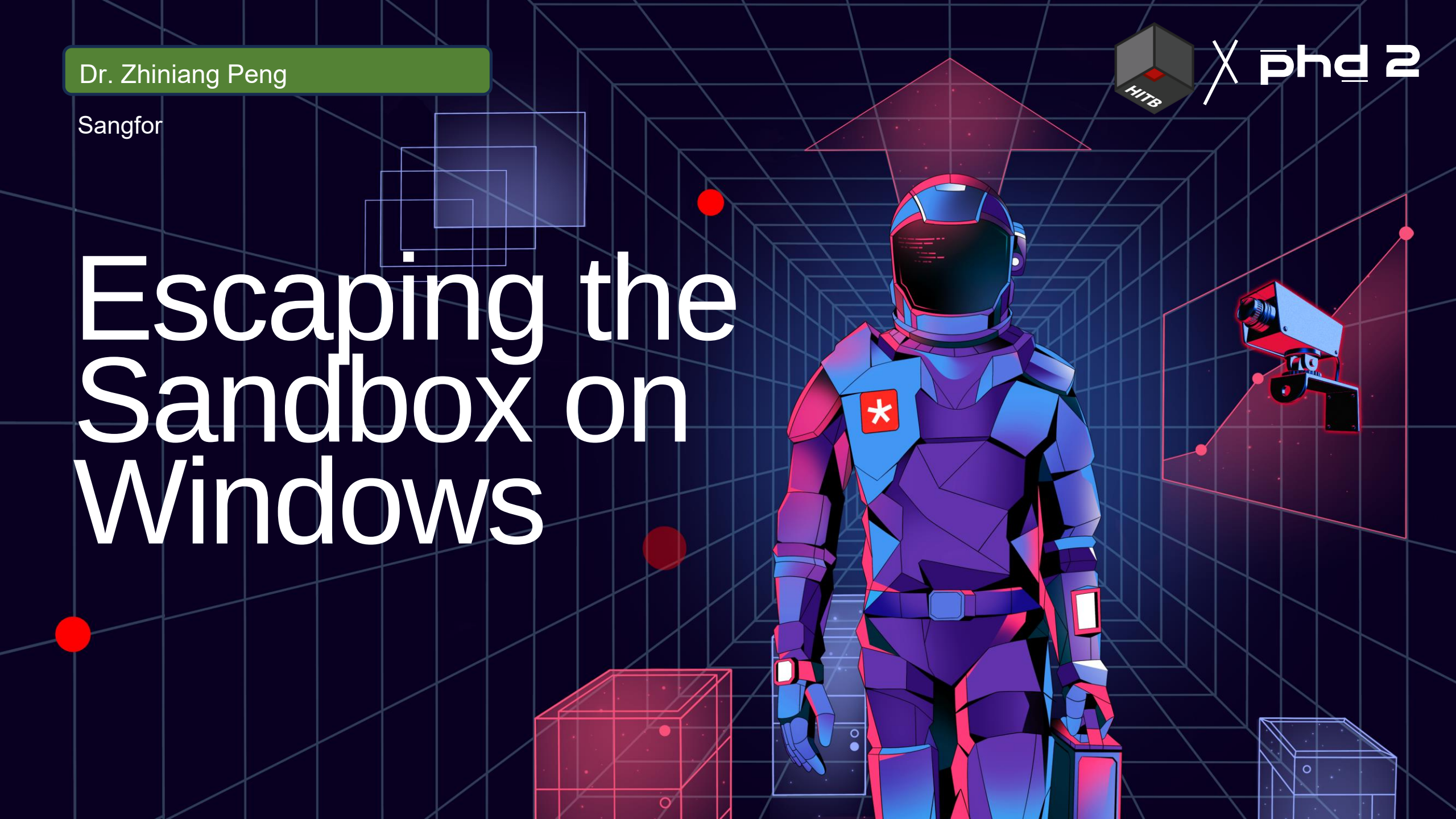
Dr. Zhiniang Peng

Sangfor

# Escaping the Sandbox on Windows



# Whoami

Zhiniang Peng @edwardzpeng

Principal Architect at Sangfor

PhD in Cryptography, interested in all areas of Computer Science

Work in Defensive & Offensive security

Published many research in both Industry & Academia

More about me: https://sites.google.com/site/zhiniangpeng

# some of my bugs

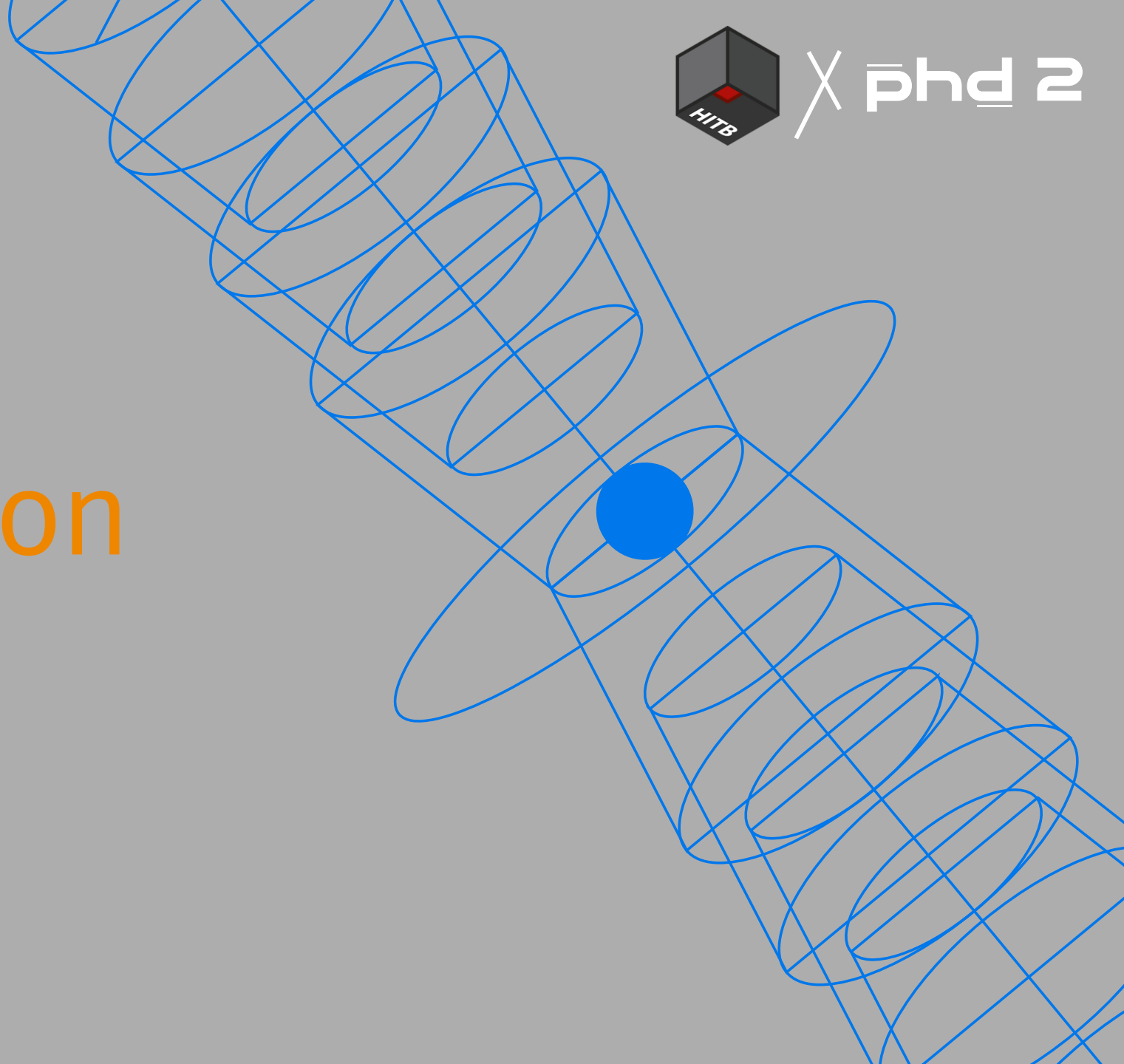
CVE-2018-20694,CVE-2018-20746,CVE-2018-20693,CVE-2018-20692,CVE-2018-20696,CVE-2018-20689,CVE-2018-20690,CVE-2018-10812,CVE-2019-6184,CVE-2019-6186,CVE-2019-6487,CVE-2019-1253,CVE-2019-1292,CVE-2019-1317,CVE-2019-1340,CVE-2019-1342,CVE-2019-1374,CVE-2019-8162,CVE-2019-1474,CVE-2019-18371,CVE-2019-18370,CVE-2020-0616,CVE-2020-0635,CVE-2020-0636,CVE-2020-0638,CVE-2020-0641,CVE-2020-0648,CVE-2020-0697,CVE-2020-0730,CVE-2020-3808,CVE-2020-0747,CVE-2020-0753,CVE-2020-0754,CVE-2020-0777,CVE-2020-0780,CVE-2020-0785,CVE-2020-0786,CVE-2020-0789,CVE-2020-0794,CVE-2020-0797,CVE-2020-0800,CVE-2020-0805,CVE-2020-0808,CVE-2020-0819,CVE-2020-0822,CVE-2020-0835,CVE-2020-0841,CVE-2020-0844,CVE-2020-0849,CVE-2020-0854,CVE-2020-0858,CVE-2020-0863,CVE-2020-0864,CVE-2020-0865,CVE-2020-0868,CVE-2020-0871,CVE-2020-0896,CVE-2020-0897,CVE-2020-0899,CVE-2020-0900,CVE-2020-0934,CVE-2020-0935,CVE-2020-0936,CVE-2020-0942,CVE-2020-0944,CVE-2020-0983,CVE-2020-0985,CVE-2020-0989,CVE-2020-1000,CVE-2020-1002,CVE-2020-1010,CVE-2020-1011,CVE-2020-1029,CVE-2020-1068,CVE-2020-1077,CVE-2020-1084,CVE-2020-1086,CVE-2020-1090,CVE-2020-1094,CVE-2020-1109,CVE-2020-1120,CVE-2020-1121,CVE-2020-1123,CVE-2020-1124,CVE-2020-1125,CVE-2020-1131,CVE-2020-1134,CVE-2020-1137,CVE-2020-1139,CVE-2020-1144,CVE-2020-1146,CVE-2020-1151,CVE-2020-1155,CVE-2020-1156,CVE-2020-1157,CVE-2020-1158,CVE-2020-1163,CVE-2020-1164,CVE-2020-1165,CVE-2020-1166,CVE-2020-1184,CVE-2020-1185,CVE-2020-1186,CVE-2020-1187,CVE-2020-1188,CVE-2020-1189,CVE-2020-1190,CVE-2020-1191,CVE-2020-1196,CVE-2020-1199,CVE-2020-1201,CVE-2020-1204,CVE-2020-1209,CVE-2020-1211,CVE-2020-1217,CVE-2020-1222,CVE-2020-1231,CVE-2020-1233,CVE-2020-1235,CVE-2020-1244,CVE-2020-1257,CVE-2020-1264,CVE-2020-1269,CVE-2020-1270,CVE-2020-1273,CVE-2020-1274,CVE-2020-1276,CVE-2020-1277,CVE-2020-1278,CVE-2020-1282,CVE-2020-1283,CVE-2020-1304,CVE-2020-1305,CVE-2020-1306,CVE-2020-1307,CVE-2020-1309,CVE-2020-1312,CVE-2020-1317,CVE-2020-1337,CVE-2020-1344,CVE-2020-1346,CVE-2020-1347,CVE-2020-1352,CVE-2020-1356,CVE-2020-1357,CVE-2020-1360,CVE-2020-1361,CVE-2020-1362,CVE-2020-1364,CVE-2020-1366,CVE-2020-1372,CVE-2020-1373,CVE-2020-1375,CVE-2020-1385,CVE-2020-1392,CVE-2020-1393,CVE-2020-1394,CVE-2020-1399,CVE-2020-1404,CVE-2020-1405,CVE-2020-1424,CVE-2020-1427,CVE-2020-1441,CVE-2020-0518,CVE-2020-1461,CVE-2020-1465,CVE-2020-1472,CVE-2020-1474,CVE-2020-1475,CVE-2020-1484,CVE-2020-1485,CVE-2020-1511,CVE-2020-1512,CVE-2020-0516,CVE-2020-1516,CVE-2020-1517,CVE-2020-1518,CVE-2020-1519,CVE-2020-1521,CVE-2020-1522,CVE-2020-1524,CVE-2020-1528,CVE-2020-1538,CVE-2020-8741,CVE-2020-1548,CVE-2020-1549,CVE-2020-1550,CVE-2020-1552,CVE-2020-1590,CVE-2020-1130,CVE-2020-16851,CVE-2020-16852,CVE-2020-1122,CVE-2020-1038,CVE-2020-17089,CVE-2020-16853,CVE-2020-16879,CVE-2020-16900,CVE-2020-16980,CVE-2020-17014,CVE-2020-17070,CVE-2020-17073,CVE-2020-17074,CVE-2020-17075,CVE-2020-17076,CVE-2020-17077,CVE-2020-17092,CVE-2020-17097,CVE-2020-17120,CVE-2021-1649,CVE-2021-1650,CVE-2021-1651,CVE-2021-1659,CVE-2021-1680,CVE-2021-1681,CVE-2021-1686,CVE-2021-1687,CVE-2021-1688,CVE-2021-1689,CVE-2021-1690,CVE-2021-1718,CVE-2021-1722,CVE-2021-24072,CVE-2021-24077,CVE-2021-3750,CVE-2021-24088,CVE-2021-26869,CVE-2021-26870,CVE-2021-26871,CVE-2021-26885,CVE-2021-28347,CVE-2021-28351,CVE-2021-28436,CVE-2021-28450,CVE-2021-31966,CVE-2021-34527,CVE-2021-42321,CVE-2021-36970,CVE-2021-38657,CVE-2021-40485,CVE-2021-41366,CVE-2021-42294,CVE-2021-42297,CVE-2021-43216,CVE-2021-43223,CVE-2021-43248,CVE-2022-21835,CVE-2022-21837,CVE-2022-21878,CVE-2022-21881,CVE-2022-21888,CVE-2022-21971,CVE-2022-21974,CVE-2022-21992,CVE-2022-23285,CVE-2022-23290,CVE-2022-24454,CVE-2022-29108,CVE-2022-24547,CVE-2022-23270,CVE-2022-26930,CVE-2022-29103,CVE-2022-29113,CVE-2022-38036,CVE-2022-35793,CVE-2022-35755,CVE-2022-35749,CVE-2022-35746,CVE-2022-34690,CVE-2022-21980,CVE-2022-22050,CVE-2022-22024,CVE-2022-22022,CVE-2022-30226,CVE-2022-30157,CVE-2022-29108,CVE-2022-21999,CVE-2023-21683,CVE-2023-21684,CVE-2023-21693,CVE-2023-21801,CVE-2023-23403,CVE-2023-23406,CVE-2023-23413,CVE-2023-24856,CVE-2023-24857,CVE-2023-24858,CVE-2023-24863,CVE-2023-24865,CVE-2023-24866,CVE-2023-24867,CVE-2023-24907,CVE-2023-24868,CVE-2023-24909,CVE-2023-24870,CVE-2023-24872,CVE-2023-24913,CVE-2023-24876,CVE-2023-24924,CVE-2023-24883,CVE-2023-24925,CVE-2023-24884,CVE-2023-24926,CVE-2023-24885,CVE-2023-24927,CVE-2023-24886,CVE-2023-24928,CVE-2023-24887,CVE-2023-24929,CVE-2023-28243,CVE-2023-28296,CVE-2023-29366,CVE-2023-29367,CVE-2023-32017,CVE-2023-32039,CVE-2023-32040,CVE-2023-32041,CVE-2023-32042,CVE-2023-32085,CVE-2023-35296,CVE-2023-35302,CVE-2023-35306,CVE-2023-35313,CVE-2023-35323,CVE-2023-35324,CVE-2023-36898,CVE-2023-36792,CVE-2023-36704,CVE-2023-36418,CVE-2023-36395,CVE-2023-36393,CVE-2023-35624,CVE-2023-21683,CVE-2023-29366,CVE-2023-46138,CVE-2023-42820,CVE-2023-42819,CVE-2024-21426,CVE-2024-29156,CVE-2024-26198,CVE-2024-21435,CVE-2024-21329,CVE-2024-21384,CVE-2024-20691,CVE-2024-21433,CVE-2024-20694, ,CVE-2024-0087,CVE-2024-0088,CVE-2024-30060,CVE-2024-29989

# Agenda

- Introduction
- Attack Surface of Different Sandboxes on Windows
- Exploiting the Chrome GPU Process
- Exploiting a Windows Kernel Vulnerability
- Summary

01

# Introduction



# Background

- ➢ Modern desktop application become more secure
  - SDLC, Mitigations, Security Architecture, ……
- ➢ Chrome and Adobe pdf Reader are still the main targets
  - 0day exploitation still in the wild.
  - Commercial Surveillance Vendors and Governments.

# Motivation

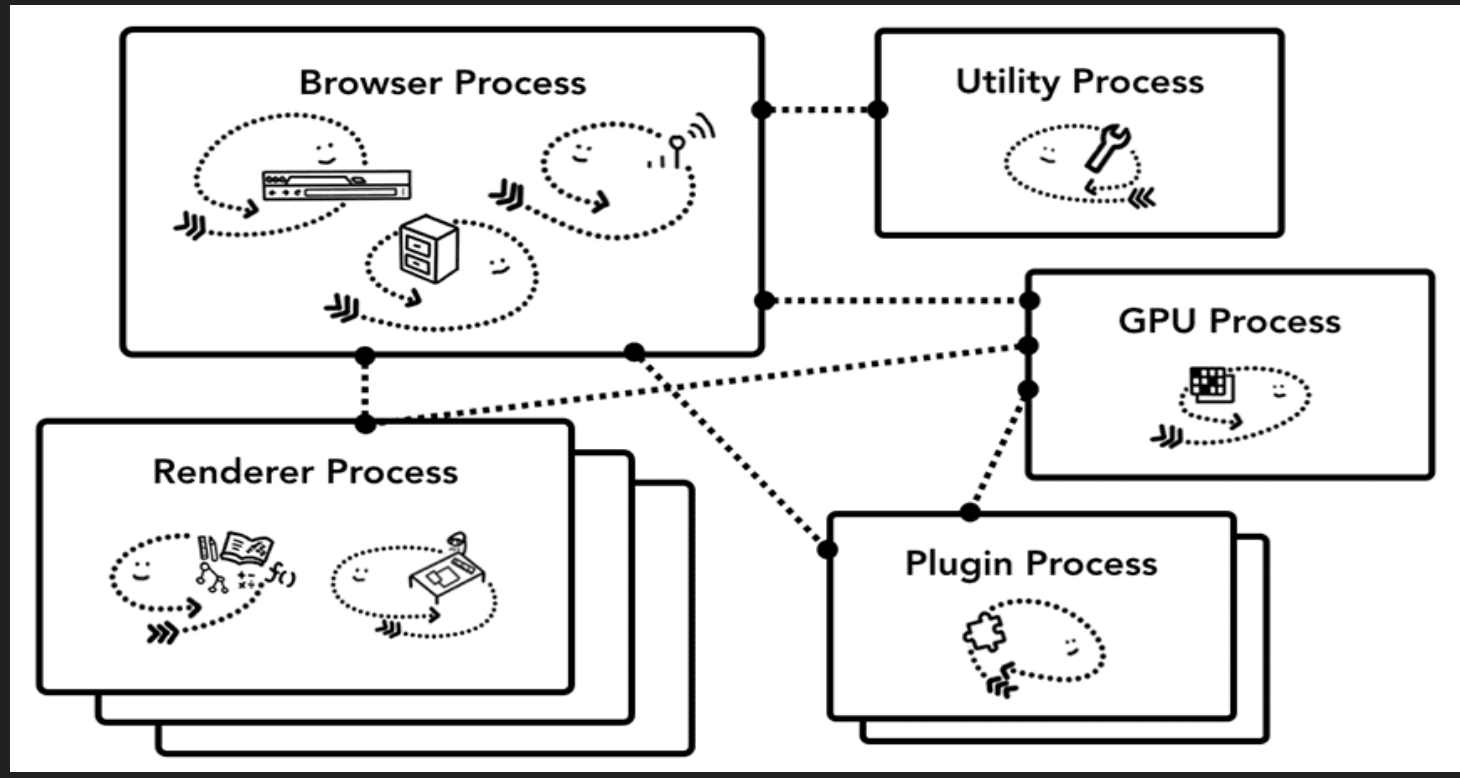
- ➢ Offensive research drive defense
  - Defense and detection
  - Attack simulation (BAS)
- ➢ Build a full chain exploit for Chrome and Adobe pdf Reader
  - No Experience on Chrome and Adobe before
  - On PC (Windows)
  - Sandbox escape is the main obstacle
  - In this talk: our journey of research on sandbox escape

02

# Attack Surface of Different Sandboxes

# Chrome Process Architecture

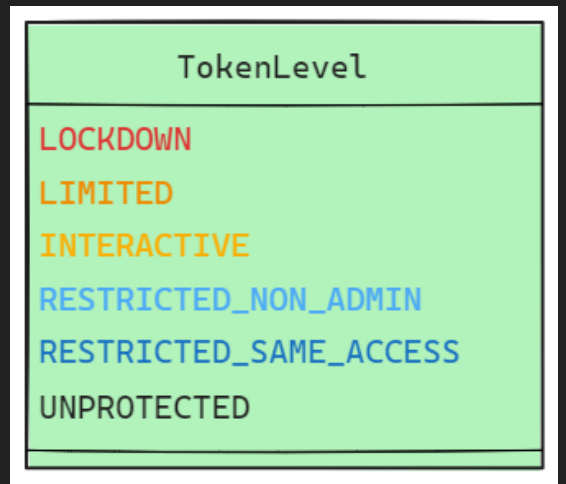
- Multi-process architecture.
- Different kind of process for different features.
- Talk to each other using Inter Process Communication (IPC).
- Sandbox restrict processes.

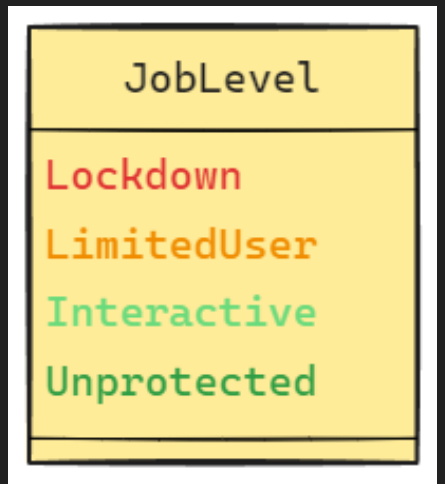


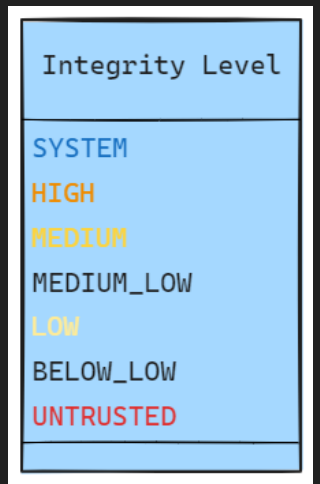

https://developer.chrome.com/blog/inside-browser-part1

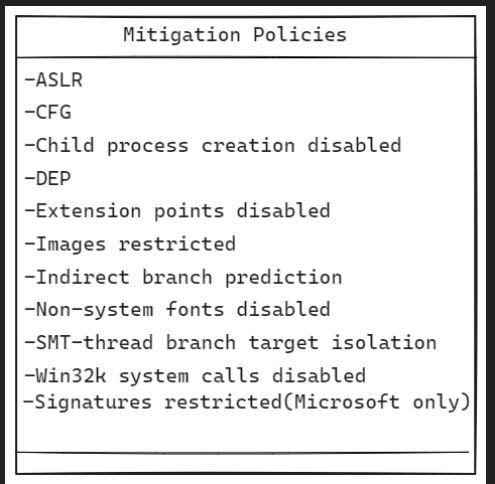
# How Sandbox Restrict Process on Windows

- Restricted Token, Job Level.
  - Restricted privileges, protecting securable resources.
- Integrity Levels.
  - Enforce mandatory access control.
- Mitigations.
  - Setup various security enforcing policies.
- Alternate Desktop.
  - Restrict sandbox to interact with user desktop.



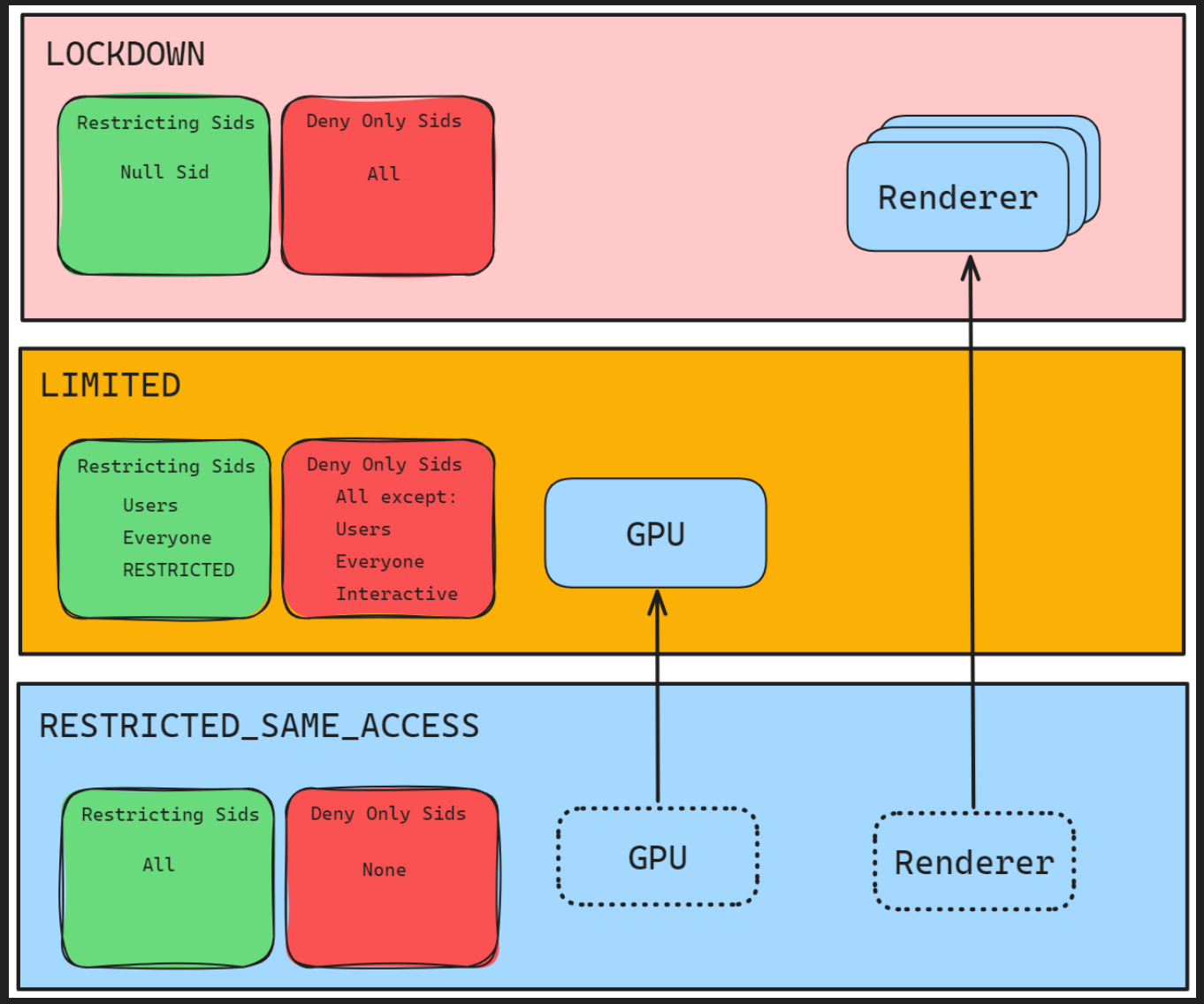



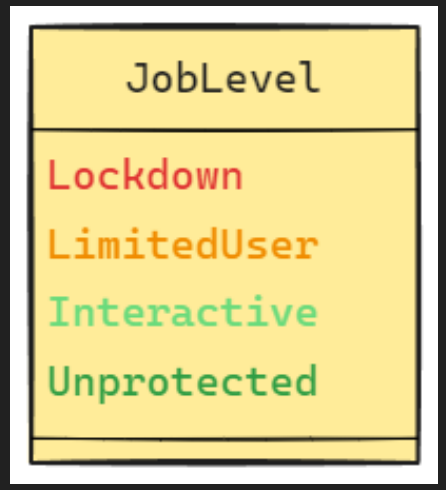
# Restricted Token

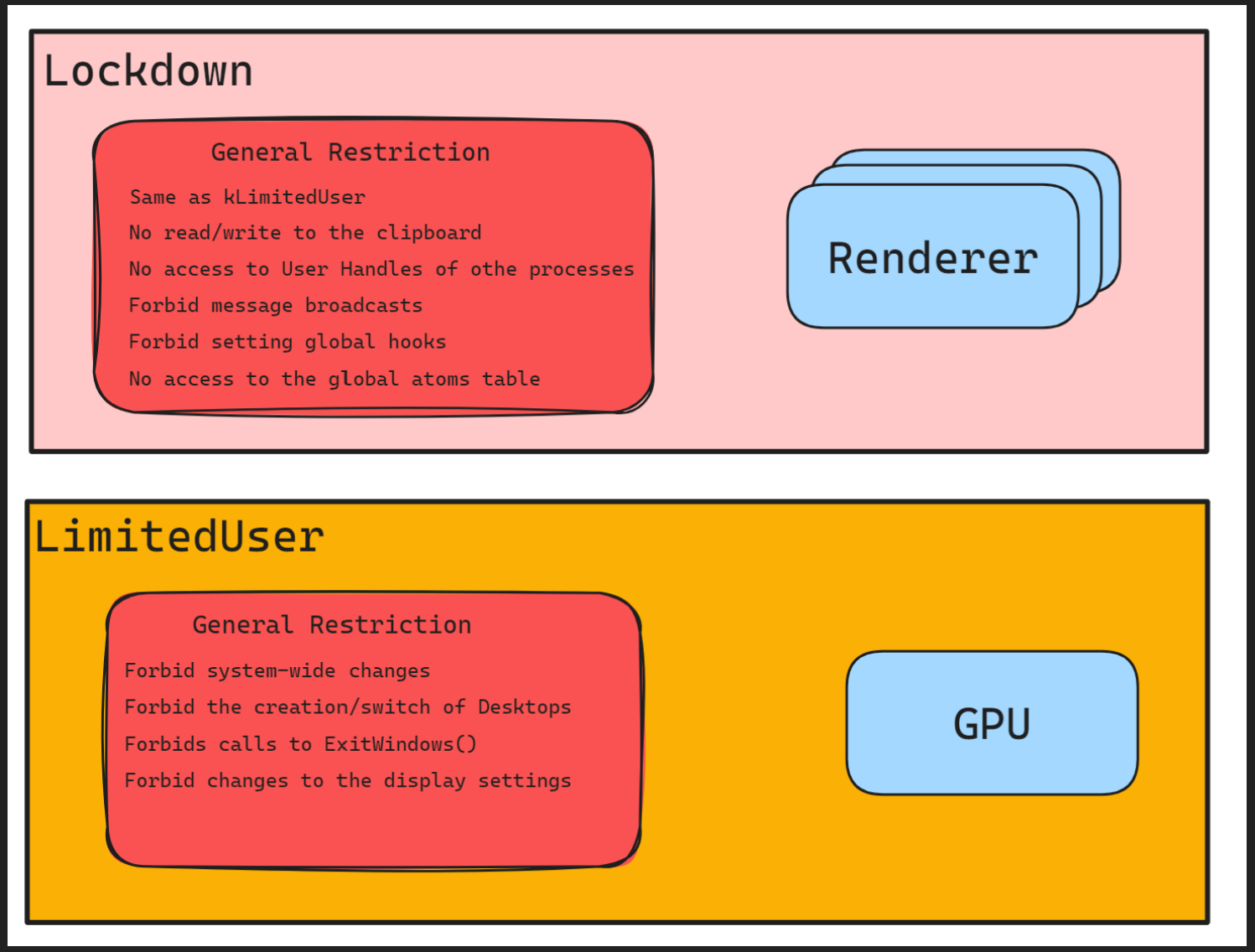
| TokenLevel |
| --- |
| LOCKDOWN |
| LIMITED |
| INTERACTIVE |
| RESTRICTED_NON_ADMIN |
| RESTRICTED_SAME_ACCESS |
| UNPROTECTED |




Restricting sids enforce limits on sandbox process for accessing resources which depends on sid.

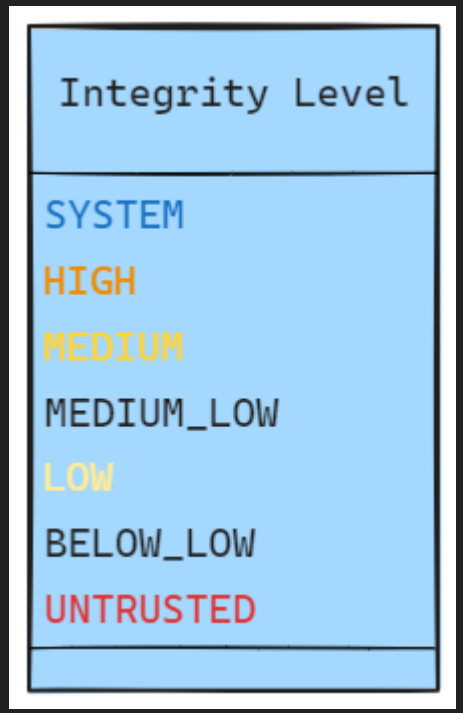
# Job Level

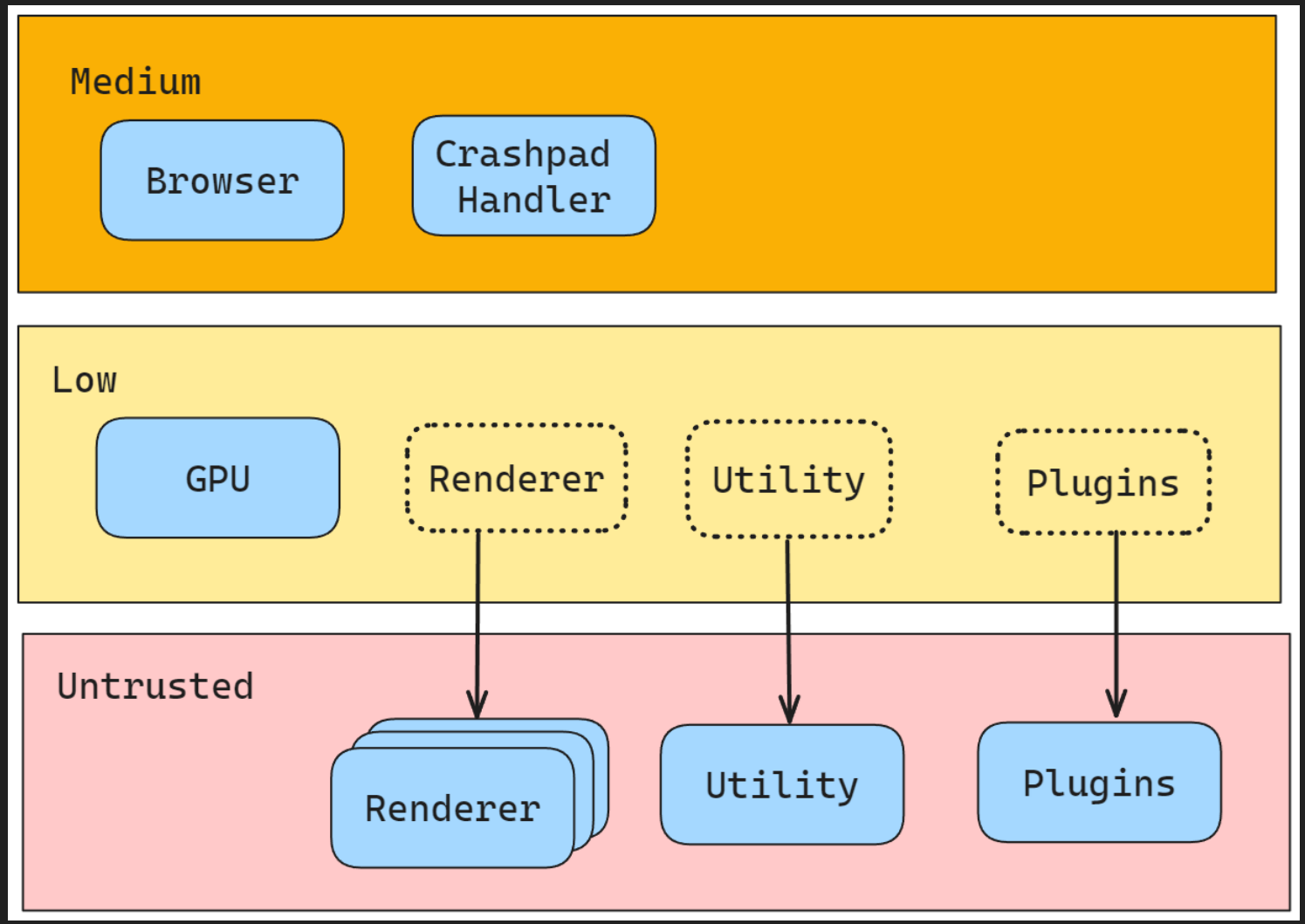



A job can enforce limits on sandbox process for operating on global resources.

# Integrity Level

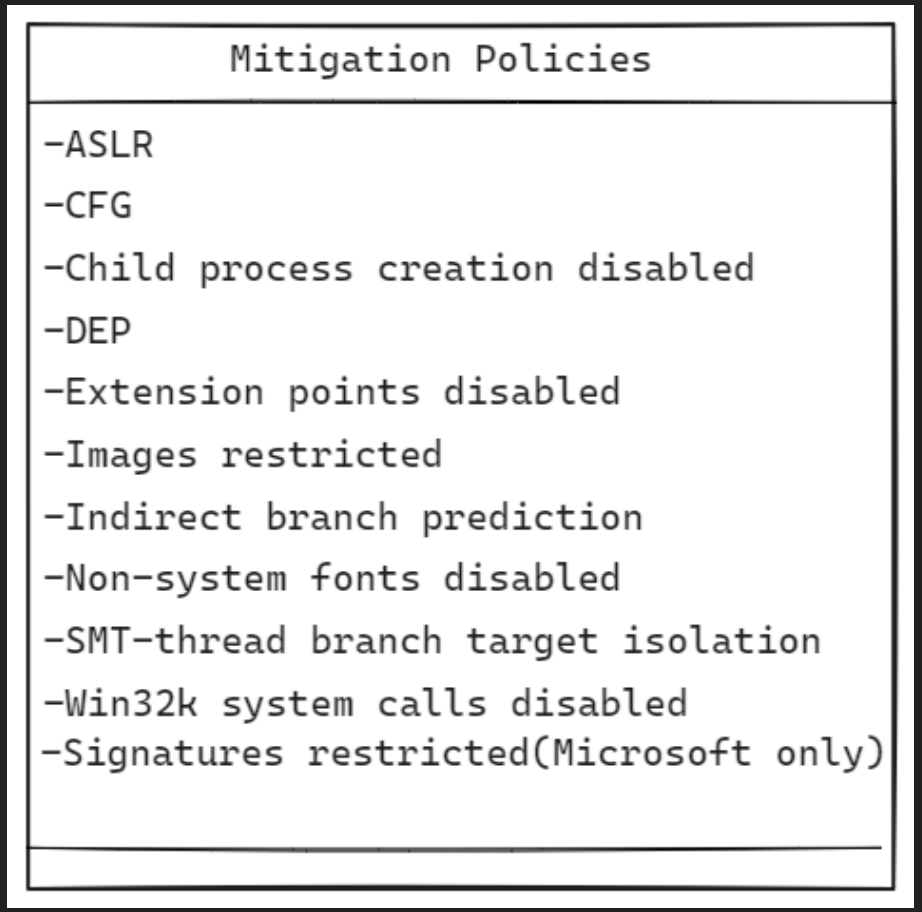


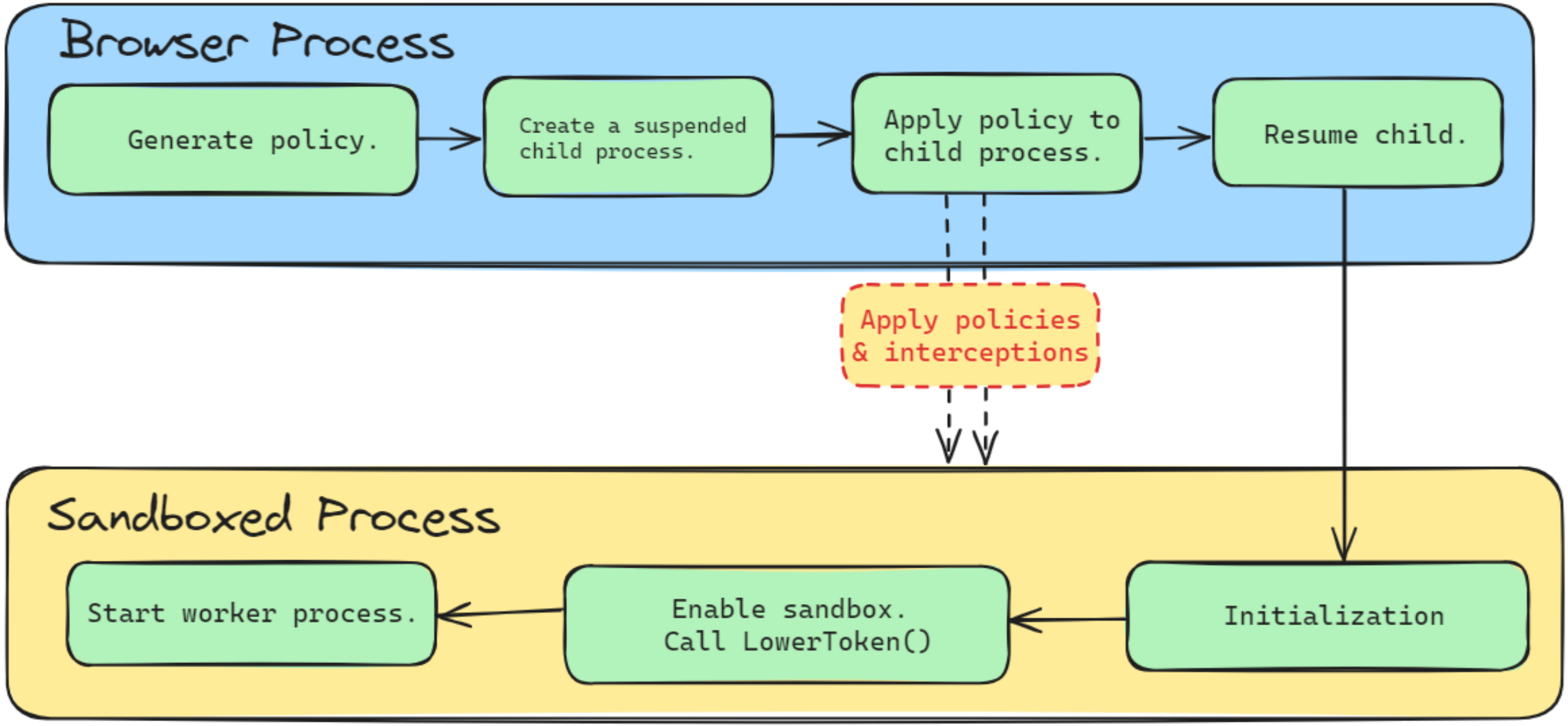

Integrity level enforce mandatory access control.

# Mitigation Policies

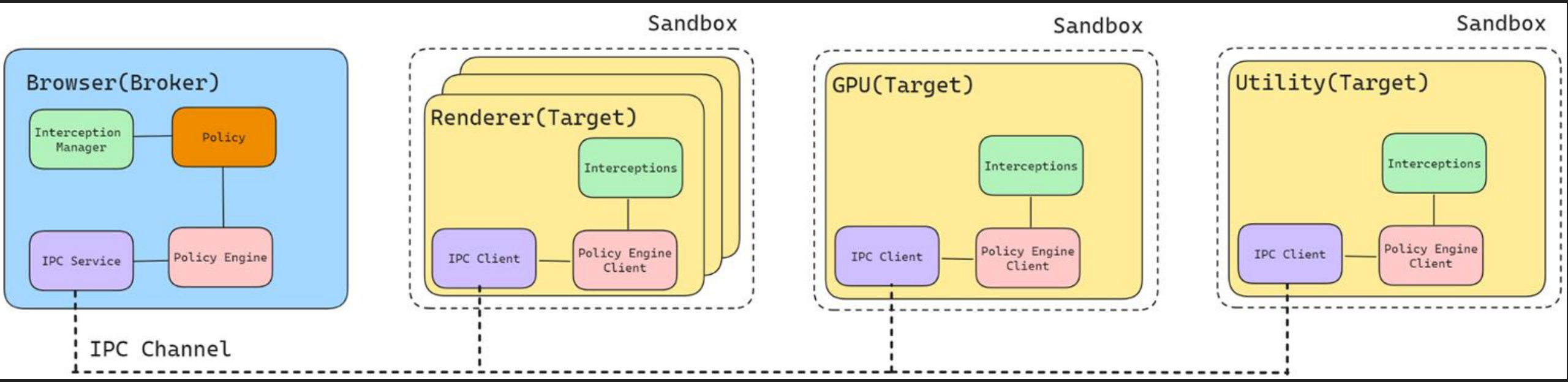
- Security enforcing policies to sandbox.
  - Most can be applied via SetProcessMitigationPolicy.



# Chrome Sandbox Startup Flow on Windows



# Chrome Sandbox Architecture



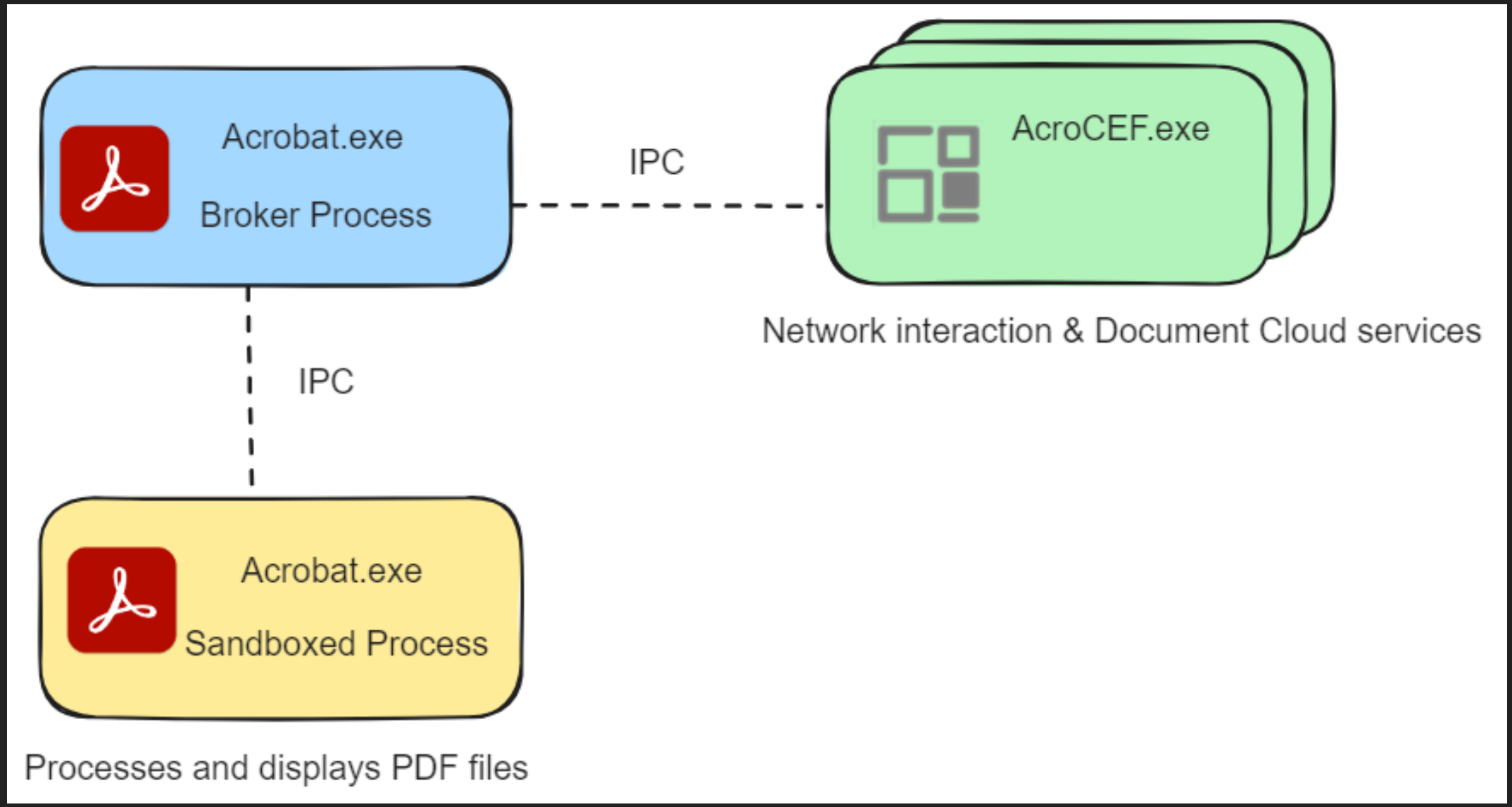
# ➢ chrome://sandbox

Chrome chrome://sandbox

## Sandbox Status

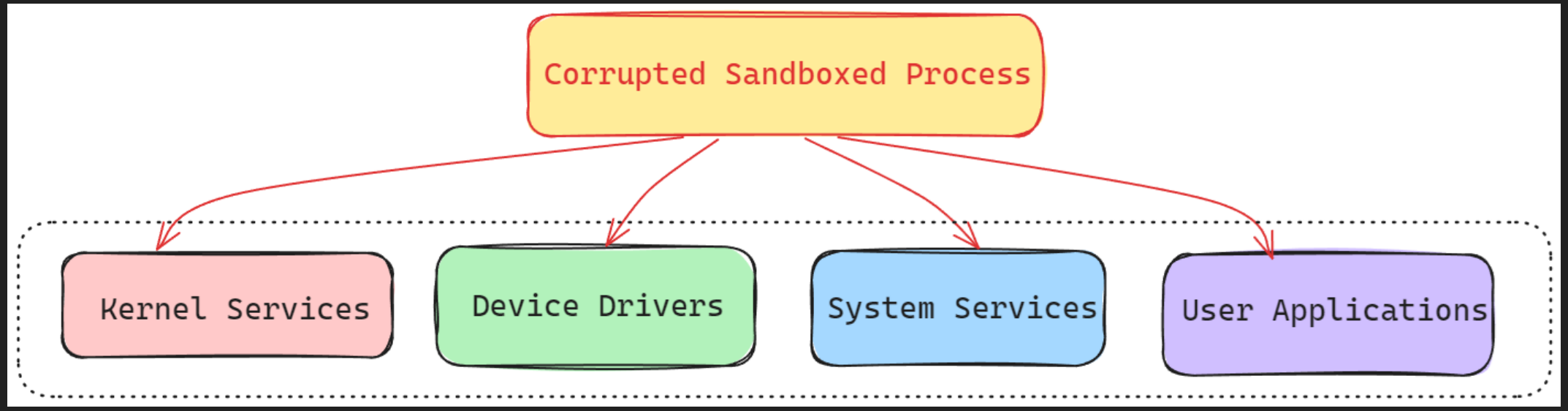
| Process | Type | Name | Sandbox | Lockdown | Integrity | Mitigations | Component Filter | Lowbox/AppContainer |
|---|---|---|---|---|---|---|---|---|
| 20840 | GPU | GPU | GPU | Limited | S-1-16-4096 Low | 0111100100011000010000000010000<br>HEAP_TERMINATE<br>BOTTOM_UP_ASLR<br>EXTENSION_POINT_DISABLE<br>BLOCK_NON_MICROSOFT_BINARIES<br>FONT_DISABLE<br>IMAGE_LOAD_NO_REMOTE<br>IMAGE_LOAD_NO_LOW_LABEL<br>RESTRICT_INDIRECT_BRANCH_PREDICTION<br>FSCTL_SYSTEM_CALL_DISABLE | 00000001 | |
| 20852 | Utility | Network Service | Not Sandboxed | | | | | |
| 20944 | Utility | Storage Service | Service | Lockdown | S-1-16-0 Untrusted | 0111101110011000010000000010000<br>HEAP_TERMINATE<br>BOTTOM_UP_ASLR<br>WIN32K_SYSTEM_CALL_DISABLE<br>EXTENSION_POINT_DISABLE<br>PROHIBIT_DYNAMIC_CODE<br>BLOCK_NON_MICROSOFT_BINARIES<br>FONT_DISABLE<br>IMAGE_LOAD_NO_REMOTE<br>IMAGE_LOAD_NO_LOW_LABEL<br>RESTRICT_INDIRECT_BRANCH_PREDICTION<br>FSCTL_SYSTEM_CALL_DISABLE | 00000001 | |
| 23356 | Utility | Audio Service | Audio | Restricted Non Admin | S-1-16-4096 Low | 0111101100011000010000000010000<br>HEAP_TERMINATE<br>BOTTOM_UP_ASLR<br>EXTENSION_POINT_DISABLE<br>PROHIBIT_DYNAMIC_CODE<br>BLOCK_NON_MICROSOFT_BINARIES<br>FONT_DISABLE<br>IMAGE_LOAD_NO_REMOTE<br>IMAGE_LOAD_NO_LOW_LABEL<br>RESTRICT_INDIRECT_BRANCH_PREDICTION<br>FSCTL_SYSTEM_CALL_DISABLE | 00000001 | |
| 8464 | Renderer | | Renderer | Lockdown | S-1-16-0 Untrusted | 0111100110011000010000020010000 | 00000001 | |

# Adobe Process Architecture



# Sandbox Escape Methodology

- Resources accessible inside the sandbox.
  - Configuration Issues may lead to direct sandbox escape.
  - Policy auditing.
- Code that can interactive inside the sandbox.
  - Vulnerability in these code might cause a sandbox escape.
  - The more resources you can access, the more code you can interact with.



# Resources Accessible inside Sandbox

Depends on three factors：

Token/Job/Mitigations: Chrome uses Mitigation Policy, Token, Job, Desktop to restrict the behavior of sandbox on Windows.

Pre-opened Objects: Sandboxed process has pre-opened objects which are necessary for the sandbox to run normally.

Policy Rules: Chrome provides the Policy Rule to allow the sandbox to access extra system resources.

# Accessible Resources by Token/Job

## Get Writable Directories

```
Get-AccessibleFile -Win32Path "C:\" -Recurse -ProcessIds 1234 -DirectoryAccessRights AddFile -CheckMode
DirectoriesOnly -FormatWin32Path | Select-Object Name
```

## Get Accessible ALPC Port

```
Get-AccessibleAlpcPort -ProcessIds 1234
```

## Get Accessible Device Object

```
Get-AccessibleDevice \Device -ProcessIds 1234
```

Install-Module -Name NtObjectManager

# Pre-opened object in Renderer

Get pre-opened object by System Informer

| | | |
|---|---|---|
| File | \Device\CNG | Read data, Synchronize |
| File | \Device\KsecDD | Read data, Write data, S... |
| File | C:\Program Files\Google\Chrome\Application\122.0.6261.112\icudtl.dat | Read |
| File | C:\Windows\apppatch\DirectXApps.sdb | Read |
| File | \Device\DeviceApi | Read |
| File | \Device\NamedPipe\mojo.3032.3852.4128734811961643837 | Write, Read, Write owner |
| File | \Device\NamedPipe\mojo.3032.3852.5633591254788134162 | Write, Read, Write owner |
| File | \Device\NamedPipe\mojo.3032.3852.3066321487145621440 | Write, Read, Write owner |
| File | \Device\NamedPipe\mojo.3032.11440.9905790863115651483 | Write, Read |
| Key | HKLM\SYSTEM\ControlSet001\Control\Nls\Sorting\Versions | Read |
| Key | HKLM | Read |
| Key | HKLM\SYSTEM\ControlSet001\Control\Session Manager | Query values |
| Key | HKLM\SYSTEM\ControlSet001\Control\Nls\Sorting\Ids | Read |
| Key | HKLM | Read |
| Key | HKLM\SOFTWARE\Microsoft\Ole | Read |
| Key | HKCU\Software\Classes\Local Settings\Software\Microsoft | Read |
| Key | HKCU\Software\Classes\Local Settings | Read |

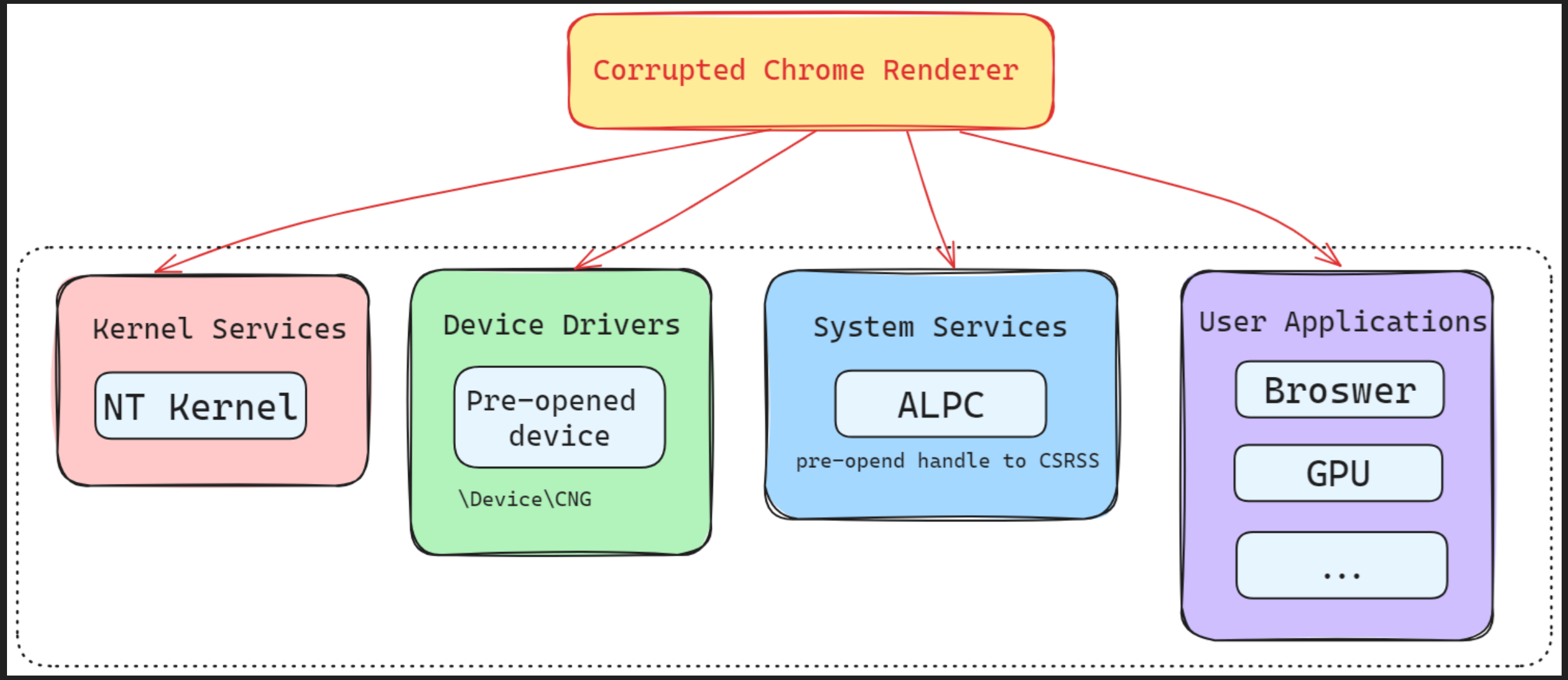
IPC based on named pipe

# Extra Resources

- Ask broker process for the extra resources
  - Crosscall answers are rule-based.
  - Get policy rules from chrome://sandbox.

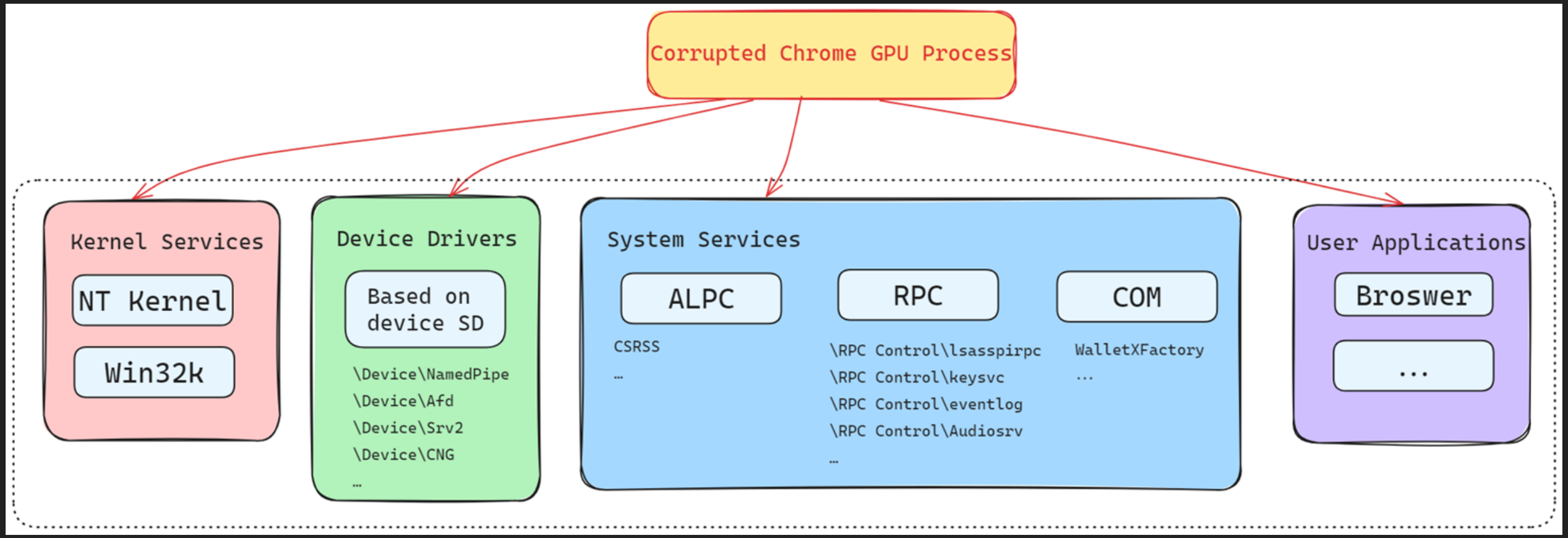
```
"policyRules": {
  "GdiDllInitialize": [
    " -> fakeSuccess"
  ],
  "GetStockObject": [
    " -> fakeSuccess"
  ],
  "NtCreateSection": [
    "exact(p[0], '\\Device\\HarddiskVolume3\\Program Files\\Google\\Chrome\\Application\\122.0.6261.128\\chrome.dll') -> askBroker",
    "exact(p[0], '\\Device\\HarddiskVolume3\\Program Files\\Google\\Chrome\\Application\\122.0.6261.128\\chrome_elf.dll') -> askBroker"
  ],
  "RegisterClassW": [
    " -> fakeSuccess"
  ]
},
```

# Code You Can Interact With (Attack Surface)




**Every attack surface listed here has vulnerabilities exploited in the wild before.**

# Attack Surface of Chrome GPU Process

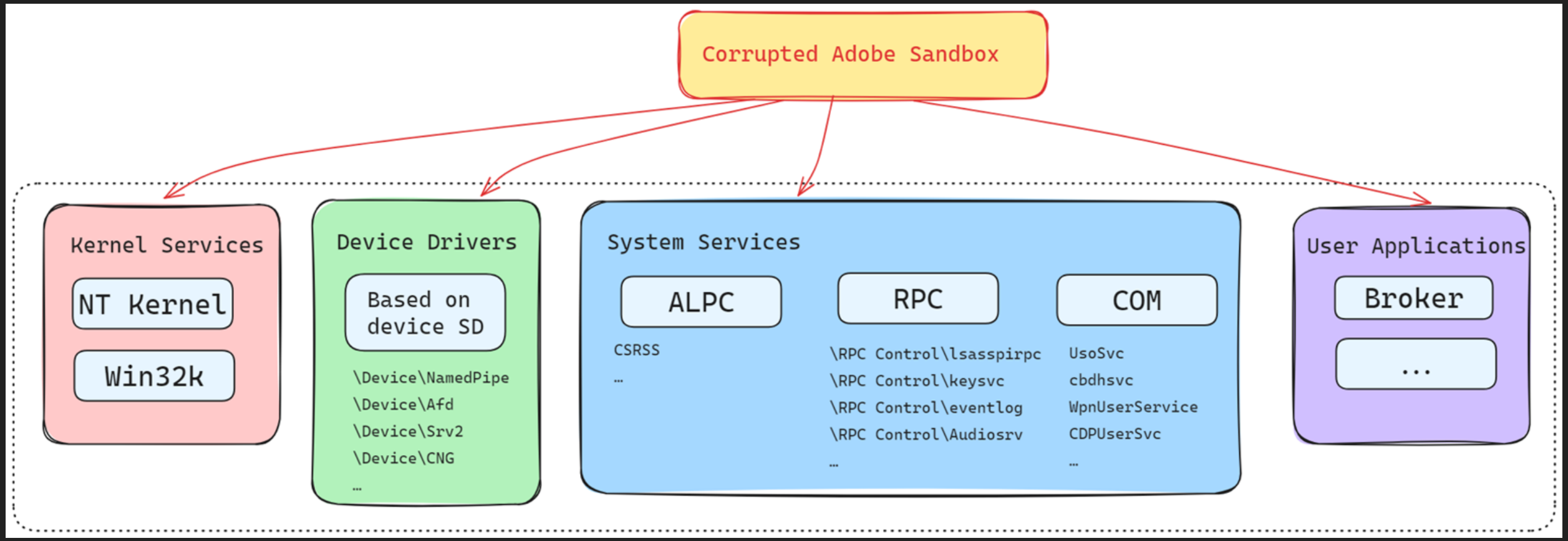



**Comparing with renderer:**

**having the same syscall in NT kernel, but you can interact with more code.**

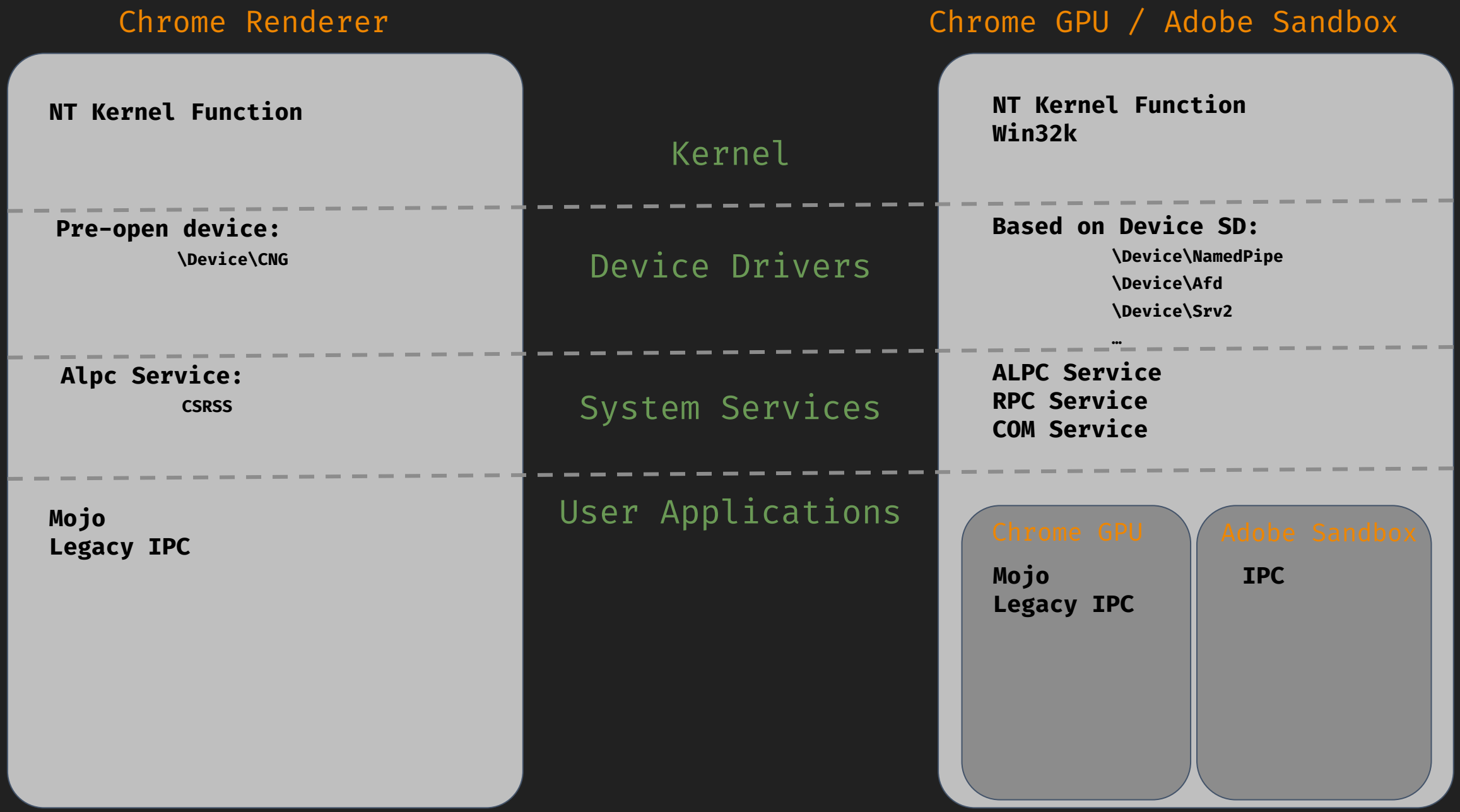
**Easier to escape.**

# Attack Surface of Adobe Sandboxed Process

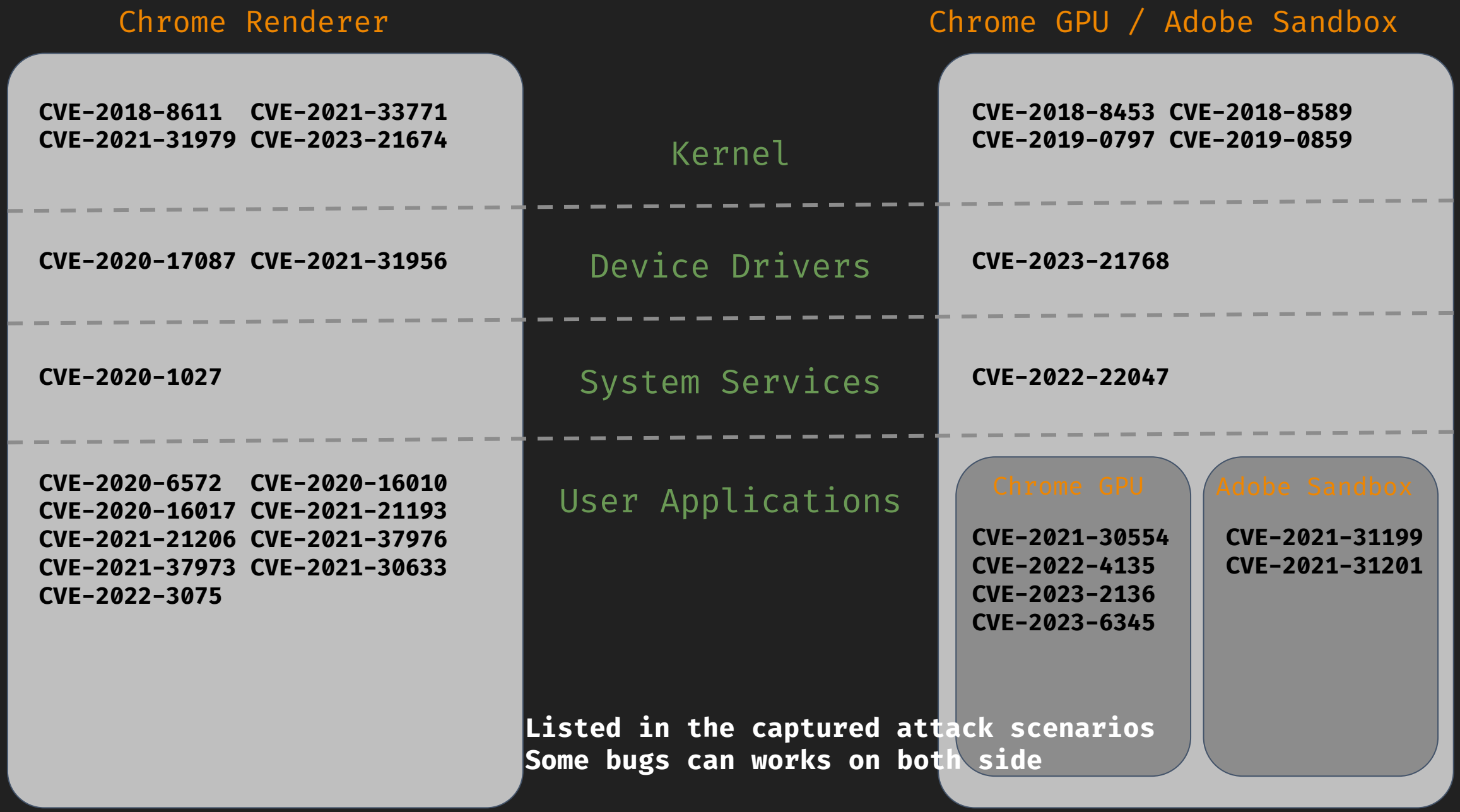



**A little bit similar with chrome GPU sandbox**

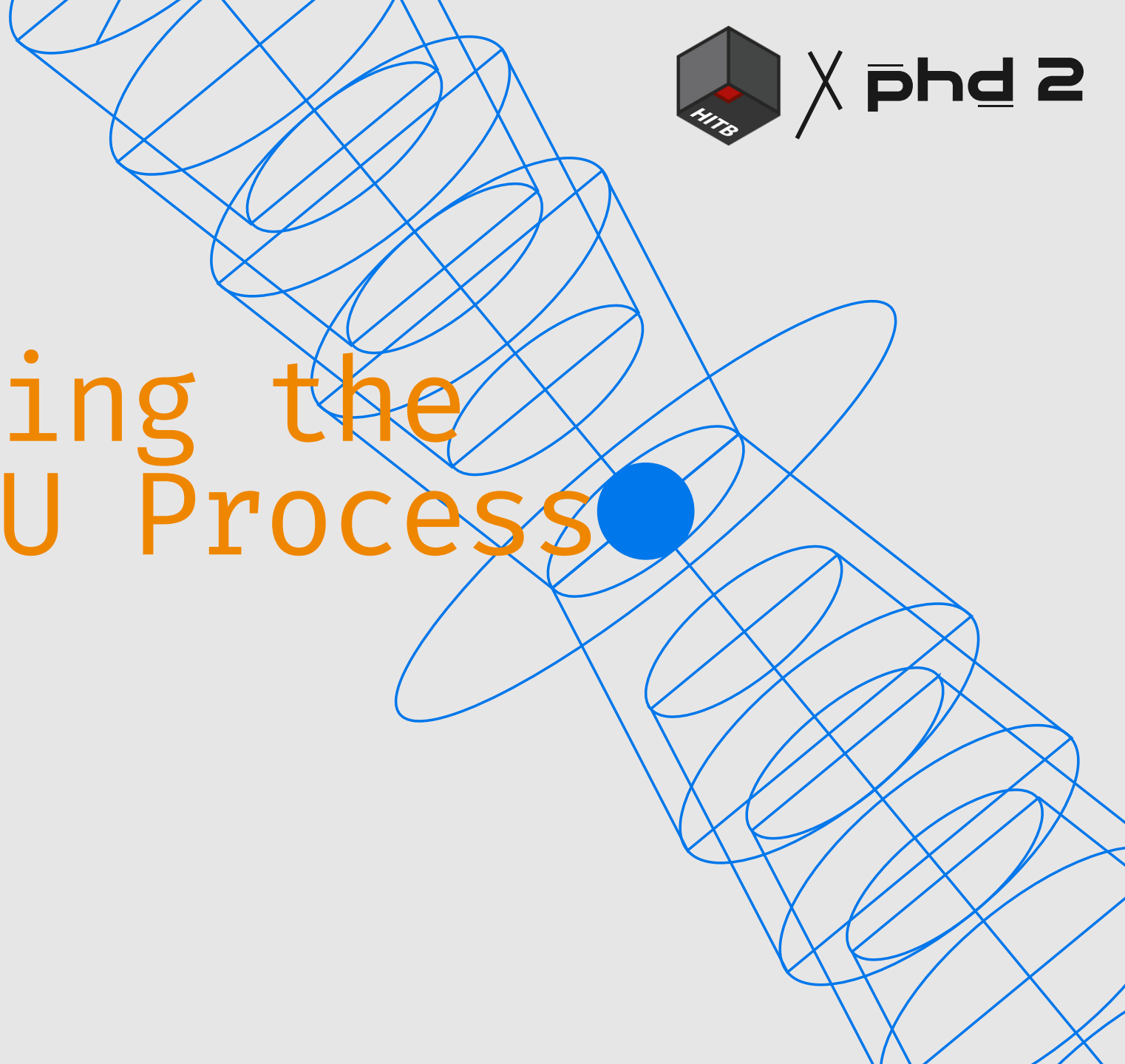
# Comparison


Chrome Renderer
Chrome GPU / Adobe Sandbox
NT Kernel Function
Kernel
NT Kernel Function
Win32k
Pre-open device:
\Device\CNG
Device Drivers
Based on Device SD:
\Device\NamedPipe
\Device\Afd
\Device\Srv2
…
Alpc Service:
CSRSS
System Services
ALPC Service
RPC Service
COM Service
Mojo
Legacy IPC
User Applications
Chrome GPU
Mojo
Legacy IPC
Adobe Sandbox
IPC

# In-the-wild Exploits



# 03



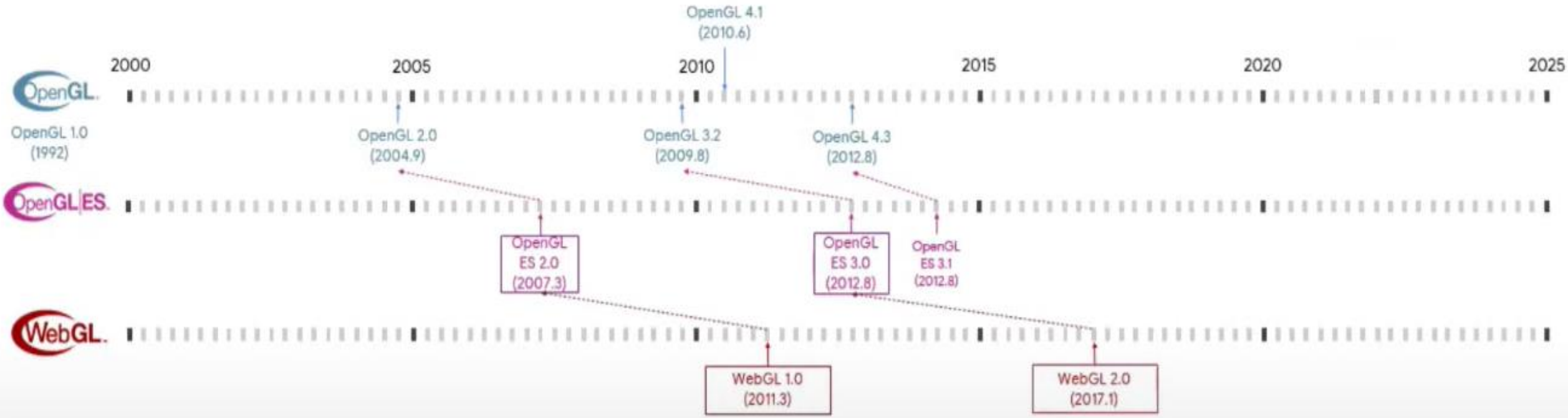
# Exploiting the Chrome GPU Process

# Introduction to WebGL

- Web standard for a low-level 3D graphics API.
- Browser-compatible JavaScript API.
- WebGL stays very close to the related OpenGL ES specification.
- …

# History of WebGL

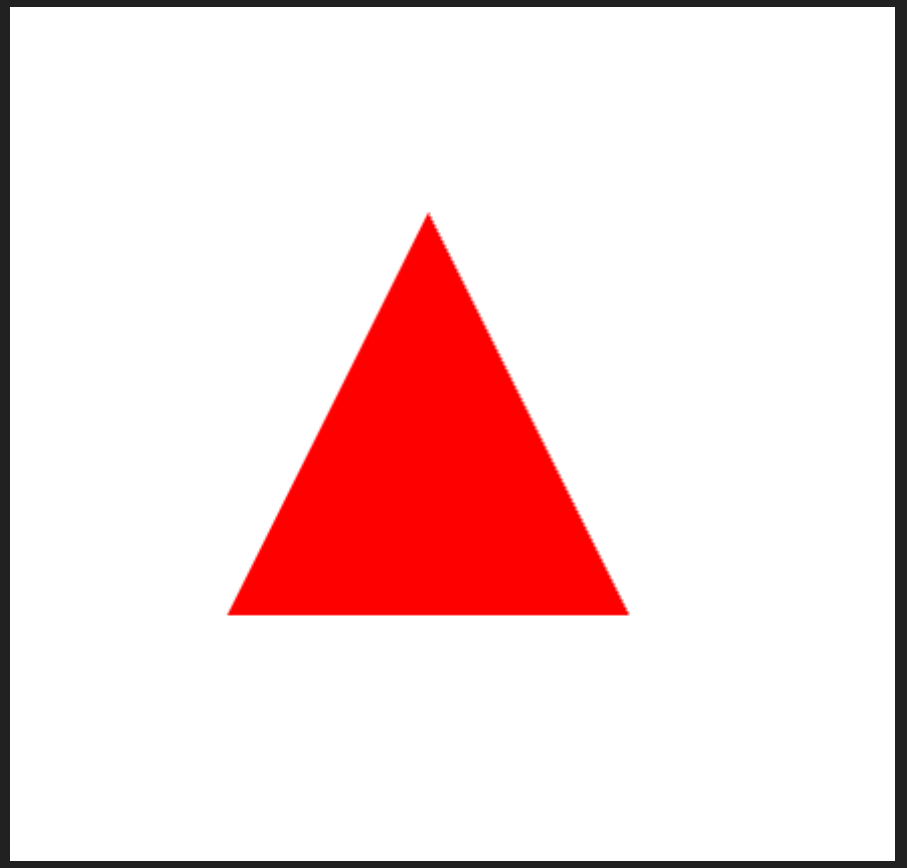
- WebGL 1.0 was released in 2011.
  - Based on OpenGL ES 2.0(2007).
- WebGL 2.0 was released in 2017.
  - Based on OpenGL ES 3.0(2012).



# WebGL Demo

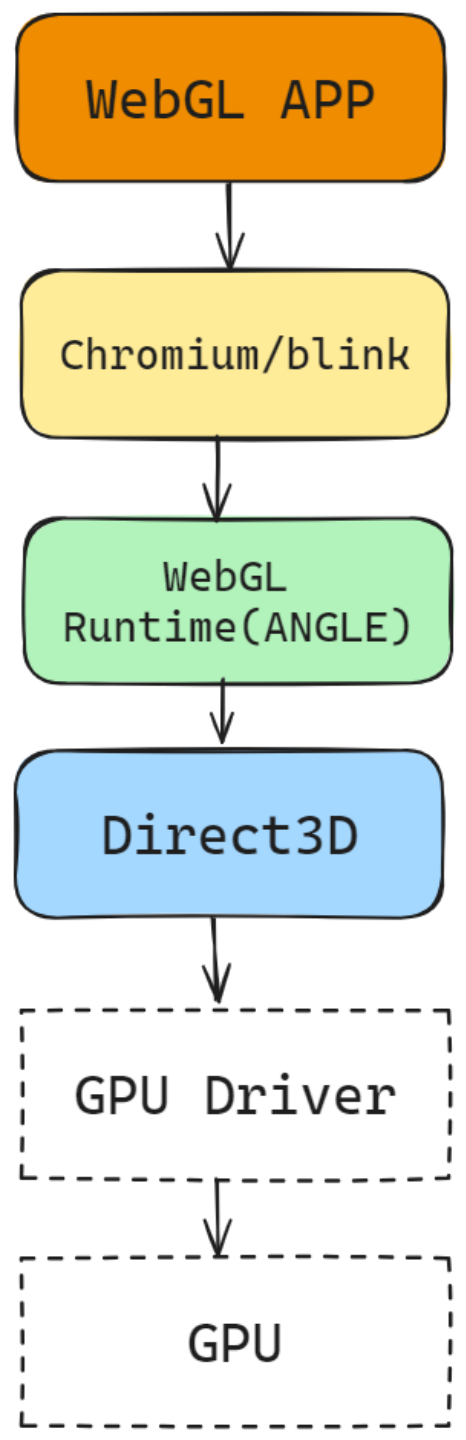
➢ Simple webgl2 code drawing a triangle

```
const vertices = new Float32Array([
    0.0, 0.5,   // Vertex 1 (top)
    -0.5, -0.5, // Vertex 2 (bottom-left)
    0.5, -0.5   // Vertex 3 (bottom-right)
]);

const positionBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);
gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);

const vsSource = `
    attribute vec2 a_position;
    void main() {
        gl_Position = vec4(a_position, 0.0, 1.0);
    }
`;

const fsSource = `
    void main() {
        gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0); // Red color
    }
`;

const program = createProgram(gl, compileShader(gl, vsSource, gl.VERTEX_SHADER),
    compileShader(gl, fsSource, gl.FRAGMENT_SHADER));

const positionLoc = gl.getAttribLocation(program, 'a_position');
gl.enableVertexAttribArray(positionLoc);
gl.vertexAttribPointer(positionLoc, 2, gl.FLOAT, false, 0, 0);

gl.useProgram(program);
gl.drawArrays(gl.TRIANGLES, 0, 3);
```



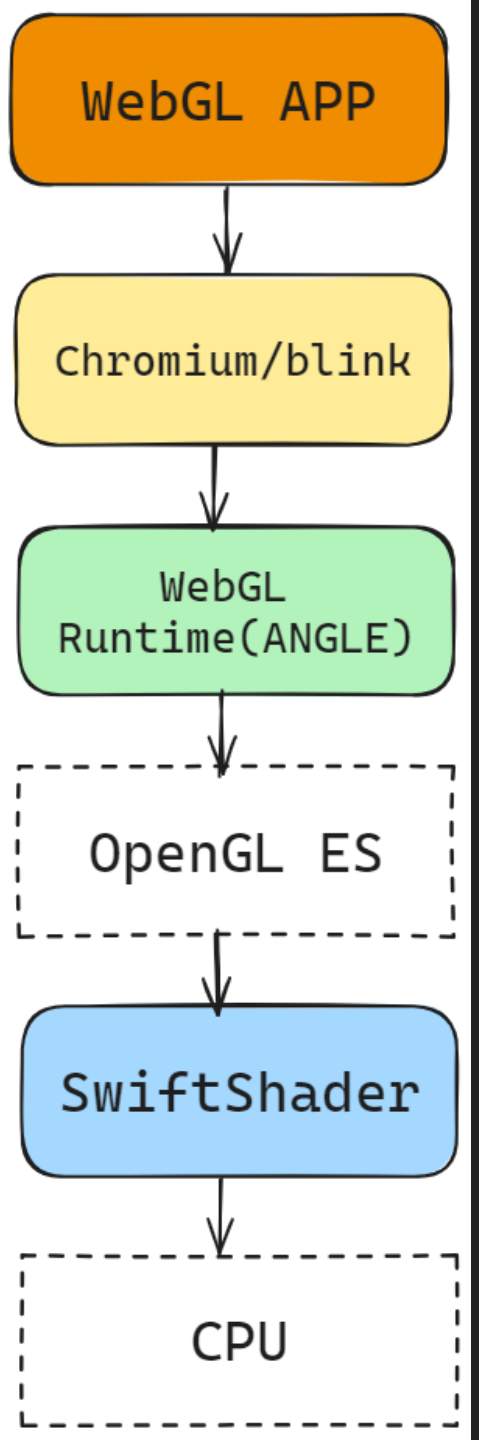
# Chrome GPU Architecture on Windows

- Using Hardware Acceleration.
- ANGLE calls the D3D API directly.



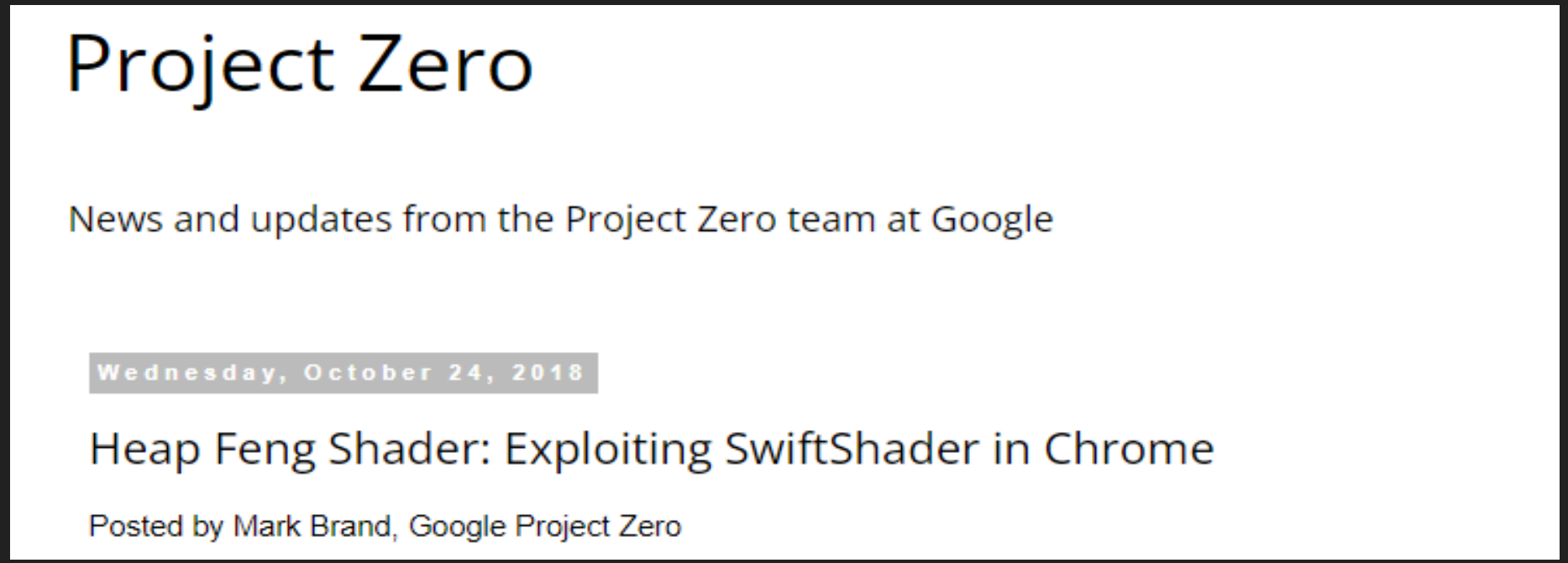
# Chrome GPU Architecture on Windows

- With SwiftShader enabled.
  - Pure software implementation.
  - Running on devices that do not support hardware acceleration.
  - Cross-platform features.



# What does swiftshader bring us?

- Attack surface from render to GPU process.




https://googleprojectzero.blogspot.com/2018/10/heap-feng-shader-exploiting-swiftshader.html

# Check swiftshader exists

```
const canvas = document.createElement('canvas');
const gl = canvas.getContext('webgl2');
const debugInfo = gl.getExtension("WEBGL_debug_renderer_info");
const renderer = gl.getParameter(debugInfo.UNMASKED_RENDERER_WEBGL);
console.log(renderer);
```

```
> ANGLE (Intel, Intel(R) UHD Graphics 630 (0x00003E92) Direct3D11
vs_5_0 ps_5_0, D3D11)
// without swiftshader


> ANGLE (Google, Vulkan 1.3.0 (SwiftShader Device (Subzero)
(0x0000C0DE)), SwiftShader driver)
// with swiftshader
```

# How to Enable Swiftshader

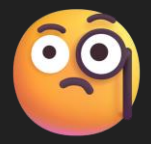
- Adding --disable-gpu to the chrome startup command line.
  - Not the default.
- Or turning off 3D acceleration in virtual machines.
  - Common in cloud platform, but not universal for physical machines.
- In other cases, the swiftshader backend is disabled.

So how can we enable swiftshader by default in stable chrome?

# Enable Swiftshader

It turns out that if you have a supported GPU, it's still relatively straightforward for an attacker to force your browser to use SwiftShader for accelerated graphics - if the GPU process crashes more than 4 times, Chrome will fallback to this software rendering path instead of disabling acceleration. In my testing it's quite simple to cause the GPU process to crash or hit an out-of-memory condition from WebGL - this is left as an

- Crash the gpu process more than 4 times.
  - However, it didn't work…



```
for (let i = 0; i < 4; i++)
{
    gpuKiller(gl);
}
```

# Crash GPU Process 4 times

0: gpuKiller() -> Crashed

```
:ERROR:gpu_process_host.cc(967)] GPU process exited unexpectedly: exit_code=34
:WARNING:gpu_process_host.cc(1273)] The GPU process has crashed 1 time(s)
:INFO:CONSOLE(0)] "WebGL: CONTEXT_LOST_WEBGL: loseContext: context lost", source
:WARNING:gpu_process_host.cc(995)] Reinitialized the GPU process after a crash.
```

1: gpuKiller() -> Failed

2: gpuKiller() -> Failed

3: gpuKiller() -> Failed

```
canvas.addEventListener("webglcontextcreationerror",
(event) => {console.log(`${event.statusMessage}`);});
```

```
js> Web page caused context loss and was
blocked
js> Failed to create a WebGL2 context.
```

# GPU Fallback

```
void GpuProcessHost::RecordProcessCrash() {
…
if (recent_crash_count_ >= GetFallbackCrashLimit() && !disable_crash_limit) {
   base::UmaHistogramEnumeration(kFallbackEventCause,
                                 GPUFallbackEventCauseType::kCrashLimit);
   GpuDataManagerImpl::GetInstance()->FallBackToNextGpuMode();
 }
```

https://source.chromium.org/chromium/chromium/src/+/c7c5bedfc6c313826cb8cfa884dc3a3b20831311:content/browser/gpu/gpu_process_host.cc;l=1452;bpv=0;bpt=0

- The GetFallbackCrashLimit() function returns constant 3 on windows.
  - If gpu process crashes more than 3 times, it tries to fallback to the next GPU mode.
  - After gpu mode switched, recent_crash_count_ will be reset.
- There are 2 gpu fallbacks on windows.
  - gpu::GpuMode::SWIFTSHADER
  - gpu::GpuMode::DISPLAY_COMPOSITOR

# IsWebGLBlocked

```
js> Web page caused context loss and was blocked
js> Failed to create a WebGL2 context.
```

```
WebGLRenderingContextBase::CreateWebGraphicsContext3DProvider(
    …
    if (!host->IsWebGLBlocked())
       return provider;
     host->SetContextCreationWasBlocked();
     host->HostDispatchEvent(WebGLContextEvent::Create(
         event_type_names::kWebglcontextcreationerror,
         "Web page caused context loss and was blocked"));
     return nullptr;
    }
```

https://source.chromium.org/chromium/chromium/src/+/main:third_party/blink/renderer/modules/webgl/webgl_rendering_context_base.cc;l=688;drc=79fd5d71c46d0e6ecd842867bc1c787fae68e218;bpv=1;bpt=1

```
bool HTMLCanvasElement::IsWebGLBlocked() const {
 bool blocked = false;
 …
gpu_data_manager
     ->Are3DAPIsBlockedForUrl(document.Url(),
&blocked);
 return blocked;
}
```

https://source.chromium.org/chromium/chromium/src/+/main:third_party/blink/renderer/core/html/canvas/html_canvas_element.cc;l=579;drc=79fd5d71c46d0e6ecd842867bc1c787fae68e218;bpv=1;bpt=1?q=iswebglblock&ss=chromium%2Fchromium%2Fsrc

# Are3DAPIsBlockedAtTime

```
GpuDataManagerImplPrivate::Are3DAPIsBlockedAtTime
…
std::string domain = GetDomainFromURL(url);
 size_t losses_for_domain = base::ranges::count(
     blocked_domains_, domain,
     [](const auto& entry) { return entry.second.domain; });
 if (losses_for_domain > 1)
   return DomainBlockStatus::kBlocked;
…
```

https://source.chromium.org/chromium/chromium/src/+/main:content/browser/gpu/gpu_data_manager_impl_private.cc;l=1608;drc=b5b5329172a1607685db895653aa928560848ed3

```
GpuHostImpl::DidLoseContext
```

```
void GpuDataManagerImplPrivate::BlockDomainsFrom3DAPIsAtTime(
…
for (const auto& domain : domains) {
   blocked_domains_.insert({at_time, {domain, guilt}});
 }
```

https://source.chromium.org/chromium/chromium/src/+/main:content/browser/gpu/gpu_data_manager_impl_private.cc;l=1567;drc=b5b5329172a1607685db895653aa928560848ed3;bpv=1;bpt=1

```
var gl = [redacted].getContext('webgl2');
```

# 3 Domains to Enable Swiftshader

- Attacker need more than 3 different domains to deploy exploits.
  - Noisy and not OPSEC.
  - Inconvenient in the restricted scenarios.

# Patch the Renderer

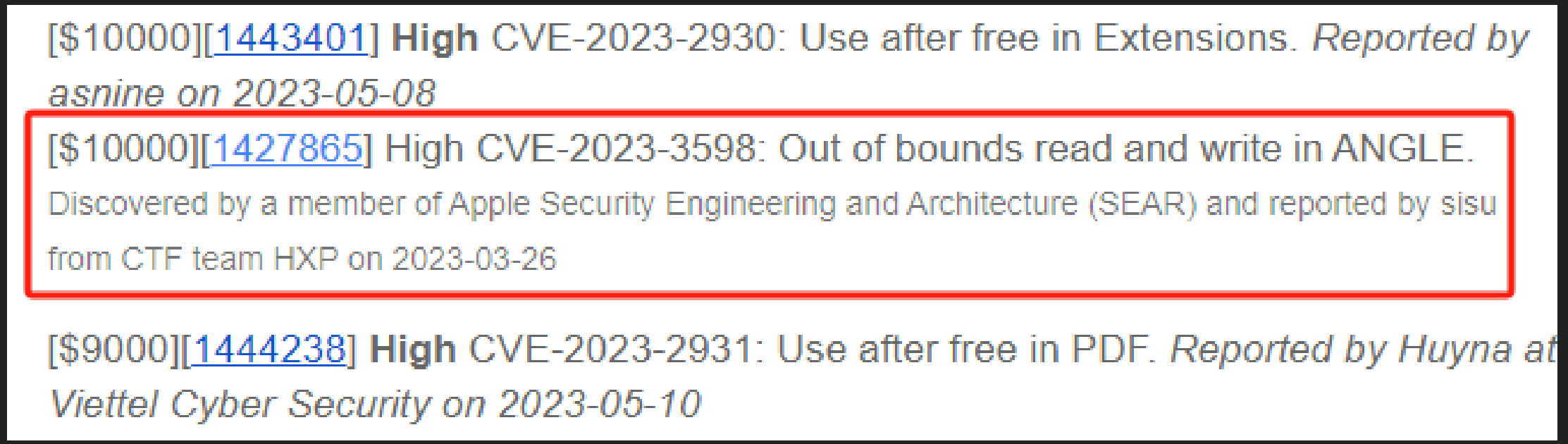
```
WebGLRenderingContextBase::CreateWebGraphicsContext3DProvider(
…
if (!host->IsWebGLBlocked())
   return provider;
 host->SetContextCreationWasBlocked();
 host->HostDispatchEvent(WebGLContextEvent::Create(
     event_type_names::kWebglcontextcreationerror,
     "Web page caused context loss and was blocked"));
 return nullptr;
}
```

➢ IsWebGLBlocked() simply returns true or false.
➢ No side effect in browser process.

=====> 1 assembly instruction patch to renderer process to bypass Webgl block checking.

# CVE-2023-3598: Out of Bounds Read and Write in ANGLE

- CVE-2023-3598 was discovered and exploited in hxpCTF
- Organizers caught the traffic and reported it to vender



# Analysis of CVE-2023-3598

- Vulnerability
  - Implement of merging allocas in SUBZERO JIT
    - Alloca
      - An instruction to allocate memory on stack.
      - Storing local variables and temporary data.
    - Integer overflow in Cfg::sortAndCombineAllocas.
      - Causes arbitrary code execution in GPU process.

- More details
  - https://issues.chromium.org/issues/40065276

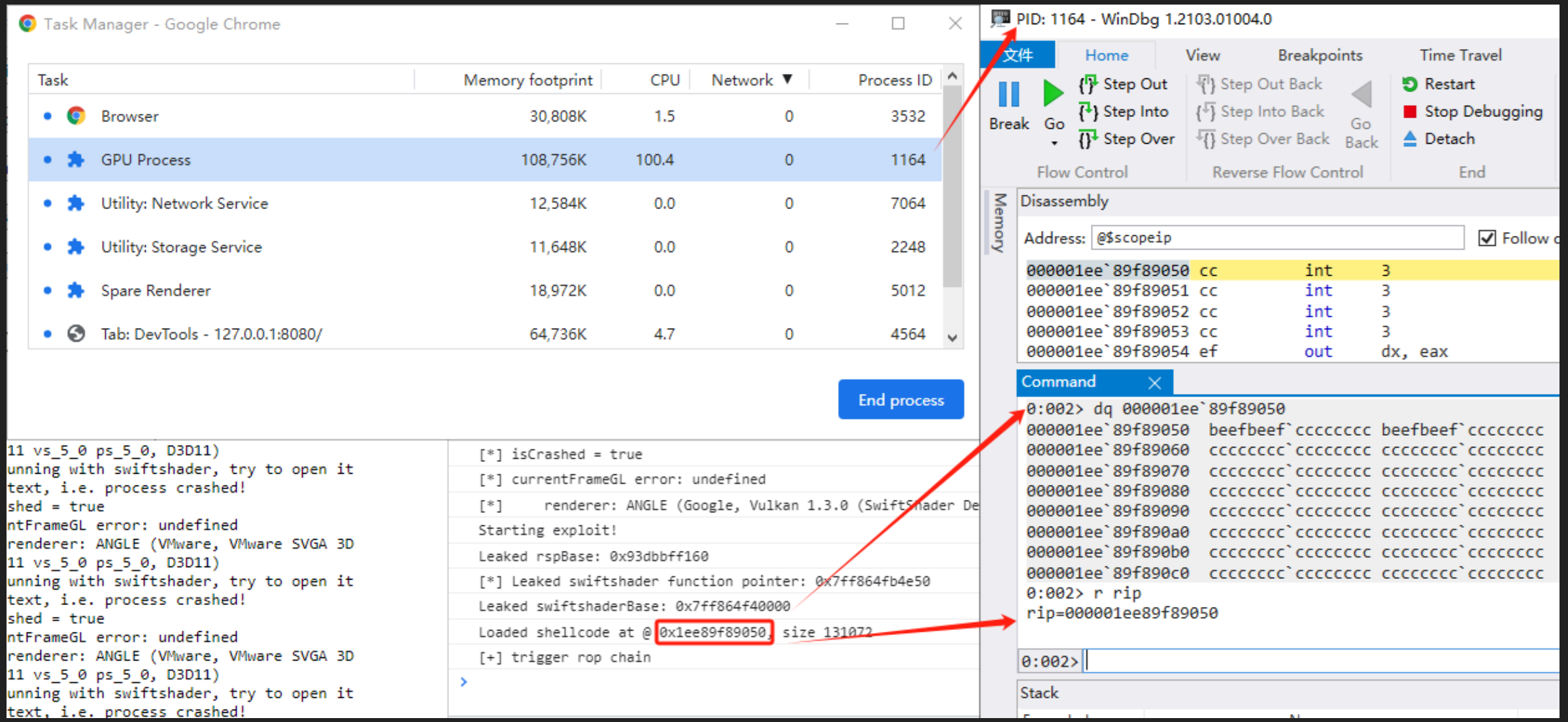
# Exploitation of CVE-2023-3598 on Windows

- Exploit on macOS does not work on windows.
- But primitives work :)
  - bufAccess
    - oobAccess
      - oobRead
      - oobWrite
- Extremely reduced development time.

# Steps of Exploitation

1. Leak chrome.dll and vk_swiftshader.dll and setup ROP gadgets.
   a. Based on oobRead primitive.
2. Place shellcode at a known address in the GPU process memory.
   a. Create Uniform Buffer Objects(UBOs).
   b. Setup data with shellcode.
   c. Leak heap address based on oobRead.
3. Use oobWrite to execute the ROP chain.
   a. Use vk_swiftshader!rr::protectMemoryPages to modify shellcode permissions to PAGE_EXECUTE_READWRITE.
   b. Jump to shellcode.

# Exploitation of CVE-2023-3598 on Windows



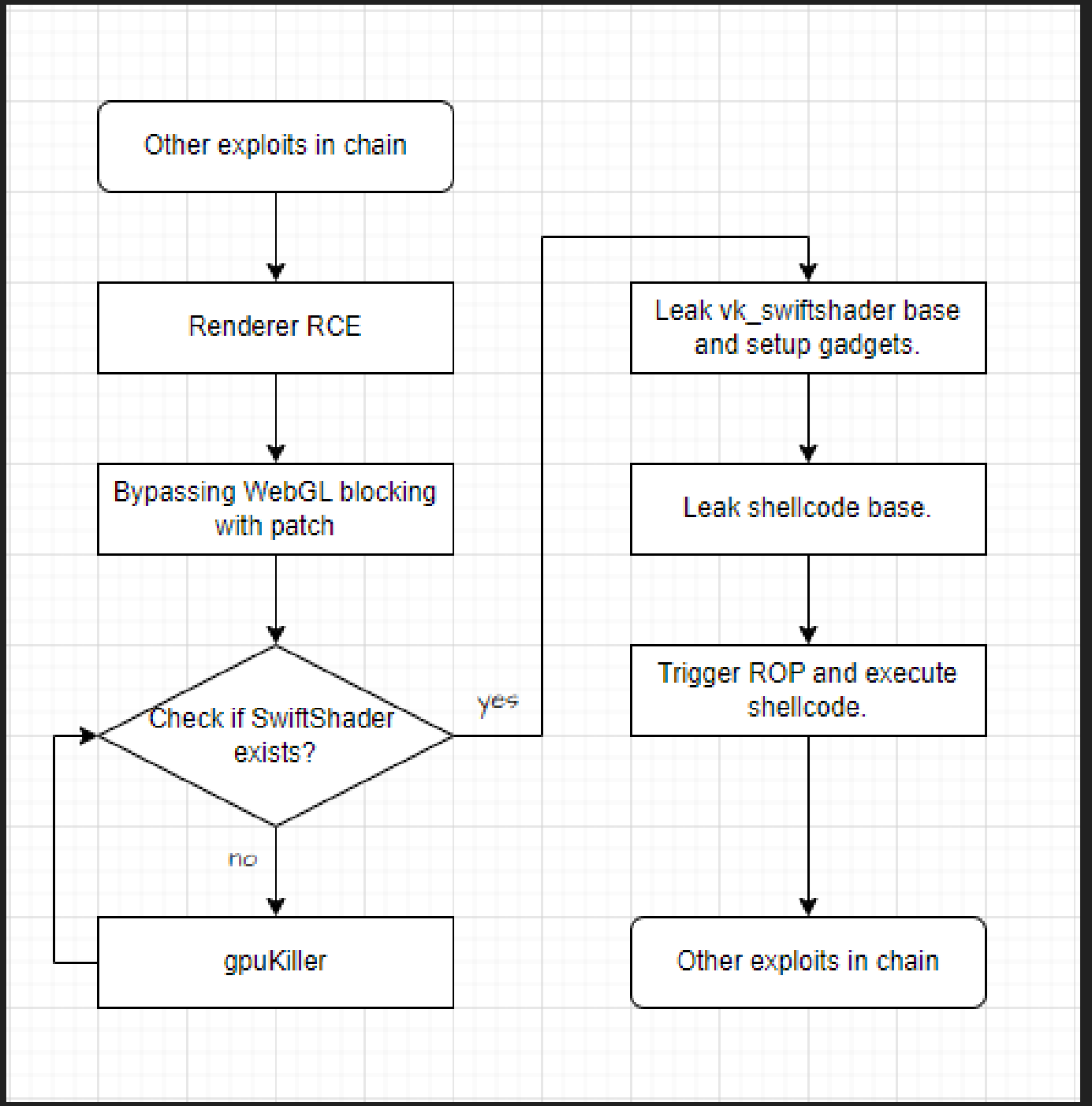
# Review Full Exploit Chain

1. Get renderer RCE.
   a. Bypass WebGL blocking

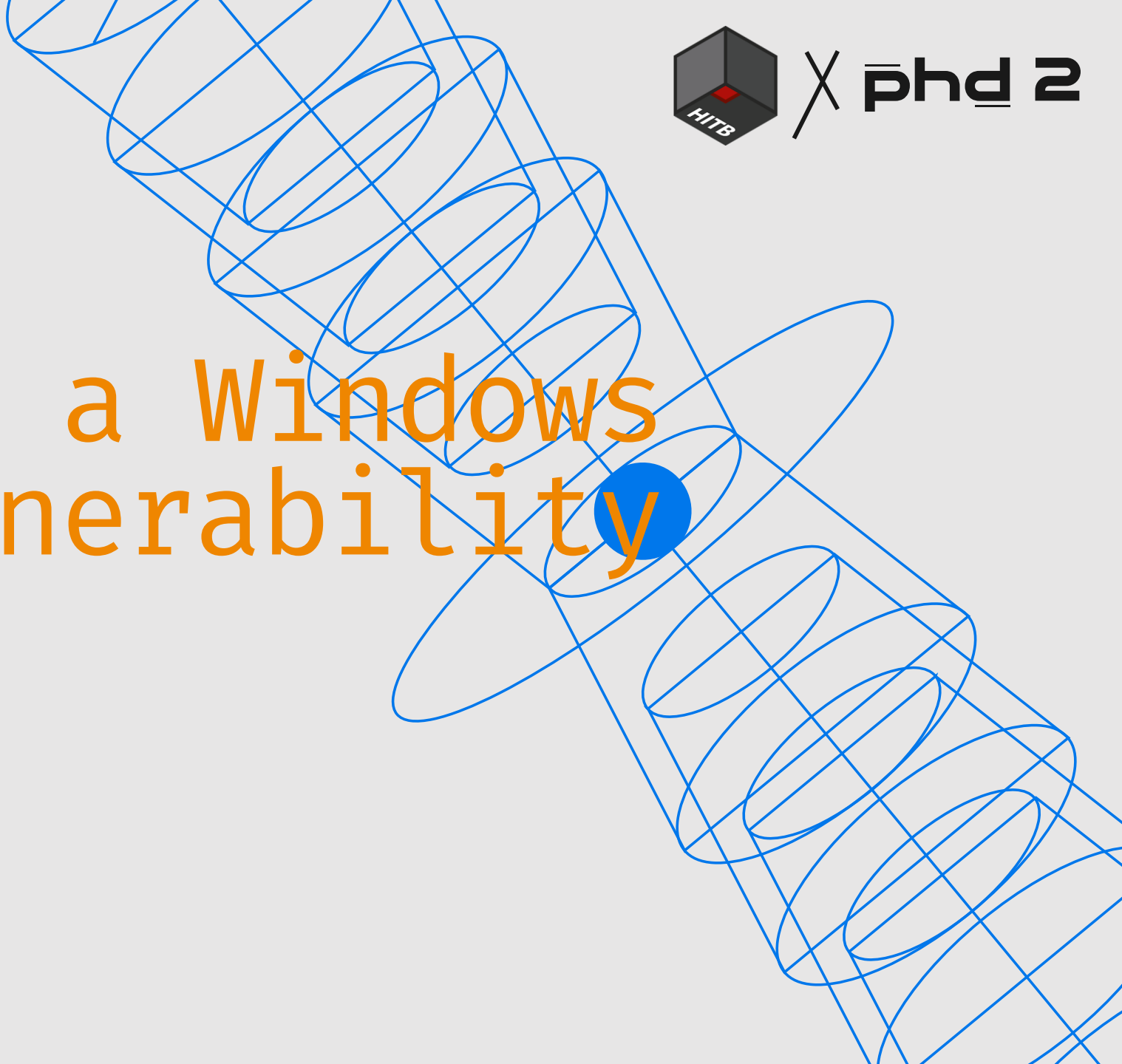
2. Using gpuKiller to enable the swiftshader backend.

3. Leaking addresses and building gadgets.

4. Trigger ROP to execute next stage shellcode.



# 04



# Exploiting a Windows Kernel Vulnerability

# Escape Chrome GPU/Adobe Sandboxed Process

Find a target -> Afd.sys

- ➢ Several bugs in the past few years.
- ➢ Exploited in the Pwn2Own 2014.
- ➢ Can be accessed from Chrome GPU/Adobe Sandboxed Process.

Case study -> CVE-2023-35632 (fixed in 2023.12)

# Pseudo Code Snippet

```
DWORD AlignedBufSize = (SendBufSize + 7) & 0xFFFFFFF8;
if (AlignedBufSize > 0x8000){
    LocalBuf = AfdGetBufferSlow(AlignedBufSize, …);
}else{
    LocalBuf = AfdGetBufferFast(AlignedBufSize, …);
}
/*
    ...
*/
try{
    if (ExGetPreviousMode() != KernelMode)
        ProbeForRead( UserBuf, SendBufSize, sizeof(UCHAR));

    memmove(LocalBuf, UserBuf, SendBufSize);
} except (EXCEPTION_EXECUTE_HANDLER) {
    return GetExceptionCode();
}
```

always hit here

if SendBufSize == 0xfffffff9, AlignedBufSize will be 0

# Limitations

Issues we meet:

1. The OOBW size (0xfffffff9) is too large for exploiting.

2. Vuln pool is got from AFD Lookaside List but not directly from system.

# Bad News

The OOBW size is too large for exploiting

```
try{
    if (ExGetPreviousMode() != KernelMode)
        ProbeForRead( UserBuf, 0xfffffff9, sizeof(UCHAR));

    memmove(LocalBuf, UserBuf, 0xfffffff9)
} except (EXCEPTION_EXECUTE_HANDLER) {
    return GetExceptionCode();
}
```

point to user mode

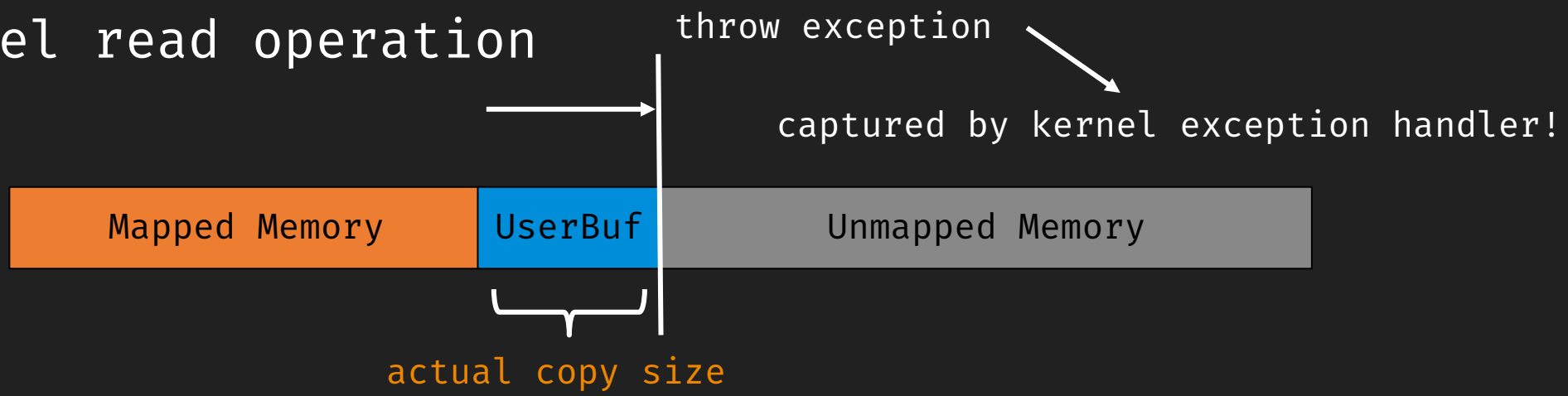
Can we abuse this?

# Make OOBW size controllable

- VirtualAllocEx -> Mapped Memory


- Put UserBuf in the tail


- Break kernel read operation



# Bad News

Vuln pool is got from AFD Lookaside List but not directly from system

```
LocalBuf = AfdGetBufferFast(AlignedBufSize, …);
```

```
AfdGetBufferFast(
    DWORD BufferDataSize,
    ...
){
    if ( BufferDataSize <= AfdSmallBufferSize ) {
        lookasideList = &AfdLookasideLists->SmallBufferList;
    } else if ( BufferDataSize <= AfdMediumBufferSize ) {
        lookasideList = &AfdLookasideLists->MediumBufferList;
    } else {
        lookasideList = &AfdLookasideLists->LargeBufferList;
    }

    buffer = ExAllocateFromNPagedLookasideList( lookasideList );
    return buffer
}
```

always hit here

# What is Lookaside List?

Lookaside lists are single linked lists containing pool allocations of a fixed size. They are used by drivers for caching memory allocations instead of always requesting them from the memory manager.

https://windows-internals.com/lookaside-list-forensics/

Allow drivers to manage the ‘freed’ pool, speed up the process of pool allocation.

```
ExAllocateFromNPagedLookasideList(
    PNPAGED_LOOKASIDE_LIST Lookaside
    ){

    PVOID Entry = InterlockedPopEntrySList(&Lookaside->L.ListHead);

    if (Entry == NULL) {

        Entry = (Lookaside->L.Allocate)(Lookaside->L.Type,
                                        Lookaside->L.Size, /* 0x2e0 */
                                        Lookaside->L.Tag);

    }

    return Entry;
}
```

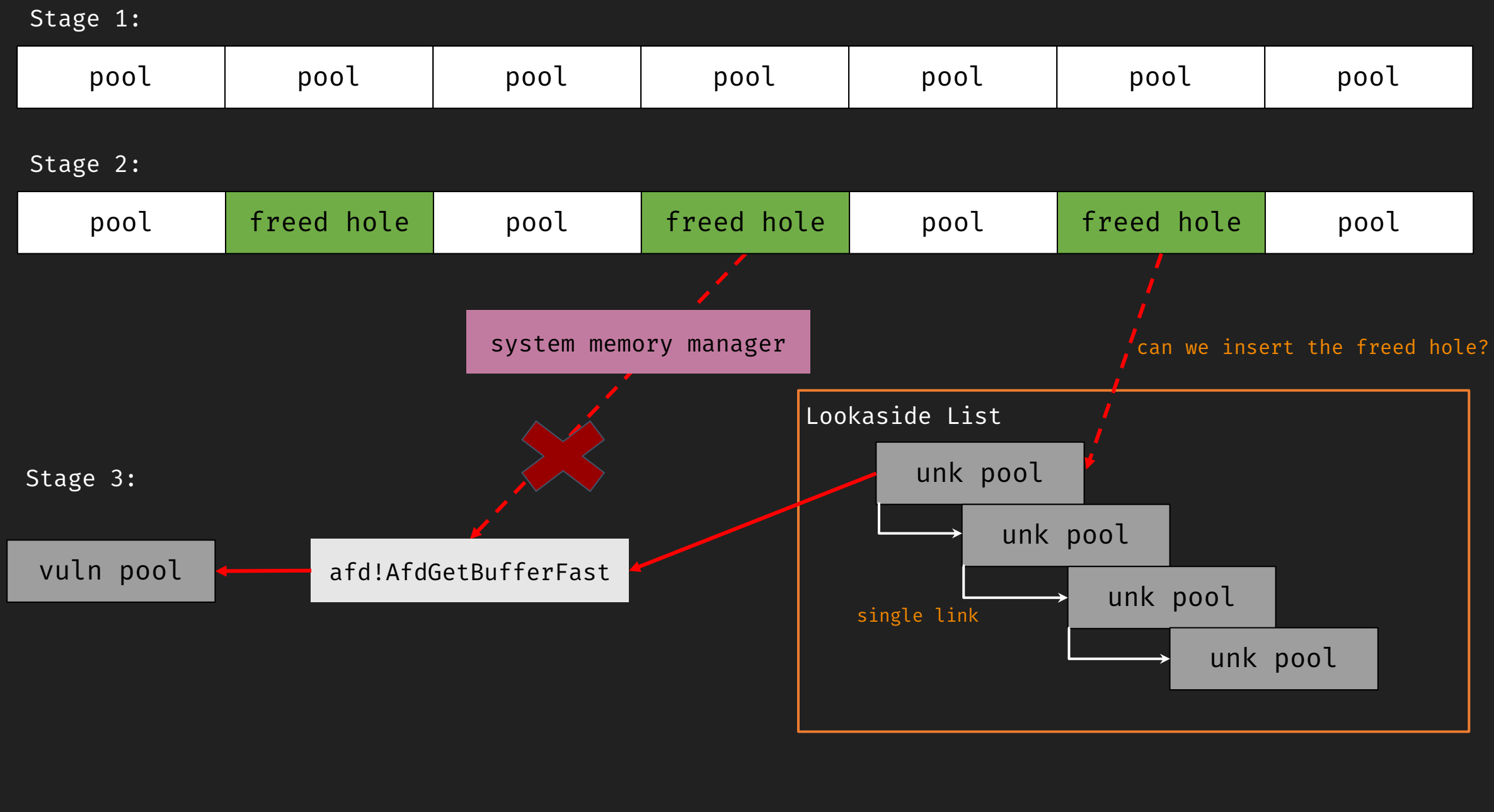
get from LookasideList as default

get from system if LookasideList is empty

```
AfdAllocateBuffer (
    IN POOL_TYPE PoolType,
    IN SIZE_T NumberOfBytes,
    IN ULONG Tag
    ){

    PVOID buffer = ExAllocatePoolWithTagPriority(PoolType,
NumberOfBytes, Tag, LowPoolPriority);
    /* Initialize Buffer */
}
```

NonPagedPool

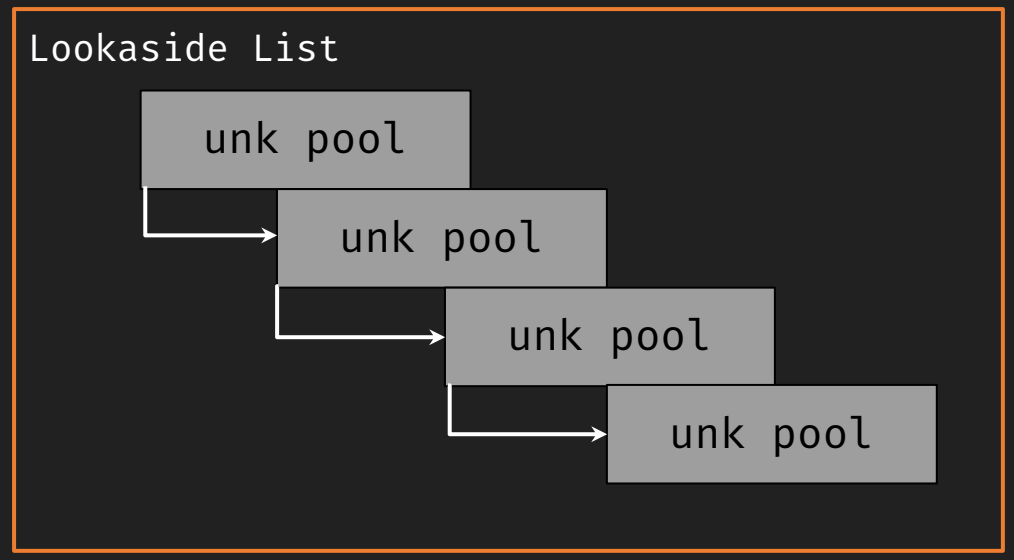
# Pool Fengshui



# Exhausting Cached Pool in Lookaside List

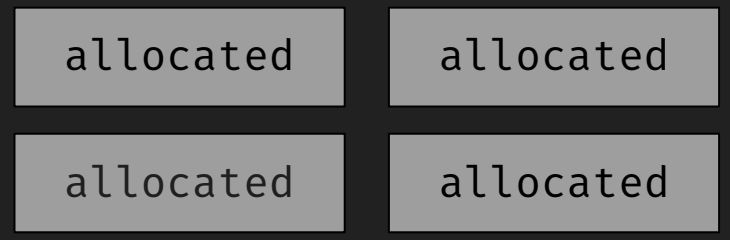
A new pool is created only when the Lookaside List is empty!

```
if (Entry == NULL) {
        Entry = (Lookaside->L.Allocate)(Lookaside->L.Type,
                                        Lookaside->L.Size,
                                        Lookaside->L.Tag);
    }
```

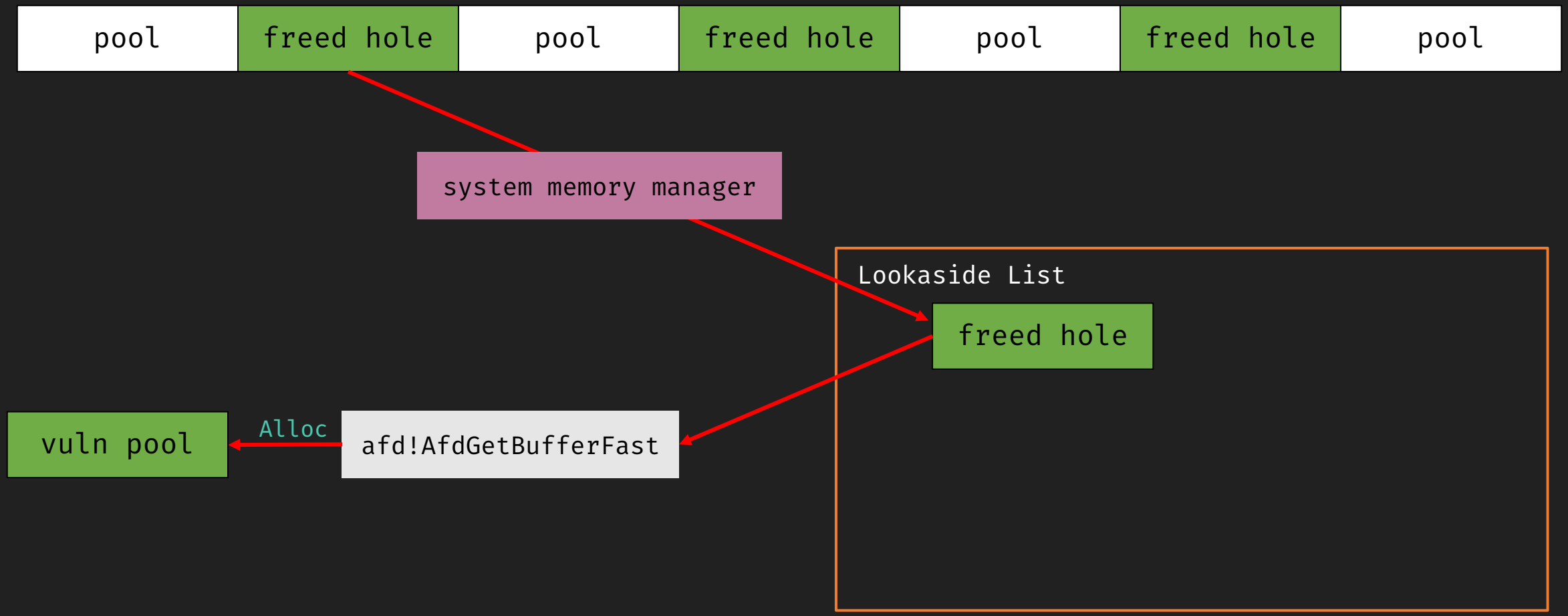




Many AFD functions can help us to do that since lookaside list is used everywhere



# Fill Lookaside List with Freed Hole



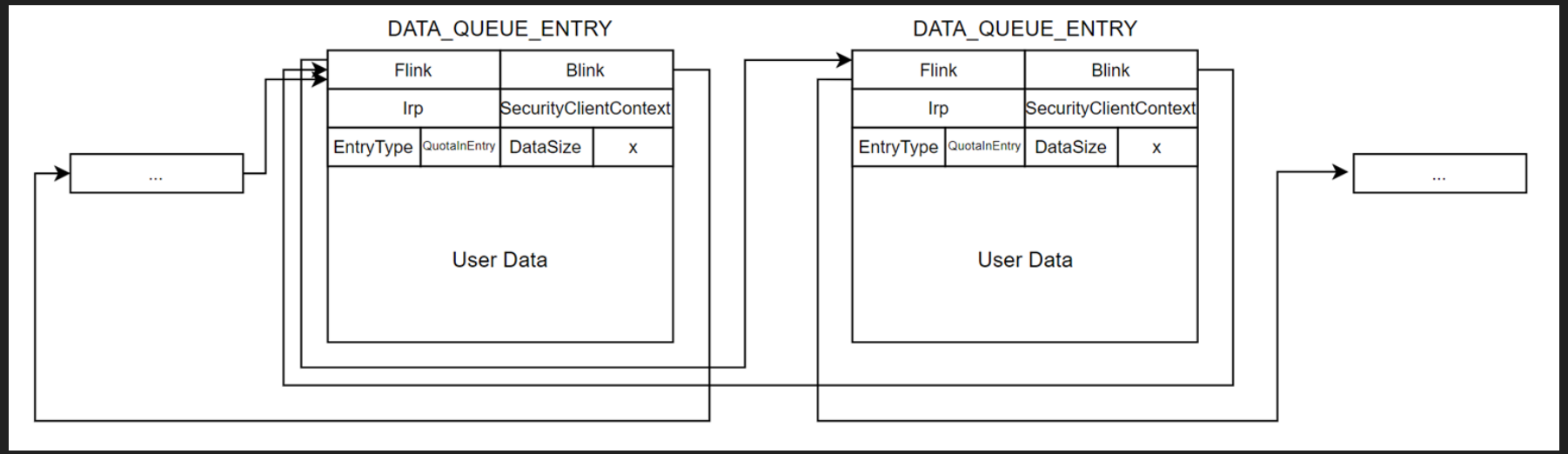
# What We Have Now:

- Vuln pool is got from system memory manager.
- Vuln pool size is 0x2e0.
- Vuln pool type is nonpaged pool.
- Out-of-Bound Writes arbitrary size.

# Spray Named Pipe

## To Get Arbitrary Read/Write primitive

```
struct DATA_QUEUE_ENTRY {
    LIST_ENTRY NextEntry;
    _IRP* Irp;
    _SECURITY_CLIENT_CONTEXT* SecurityContext;
    uint32_t EntryType;
    uint32_t QuotaInEntry;
    uint32_t DataSize;
    uint32_t x;
    char Data[];
}
```



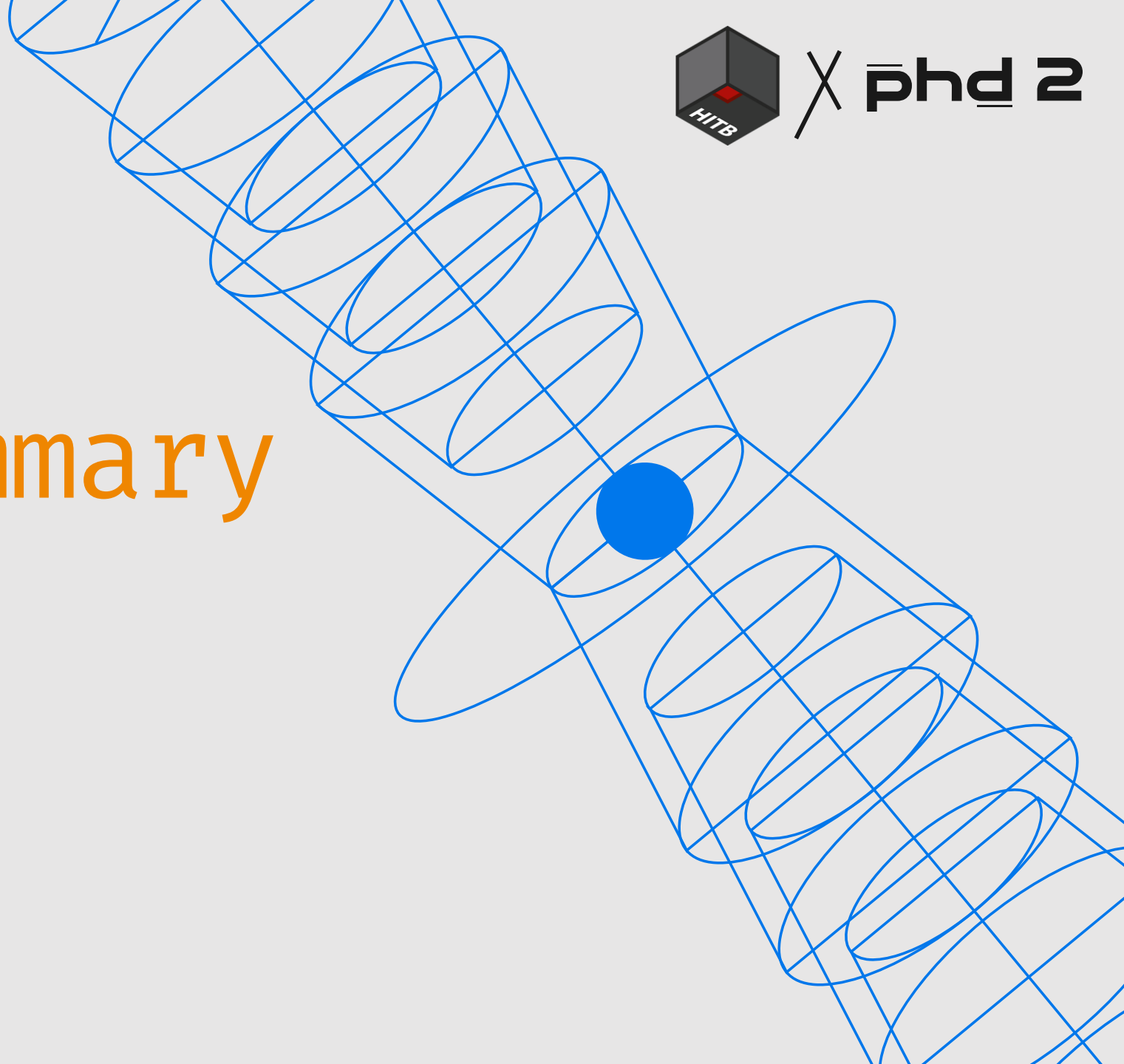
# Replace Our Token with SYSTEM Token

Abusing our Arbitrary Read/Write primitive:

- Parsing Leaked IRP:
  - Get sandbox process token kernel address
  - Get system process token kernel address
- Replace sandbox process token with system token
- Inject shellcode into winlogon.exe to spawn a new process
  (Directly spawn new process in sandbox is not allowed, because of the job limitation)

# Demo for Chrome

05



# Summary

# Review

- Sandbox Internal
- Sandbox Escape Methodology
- Attack Surface Comparison
  - Chrome Renderer
  - Chrome GPU Process
  - Adobe Sandboxed Process
- Build Fullchain 1-Day Exploit
  - Chrome Renderer
  - Adobe Sandboxed Process

# Takeaway

- Chrome Renderer SBX is Hard but Doable
  - GPU process is a good stairway: both in Android and Windows
- SBX for Chrome GPU and Adobe Sandboxed Process is Relatively Easier
  - More restricts need
- Exploitation Tricks
  - Chrome and Windows Kernel


ESC
HITB
Thank You!
ENERGY HACK

# Reference


[1] Chromium Docs. “Sandbox”. [Online]
https://chromium.googlesource.com/chromium/src/+/refs/heads/main/docs/design/sandbox.md
[2] Mariko Kosaka. “Inside look at modern web browser (part 1)”. [Online]
https://developer.chrome.com/blog/inside-browser-part1
[3] Chromium Code Search. [Online]
https://source.chromium.org/chromium/chromium/src/+/main:sandbox/win/src/security_level.h
[4] Google Project Zero. “0-days In-the-Wild”. [Online]
https://googleprojectzero.github.io/0days-in-the-wild/
[5] Wikipedia contributors. “WebGL”. [Online]
https://en.wikipedia.org/wiki/WebGL
[6] Intel Web Graphics Team. “WebGPU, The Next Generation Graphics API on the Web”. [Online]
https://www.youtube.com/watch?v=y2dZYG5YTRU
[7] Google Inc. SwiftShader. [Online]
https://swiftshader.googlesource.com/SwiftShader
[8] Apple Security Engineering and Architecture. “Security: Out-of-bounds reads/writes on stack in GPU process reachable from any WebGL shader”. [Online]
https://issues.chromium.org/issues/40065276
[9] Nicolas Capens. “SwiftShader Reference Implementation and Fallback”.[Online]
https://www.khronos.org/assets/uploads/developers/library/2018-vulkan-devday/08-SwiftShader.pdf
[10] Mark Brand. “Heap Feng Shader: Exploiting SwiftShader in Chrome”.[Online]
https://googleprojectzero.blogspot.com/2018/10/heap-feng-shader-exploiting-swiftshader.html

# Reference


[11] Google Chrome team. May 30, 2023. “Stable Channel Update for Desktop”. [Online]
https://chromereleases.googleblog.com/2023/05/stable-channel-update-for-desktop_30.html
[12] “[subzero] Fix integer overflows during alloca coalescing”. [Online]
https://source.chromium.org/chromium/_/swiftshader/SwiftShader/+/4e401427f8dd799b17ac6c805391e2da1e017672
[13] National Vulnerability Database. CVE-2023-3598. [Online]
https://nvd.nist.gov/vuln/detail/CVE-2023-3598
[14] Windows Ancillary Function Driver for WinSock Elevation of Privilege Vulnerability. [Online]
https://msrc.microsoft.com/update-guide/vulnerability/CVE-2023-35632
[15] “Move aside, signature scanning!” Better kernel data discovery through lookaside lists. [Online]
https://windows-internals.com/lookaside-list-forensics
[16] Windows Non Paged Pool Overflow Exploitation. [Online]
https://github.com/vp777/Windows-Non-Paged-Pool-Overflow-Exploitation
[17] sandbox-attacksurface-analysis-tools. [Online]
https://github.com/googleprojectzero/sandbox-attacksurface-analysis-tools