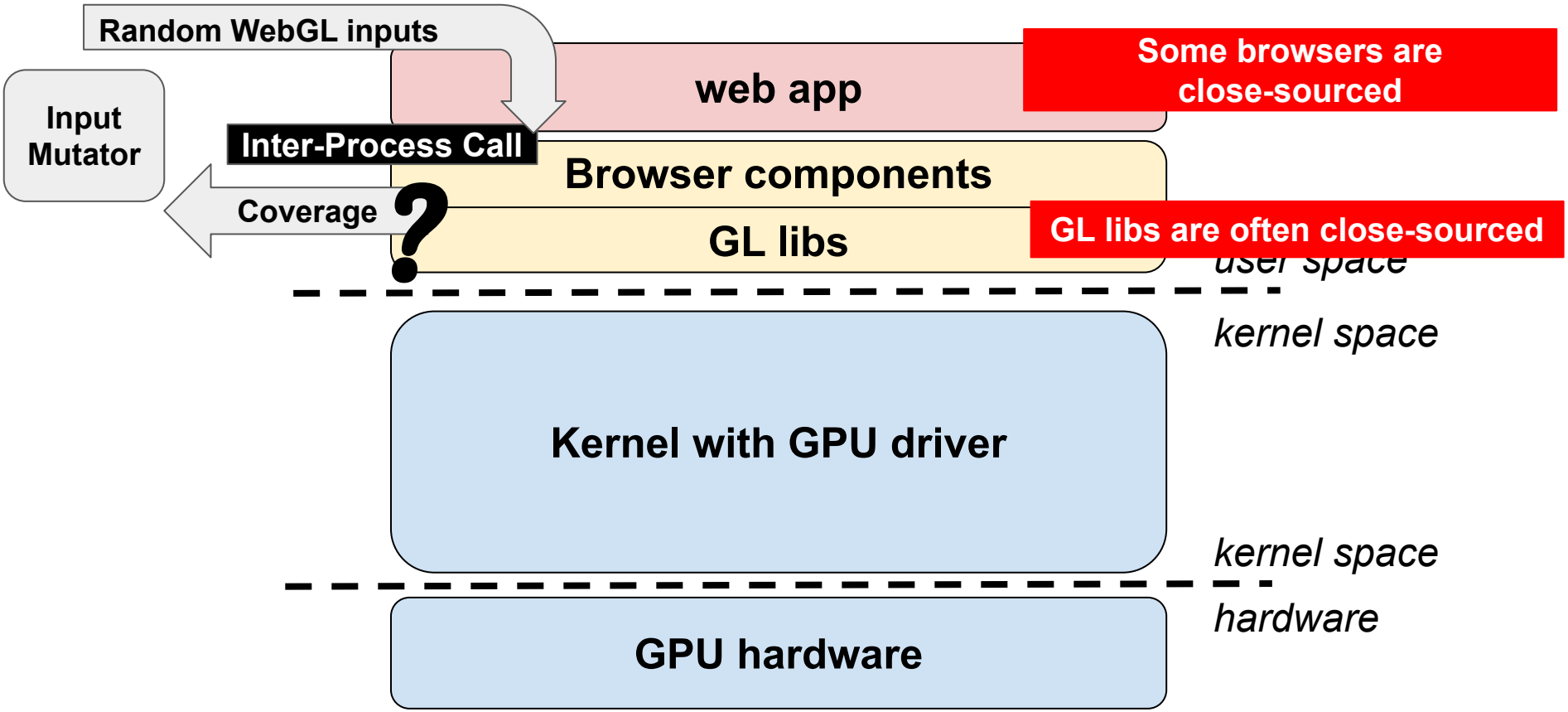
Challenge 2: Collect coverage from close-sourced binaries

Challenge 3: Collect coverage across user/kernel mode

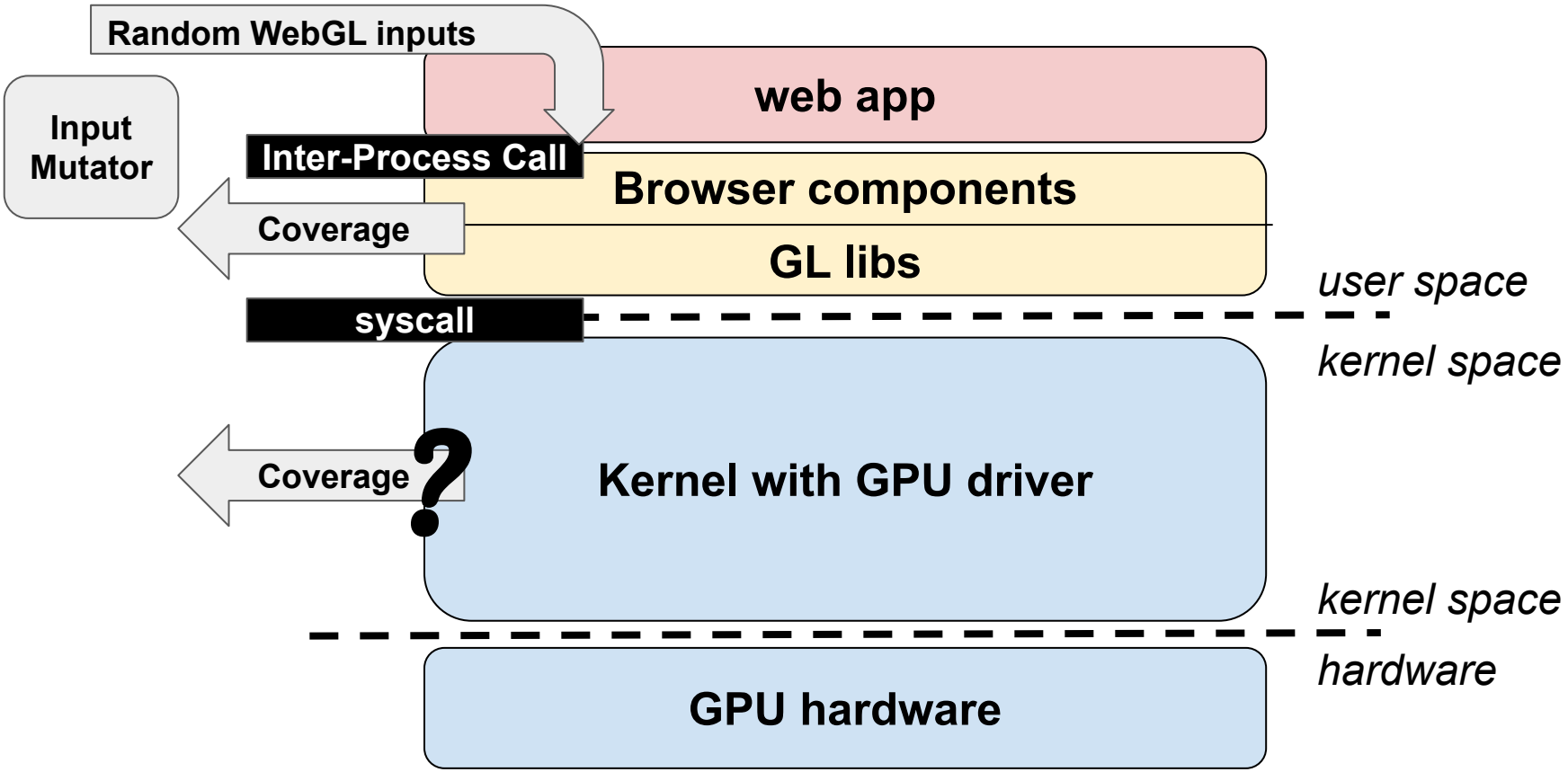
Challenge 1: Collect coverage across processes



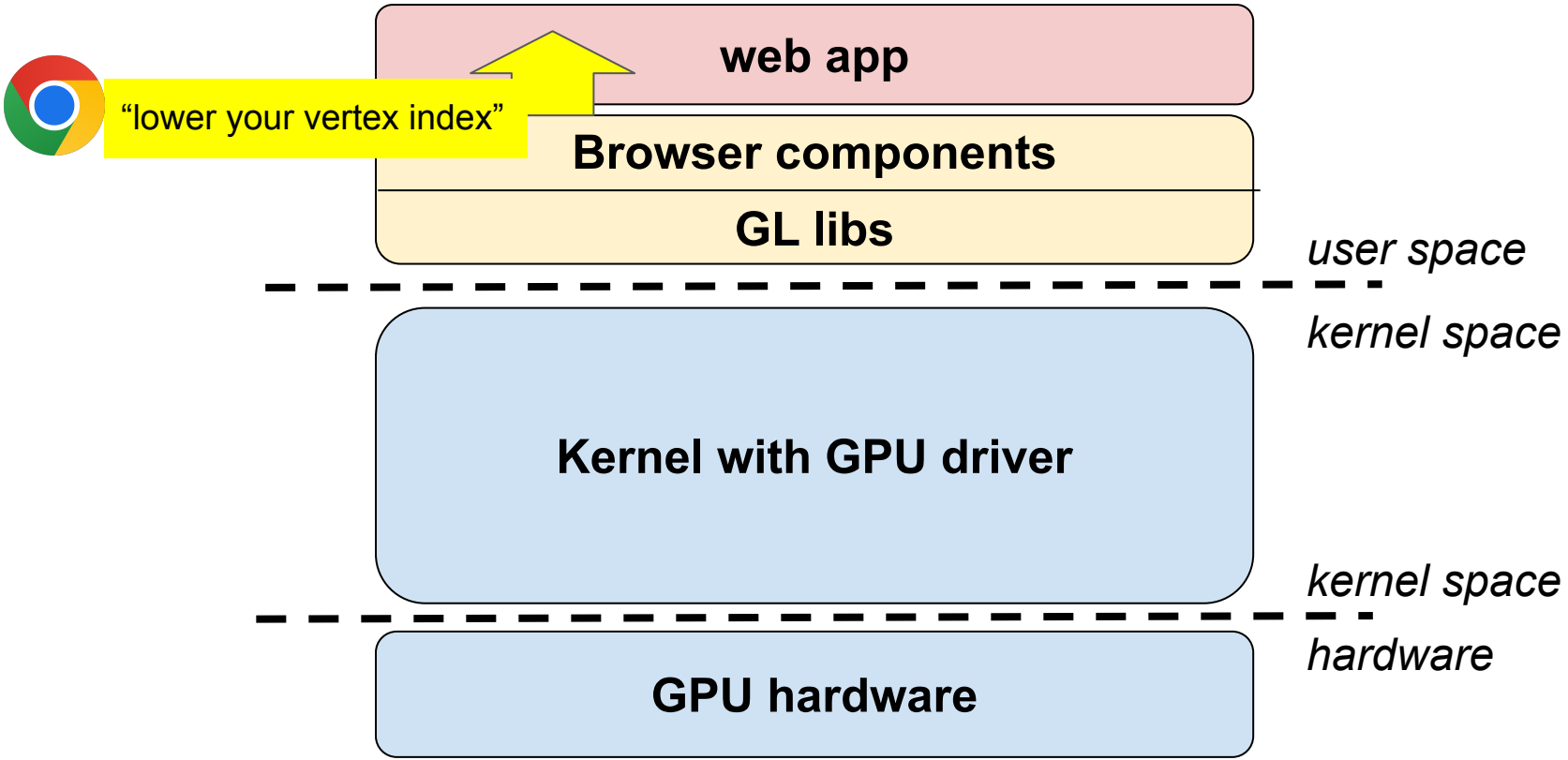
Challenge 2: Collect coverage from close-sourced binaries



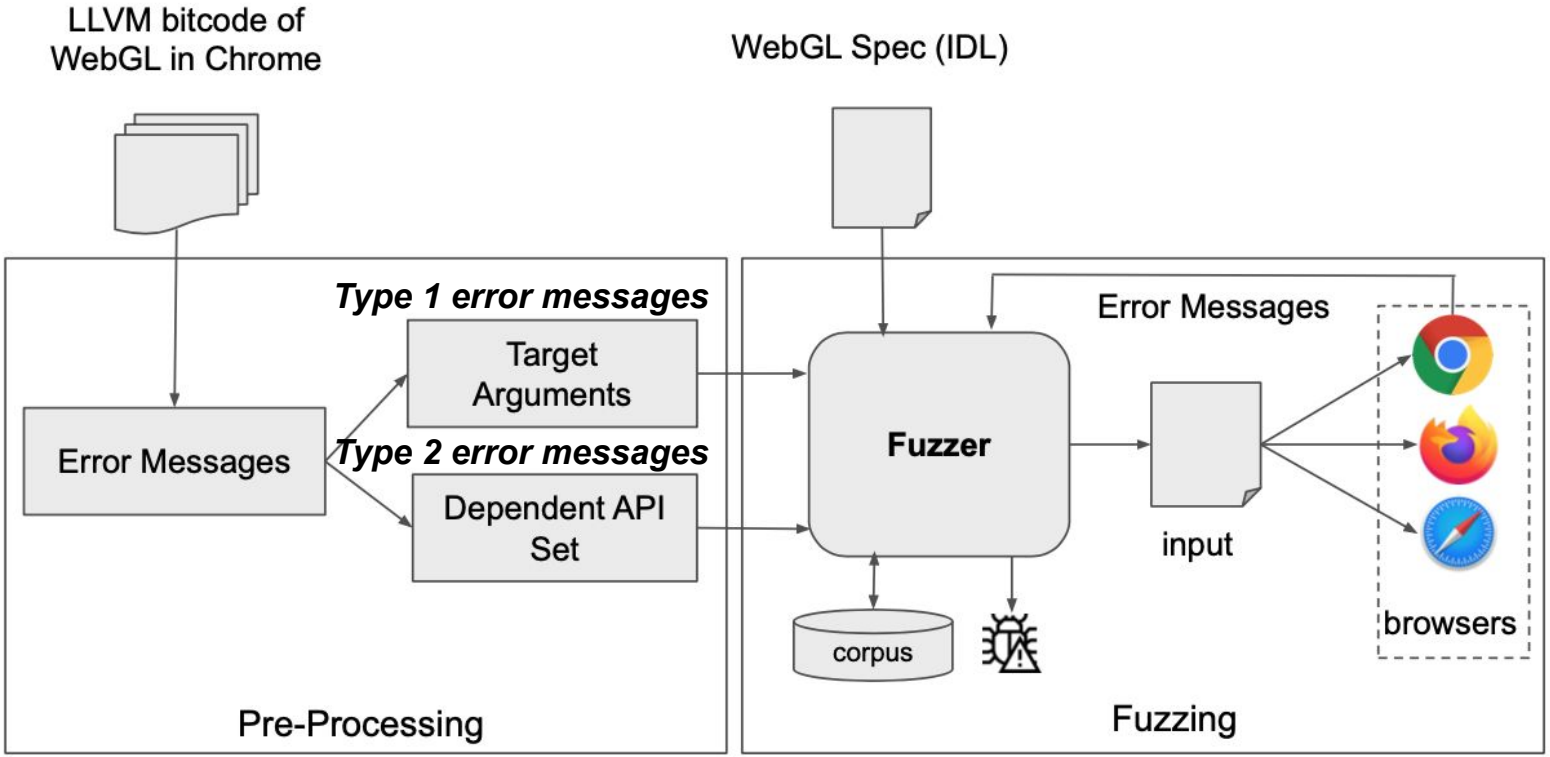
Challenge 3: Collect coverage across user/kernel mode



Observation: WebGL has excellent error feedbacks



GLeeFuzz Workflow



Type 1 error message: indicating invalidity of argument

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

Type 2 error message: indicating invalidity of internal state

```
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

```
shader = gl.createShader(gl.VERTEX_S
buffer = gl.createBuffer();
// ……
gl.bufferData(gl.ALPHA, 100, gl.STATIC_
```

```
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")
 }
}
```

Type-2 Message: Computing Dependent API Set

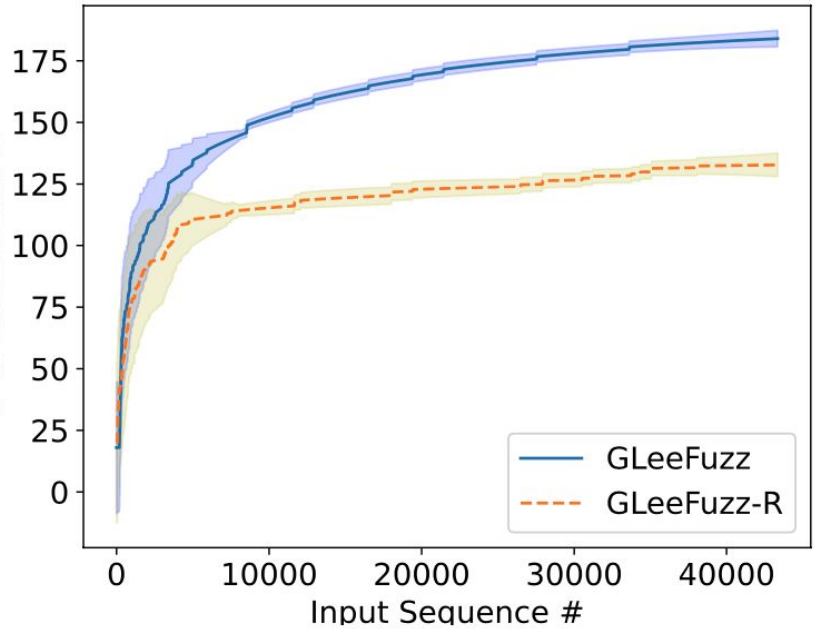
Key idea

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

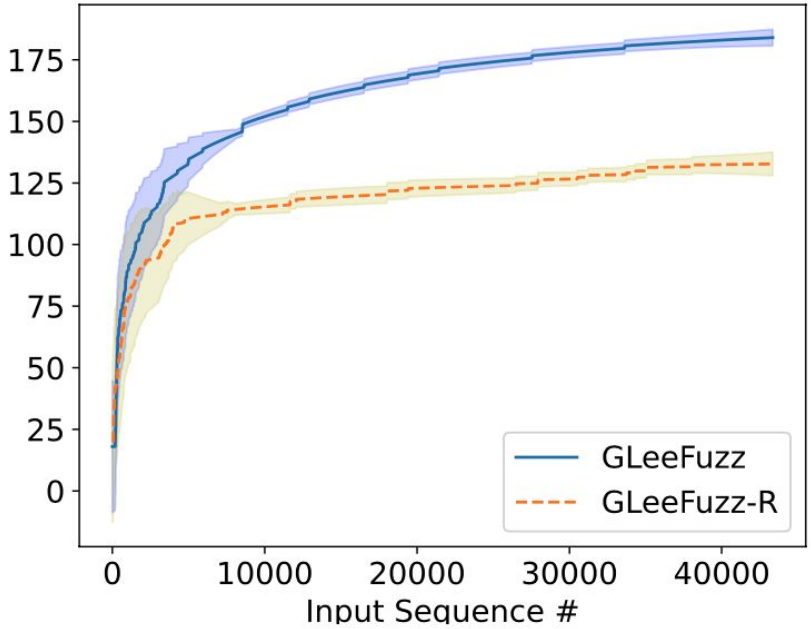
Containing useful information about which part of the input is invalid

Evaluation: GLeeFuzz outperforms random mutation

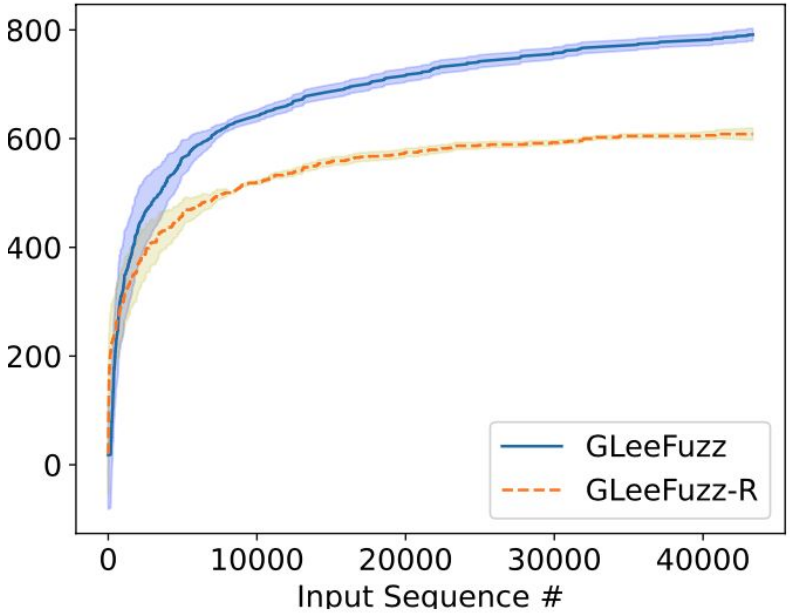


#of unique WebGL API triggered

Evaluation: GLeeFuzz outperforms random mutation



#of unique WebGL API triggered



#of unique WebGL error messages triggered

Evaluation

So far, 7 new vulnerabilities in WebGL have been found

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

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!

PURDUE UNIVERSITY UCI NJIT New Jersey Institute of Technology EPFL hexhive