black hat USA 2020
AUGUST 5-6, 2020
BRIEFINGS

TiYunZong: An Exploit Chain to Remotely Root Modern Android Devices

Guang Gong, 360 Alpha Lab

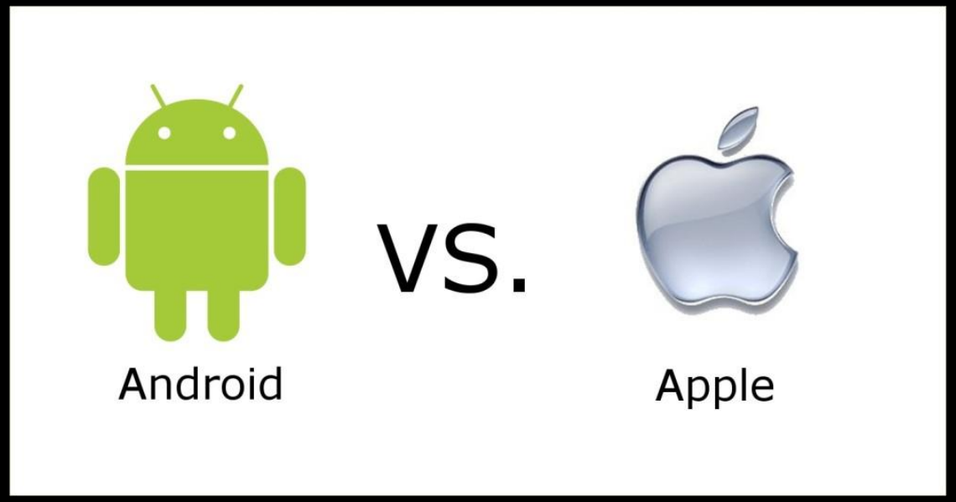
About 360 Alpha Lab

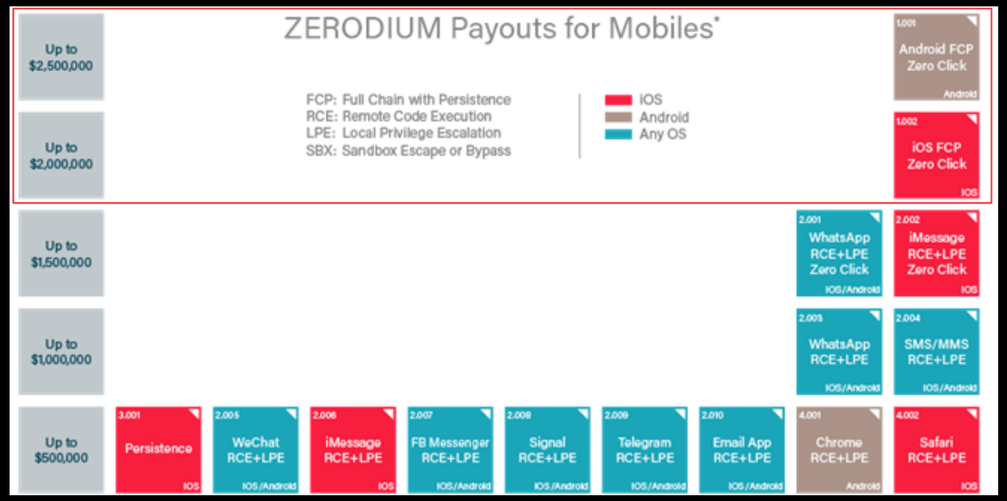
- Found 300+ Android vulnerabilities（Google Qualcomm etc）
- Won the highest reward in the history of the ASR program in 2017.
- Won the highest reward in the history of all Google VRP program in 2019
- 6 Pwn contest winners
  - Pwn2Own Mobile 2015(Nexus 6)
  - Pwn0Rama 2016 (Nexus 6p)
  - Pwn2Own 2016(Chrome)
  - PwnFest 2016(Pixel)
  - Pwn2Own Mobile 2017(Galaxy S8)
  - TianFuCup 2018(Chrome)

Agenda

- Why Google pixel phone is a tough target
- Remote attack surface of Android devices
- Experience of pwning Android devices
- Overview the exploit chain, TiYunZong
- Detail the three vulnerabilities in the chain
- Demonstrate remotely rooting pixel phone

Why Google Pixel Phone Is A Tough Target





Why Google Pixel Phone Is A Tough Target

Devices / Pwned Times / Year	Apple iPhone	Google Pixel	Samsung Galaxy	Huawei Mate/P	Xiaomi Mi
Mobile Pwn2Own 2017	5(1 partial win)	0	3	2	N/A
Mobile Pwn2Own 2018	2	0	2	0	5
Mobile Pwn2Own 2019	0	0	3(1 partial win)	0	3(1 partial win)
Total	7	0	8	2	8

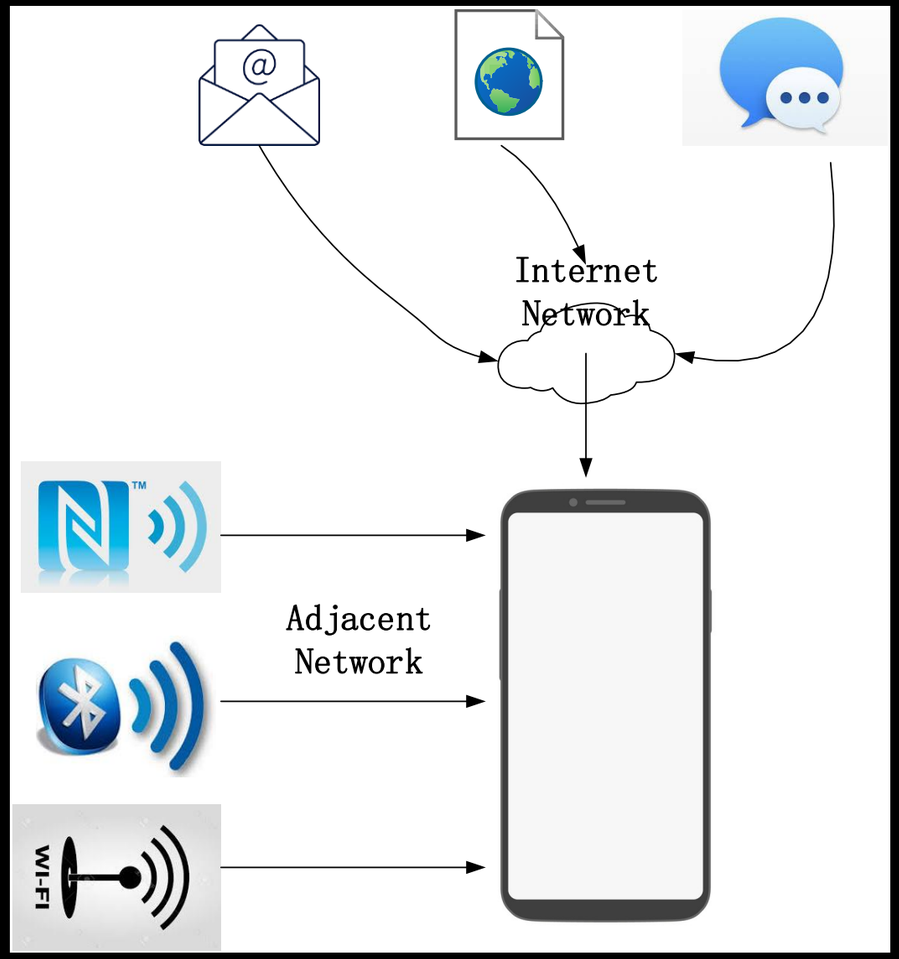
Mobile Pwn2Own results of the latest three years

Remote Attack Surface of Smart Phones

Attacks through internet

VS

Attacks through adjacent network

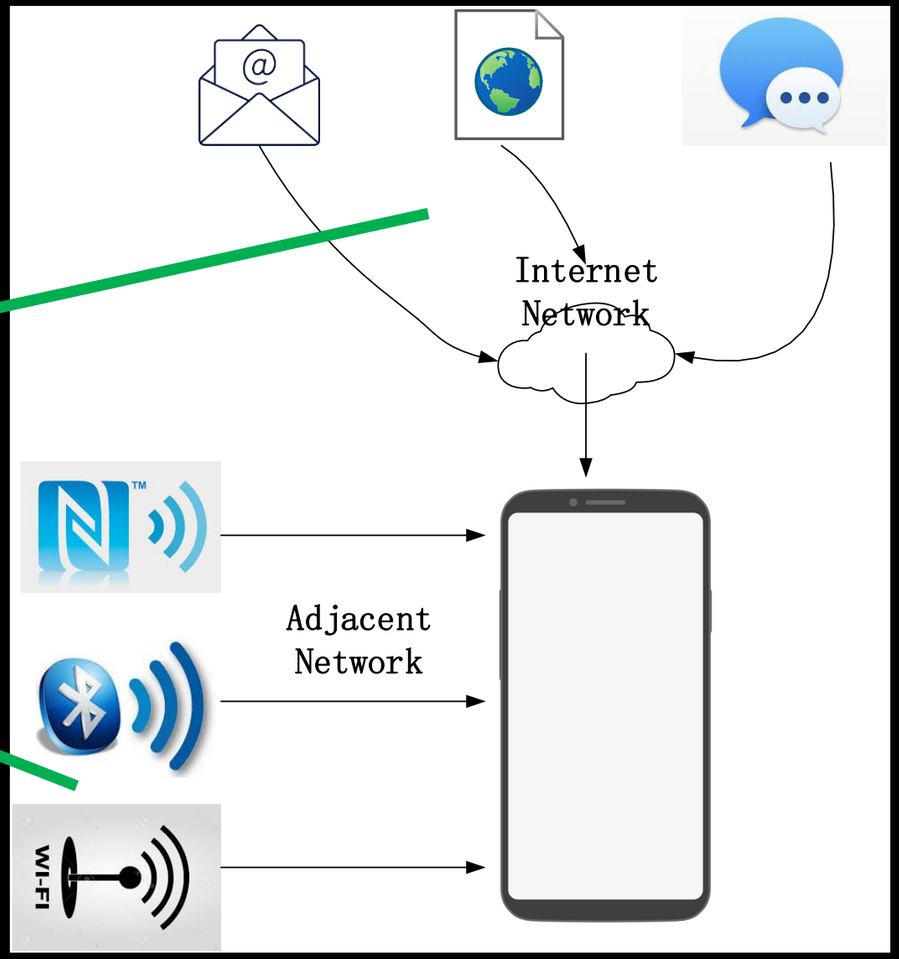


Remote Attack Surface of Smart Phones

One-click Internet Deployable Attack

VS

Zero-click Adjacent Network Attack



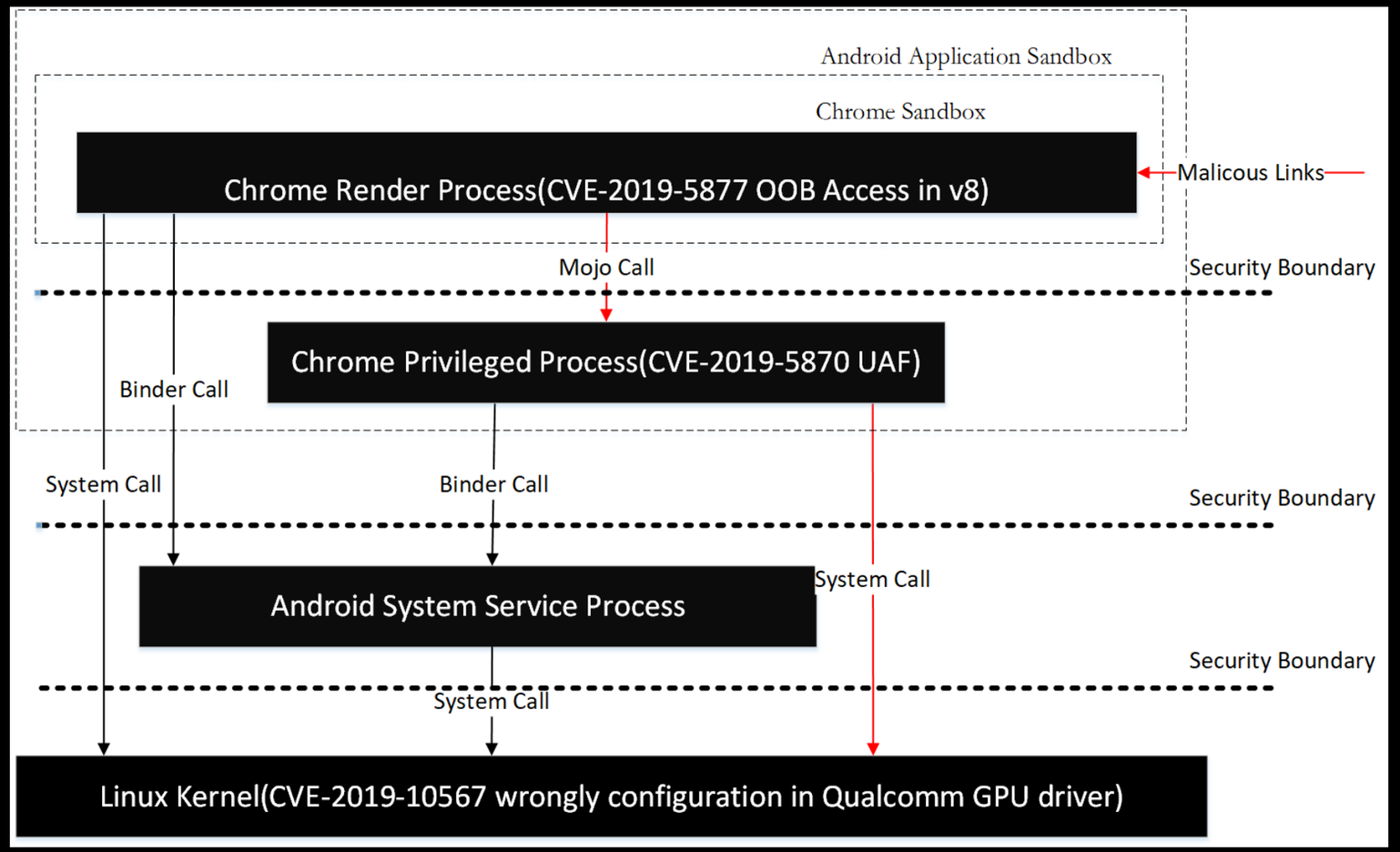
Experience of Pwning Android Devices

Event	Attack Vector	Target Phone	Obtained Permissions
▪ Pwn2Own 2015	Chrome v8 bug->RCE2UXSS->Google Play Install	Nexus 6	Install any app
▪ Pwn0Rama 2016	Chrome v8 bug ->Chrome IPC weakness->WebView bug	Nexus 6P, Galaxy Note 5, LG G4	Install any app
▪ PwnFest 2016	Chrome v8 bug ->RCE2UXSS->Google Play Install	Pixel XL	Install any app
▪ Pwn2Own 2017	Samsung Internet Browser bug -> exynos gralloc module bug	Galaxy S8	System user permission
▪ ASR 2018	Chrome v8 bug -> libgralloc module bug	Pixel	System user permission
▪ ASR 2019	Chrome v8 bug ->Mojo IPC bug->KGSL bug	Pixel 3	Root user permission

Remote working exploit chains targeting Android Found by me in recent years

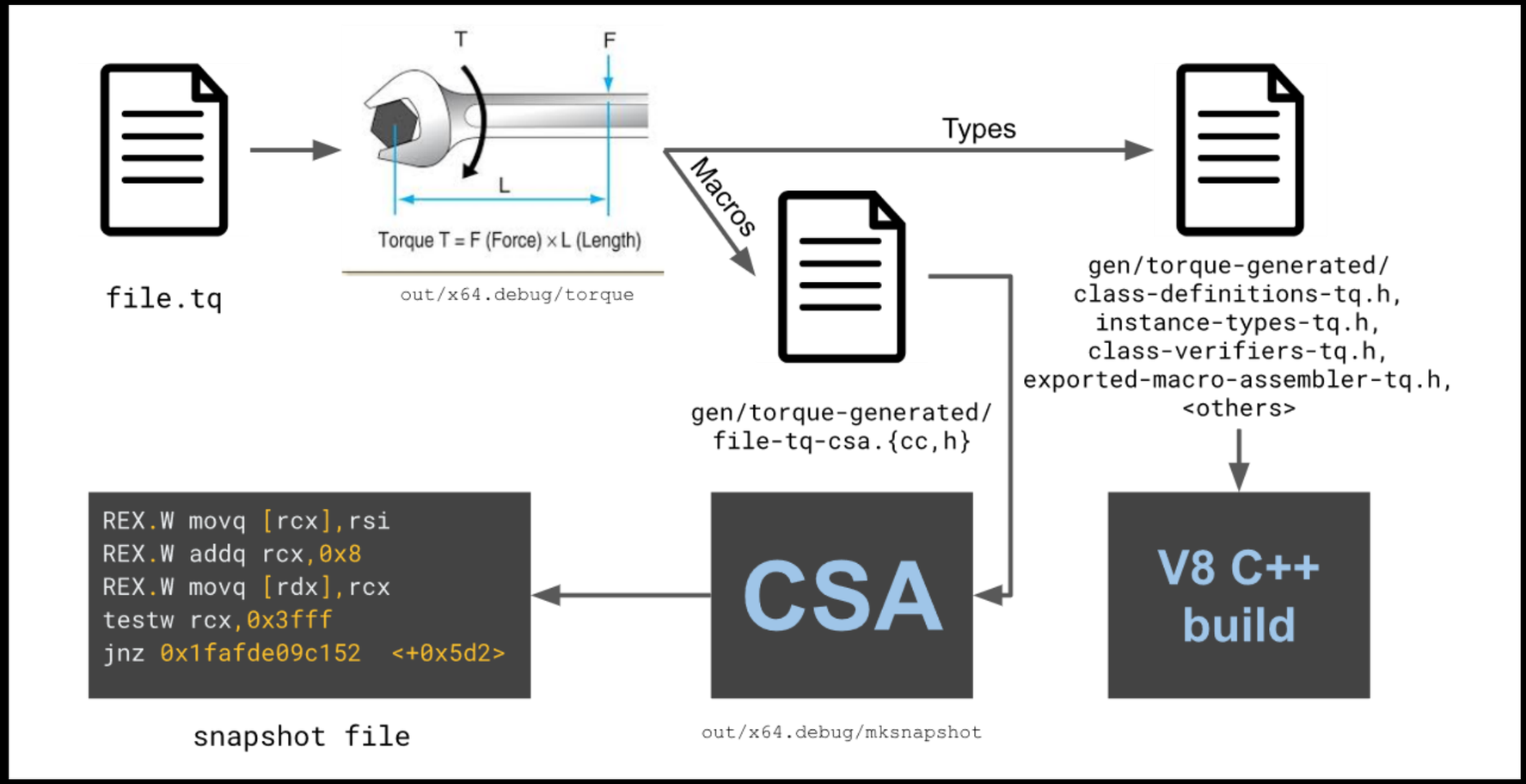
The Exploit Chain(TiYunZong)

The Exploit Chain(TiYunZong)



The RCE Vulnerability
CVE-2019-5877

Torque in Chrome v8



JSFunction Memory Layout

- JSFunction in Chrome v8 is the internal representation of function in JavaScript.
- The size of JSFunction object in v8 is not fixed. it may contain the field PrototypeOrInitialMap or not.
- The field PrototypeOrInitialMap is the last field of JSFunction if it has.

JSFunction Memory Layout

```
d8> %DebugPrint(Array);
DebugPrint: 0x2b7bcbd91b79: [Function] in OldSpace
 - map: 0x2f2f62402ff1 <Map(HOLEY_ELEMENTS)> [FastProperties]
 - prototype: 0x2b7bcbd82091 <JSFunction (sfi = 0xc83e79480e9)>
 - elements: 0x3b058dc40bf9 <FixedArray[0]> [HOLEY_ELEMENTS]
 - function prototype: 0x2b7bcbd91dc9 <JSArray[0]>
 - initial_map: 0x2f2f62403041 <Map(PACKED_SMI_ELEMENTS)>
 - shared_info: 0x0c83e79547c9 <SharedFunctionInfo Array>


0x2f2f62402ff1: [Map]
 - type: JS_FUNCTION_TYPE
 - instance size: 64
  - callable
 - constructor
 - has_prototype_slot
 - constructor: 0x2b7bcbd822e1 <JSFunction Function (sfi = 0xc83e7954449)>
```

JSFunction Memory Layout

```
d8> %DebugPrint(parseInt)
DebugPrint: 0x2b7bcbd8b999: [Function] in OldSpace
 - map: 0x2f2f624003e1 <Map(HOLEY_ELEMENTS)> [FastProperties]
 - prototype: 0x2b7bcbd82091 <JSFunction (sfi = 0xc83e79480e9)>
 - elements: 0x3b058dc40bf9 <FixedArray[0]> [HOLEY_ELEMENTS]
 - function prototype: <no-prototype-slot>
 - shared_info: 0x0c83e79557a1 <SharedFunctionInfo parseInt>


0x2f2f624003e1: [Map]
 - type: JS_FUNCTION_TYPE
 - instance size: 56
 - callable
 - constructor: 0x3b058dc401b1 <null>
```

JSFunction Memory Layout

```
d8> %DebugPrint(Proxy)
DebugPrint: 0x2b7bcbd8d6d1: [Function] in OldSpace
 - map: 0x2f2f62401d31 <Map(HOLEY_ELEMENTS)> [FastProperties]
 - prototype: 0x2b7bcbd82091 <JSFunction (sfi = 0xc83e79480e9)>
 - elements: 0x3b058dc40bf9 <FixedArray[0]> [HOLEY_ELEMENTS]
 - function prototype: <no-prototype-slot>
 - shared_info: 0x0c83e795e749 <SharedFunctionInfo Proxy>


0x2f2f62401d31: [Map]
 - type: JS_FUNCTION_TYPE
 - instance size: 56
 - callable
 - constructor
 - constructor: 0x3b058dc401b1 <null>
```

The Bug(CVE-2019-5877)

```
macro GetDerivedMap(implicit context: Context)(
    target: JSFunction, newTarget: JSReceiver): Map {
  try {
    const constructor = Cast<JSFunction>(newTarget) otherwise SlowPath;
    const map =
        Cast<Map>(constructor.prototype_or_initial_map) otherwise SlowPath; *** oob access
    if (LoadConstructorOrBackPointer(map) != target) {
      goto SlowPath;
    }
    return map;
  }
  label SlowPath {
    return runtime::GetDerivedMap(context, target, newTarget);
  }
}
```

Trigger the Bug

- JavaScript

```
var malformedTypedArray = Reflect.construct(Uint8Array, [4], Proxy)
```

- Torque

```
transitioning builtin CreateTypedArray(
    context: Context, target: JSFunction, newTarget: JSReceiver, arg1: JSAny,
    arg2: JSAny, arg3: JSAny): JSTypedArray {
  assert(IsConstructor(target));

  const map = GetDerivedMap(target, newTarget);
  …
  //create a TypedArray with the returned map
}
```

How to Exploit

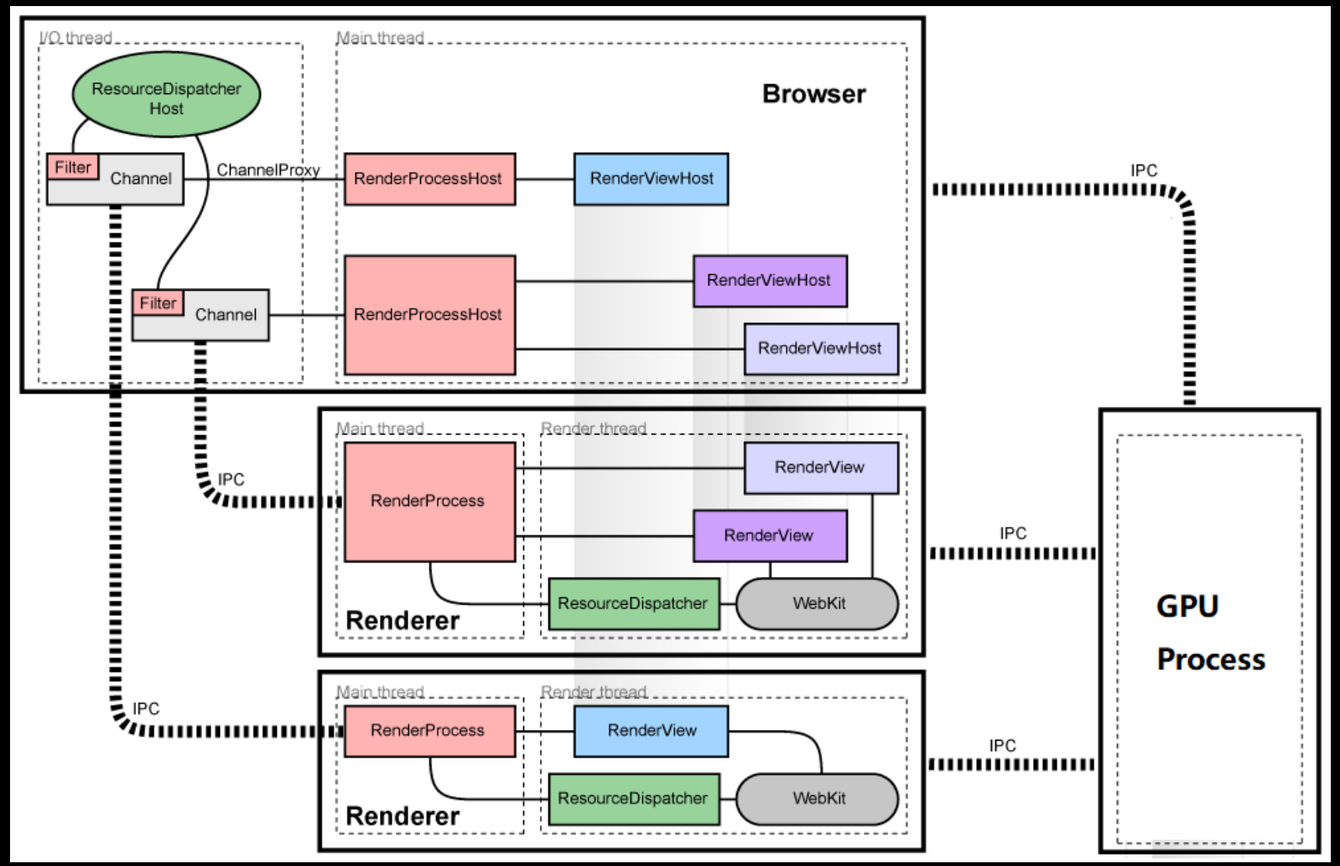
```
macro GetDerivedMap(implicit context: Context)(
    target: JSFunction, newTarget: JSReceiver): Map {
  try {
    const constructor = Cast<JSFunction>(newTarget) otherwise SlowPath;
    const map =
        Cast<Map>(constructor.prototype_or_initial_map) otherwise SlowPath; *** oob access
    if (LoadConstructorOrBackPointer(map) != target) {
      goto SlowPath;
    }
    return map;*** SlowPath will make OOB access useless, The execution flow must go to here.
  }
  label SlowPath {
    return runtime::GetDerivedMap(context, target, newTarget);
  }
}
```

Exploit Strategy

1. Free the object below the Proxy function.
2. Re-occupy the free space with an object whose Map's (named map x) constructor is Uint8Array, and then drop all reference to Map x so that GC will mark the Map x as white and will sweep it in scheduled sweep tasks.
3. Trigger the OOB access bug before Map x get swept by GC, so the vulnerable GetDerivedMap macro won’ t bail out to the slow path. The pointer of Map x will still be used in CreateTypedArray.
4. After sweep task finished, re-occupy the freed space of Map x with a map whose constructor is Uint32Array, so we can get a malformed typed array, its map is Uint32Array, but its layout, especially its element kind is Uin8Array type, it's easy to implement arbitrary read and write with this malformed object.
5. With the ability of arbitrary read and write, we can enable MojoJS bindings and exploit the following Mojo vulnerability to escalate from Chrome sandbox.

The EoP Vulnerability
CVE-2019-5870

Chrome’s Multi-Process Architecture



The Mojo Interface Definition of Content Decryption Module (CDM)

```
1 interface ContentDecryptionModule {
2   SetClient(pending_associated_remote<ContentDecryptionModuleClient> client);
3   Initialize(string key_system,
4              url.mojom.Origin security_origin,
5              CdmConfig cdm_config)
6       => (CdmPromiseResult result, int32 cdm_id,
7           pending_remote<Decryptor>? decryptor);
8   SetServerCertificate(array<uint8> certificate_data)
9       => (CdmPromiseResult result);
10  ......
11 };
```

The Implementation of the Initialized Function of CDM

```
1 void MojoCdmService::Initialize(const std::string& key_system,
2                                 const url::Origin& security_origin,
3                                 const CdmConfig& cdm_config,
4                                 InitializeCallback callback) {
5   DVLOG(1) << __func__ << ": " << key_system;
6   DCHECK(!cdm_);  ------------------>In debug version, this DCHECK will be trigger
7
8   auto weak_this = weak_factory_.GetWeakPtr();
9   cdm_factory_->Create(
10      key_system, security_origin, cdm_config,
11      base::Bind(&MojoCdmService::OnSessionMessage, weak_this),
12      base::Bind(&MojoCdmService::OnSessionClosed, weak_this),
13      base::Bind(&MojoCdmService::OnSessionKeysChange, weak_this),
14      base::Bind(&MojoCdmService::OnSessionExpirationUpdate, weak_this),
15      base::Bind(&MojoCdmService::OnCdmCreated, weak_this,
16                 base::Passed(&callback)));
17 }
```

The Callback Function OnCdmCreated

```
1 void MojoCdmService::OnCdmCreated(
2     InitializeCallback callback,
3     const scoped_refptr<::media::ContentDecryptionModule>& cdm,
4     const std::string& error_message) {
5   mojom::CdmPromiseResultPtr cdm_promise_result(mojom::CdmPromiseResult::New());
6
7   if (!cdm) {
8   ......
9   }
10    cdm_ = cdm;
11    if (context_) {
12      cdm_id_ = context_->RegisterCdm(this); ---------------------------->register twice here
13      DVLOG(1) << __func__ << ": CDM successfully registered with ID " << cdm_id_;
14    }
15    ...
16 }
```

The Fucntion RegisterCdm

```
1 int MojoCdmServiceContext::RegisterCdm(MojoCdmService* cdm_service) {
2   DCHECK(cdm_service);
3   int cdm_id = GetNextCdmId();
4   cdm_services_[cdm_id] = cdm_service; --->two cdm ids map to one cdm_service
5   DVLOG(1) << __func__ << ": CdmService registered with CDM ID " << cdm_id;
6   return cdm_id;
7 }
```

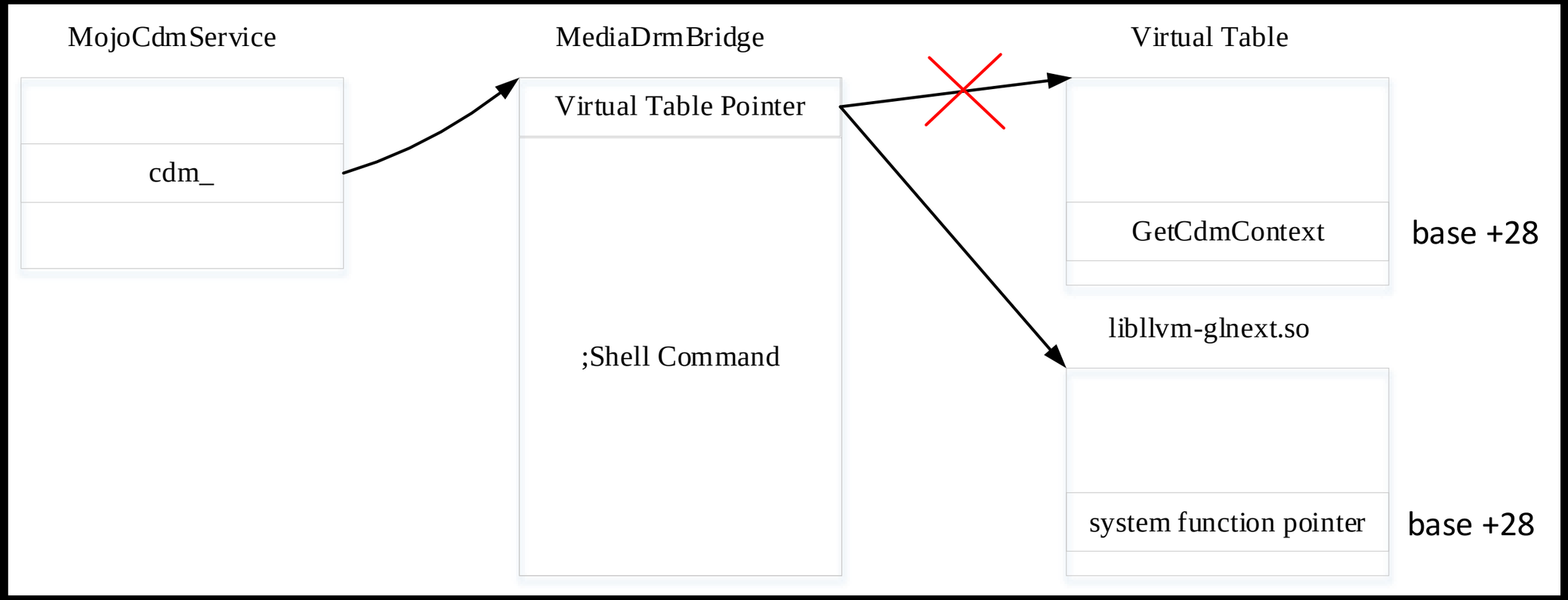
Trigger UAF

```
1 std::unique_ptr<CdmContextRef> MojoCdmServiceContext::GetCdmContextRef(
2     int cdm_id) {
3 ......
4   auto cdm_service = cdm_services_.find(cdm_id);
5   if (cdm_service != cdm_services_.end()) {
6     if (!cdm_service->second->GetCdm()->GetCdmContext()) {----------> UAF in GetCdm
7       NOTREACHED() << "All CDMs should support CdmContext.";
8       return nullptr;
9     }
10     return std::make_unique<CdmContextRefImpl>(cdm_service->second->GetCdm());
11   }
12   ......
13   return nullptr;
14 }
```

How To Exploit

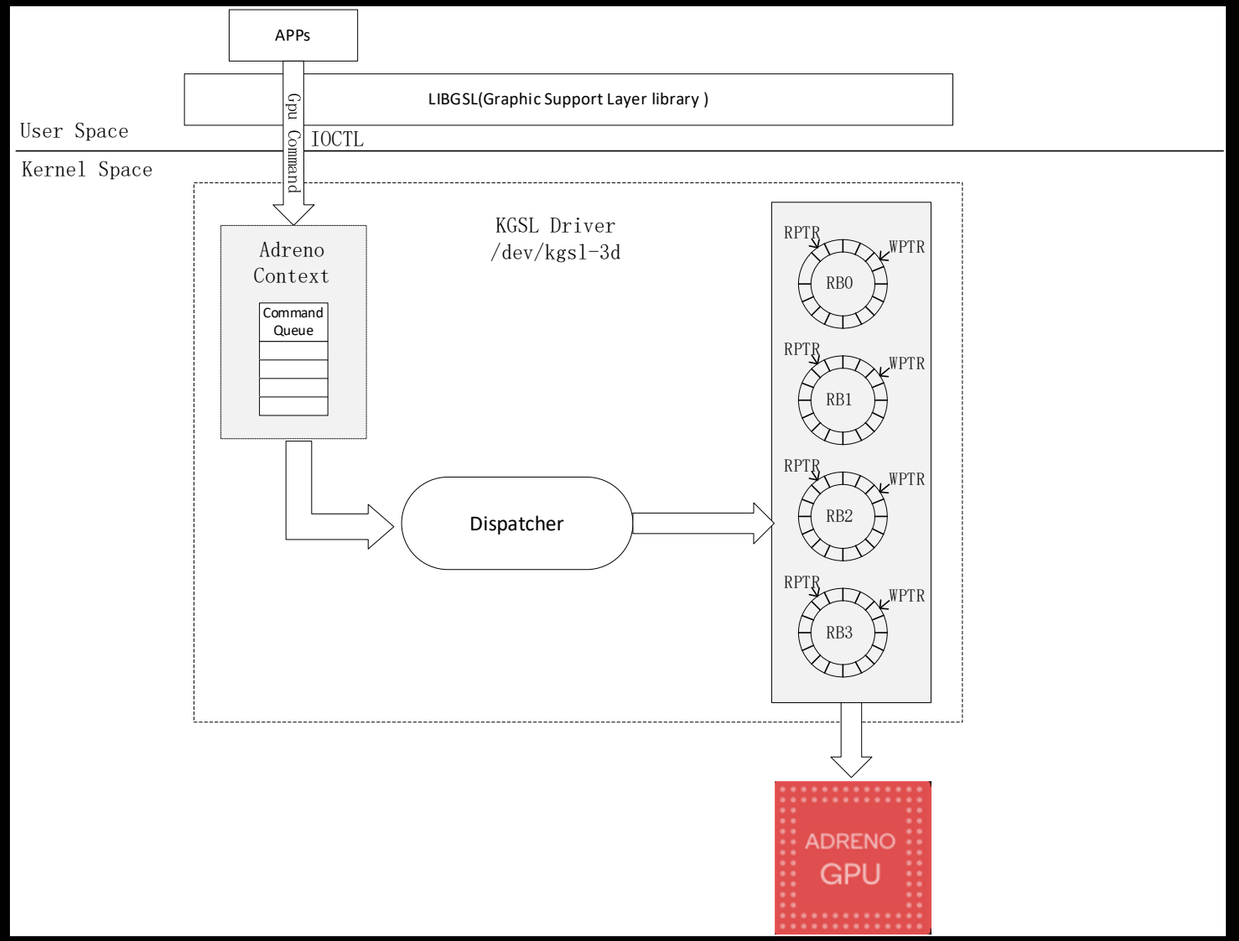
```
1 (gdb) p sizeof(media::MojoCdmService)
2 $21 = 48
3 (gdb) p sizeof(media::MediaDrmBridge)
4 $3 = 168 //the size is 160 in release version
5 (gdb) x/10xw 0xb6993300  //the content of MediaDrmBridge
6 0xb6993300:        0xca3f3a0c 0x00000000        0x00000100        0xca3f3a4c
7 0xb6993310:        0xca3f3a6c 0xb6a90750        0xb6a90750        0xb6a90760
8 0xb6993320:        0x00000000        0x00000000
9 (gdb) x/10xw 0xca3f3a0c  //the content of virtual table of MediaDrmBridge
10 0xca3f3a0c <_ZTVN5media14MediaDrmBridgeE+8>: 0xca237e09 0xca207a79 0xca237fad 0xca2382f9
11 0xca3f3a1c <_ZTVN5media14MediaDrmBridgeE+24>:        0xca2384a9 0xca238601 0xca238741 0xca238881
12 (gdb) info symbol 0xca238881
13 media::MediaDrmBridge::GetCdmContext() + 1 in section .text of libmedia.cr.so
```

Exploit the EoP Bug



The Root Vulnerability
CVE-2019-10567

The KGSL Driver Architecture of Adreno GPU



Globally Mapped Pages

```
1 crosshatch:/ # cat /sys/kernel/debug/kgsl/globals
2 0x00000000fc000000-0x00000000fc000fff     4096 setstate
3 0x00000000fc001000-0x00000000fc040fff   262144 gpu-qdss
4 0x00000000fc041000-0x00000000fc048fff    32768 memstore
5 0x00000000fc049000-0x00000000fc049fff     4096 scratch
6 0x00000000fc04a000-0x00000000fc04afff     4096 pagetable_desc
7 0x00000000fc04b000-0x00000000fc052fff    32768 ringbuffer
8 0x00000000fc053000-0x00000000fc053fff     4096 pagetable_desc
9 0x00000000fc054000-0x00000000fc05bfff    32768 ringbuffer
10 0x00000000fc05c000-0x00000000fc05cfff     4096 pagetable_desc
11 0x00000000fc05d000-0x00000000fc064fff     32768 ringbuffer
12 0x00000000fc065000-0x00000000fc065fff     4096 pagetable_desc
13 0x00000000fc066000-0x00000000fc06dfff     32768 ringbuffer
14 0x00000000fc06e000-0x00000000fc09dfff    196608 profile
15 0x00000000fc09e000-0x00000000fc0a5fff     32768 ucode
16 0x00000000fc0a6000-0x00000000fc0a8fff     12288 capturescript
17 0x00000000fc0a9000-0x00000000fc113fff    438272 capturescript_regs
18 0x00000000fc114000-0x00000000fc114fff     4096 powerup_register_list
19 0x00000000fc115000-0x00000000fc115fff     4096 alwayson
20 0x00000000fc116000-0x00000000fc116fff     4096 preemption_counters
21 0x00000000fc117000-0x00000000fc326fff   2162688 preemption_desc
22 0x00000000fc327000-0x00000000fc327fff     4096 perfcounter_save_restore_desc
23 0x00000000fc328000-0x00000000fc537fff   2162688 preemption_desc
24 0x00000000fc538000-0x00000000fc538fff     4096 perfcounter_save_restore_desc
25 0x00000000fc539000-0x00000000fc748fff   2162688 preemption_desc
26 0x00000000fc749000-0x00000000fc749fff     4096 perfcounter_save_restore_desc
27 0x00000000fc74a000-0x00000000fc959fff   2162688 preemption_desc
28 0x00000000fc95a000-0x00000000fc95afff     4096 perfcounter_save_restore_desc
29 0x00000000fc95b000-0x00000000fc95bfff     4096 smmu_info
```

The Format of the Scratch Memory

The scratch memory is one-page data that is mapped into the GPU. This allows for some ‘shared' data between the GPU and CPU. For example, it will be used by the GPU to write updated RPTR for each ring buffer. The format of the scratch is:

Offset	Length(Bytes)	Content
0	4*4	RB0 RPTR,RB1 RPTR, RB2 RPTR, RB3 RPTR
0x10	8*4	RB0 Context Restore Address, RB1 Context Restore Address RB2 Context Restore Address, RB3 Context Restore Address

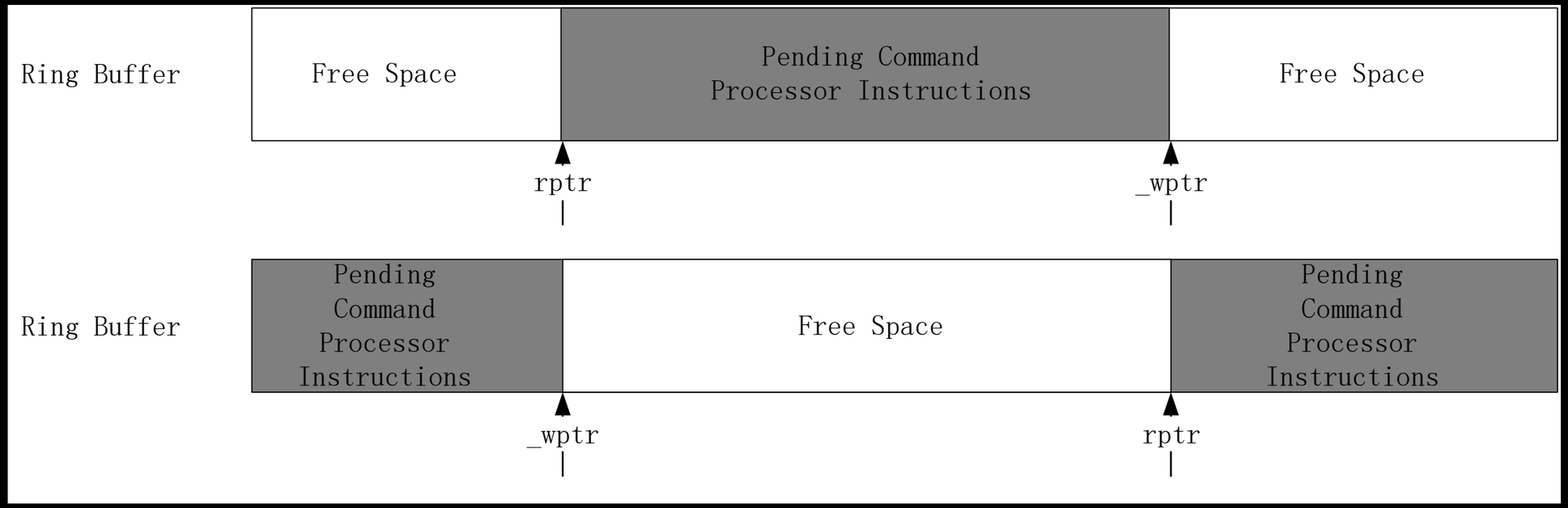
Where is the Bug

```
1 int adreno_ringbuffer_probe(struct adreno_device *adreno_dev, bool nopreempt)
2 {
3     struct kgsl_device *device = KGSL_DEVICE(adreno_dev);
4     struct adreno_gpudev *gpudev = ADRENO_GPU_DEVICE(adreno_dev);
5     int i;
6     int status = -ENOMEM;
7
8     if (!adreno_is_a3xx(adreno_dev)) {
9         status = kgsl_allocate_global(device, &device->scratch, //scratch is allocated as writable by normal Command Processor instructions
10                 PAGE_SIZE, 0, KGSL_MEMDESC_CONTIG, "scratch");
11         if (status != 0)
12             return status;
13     }
14 ...
15 }
```

Where is the Bug

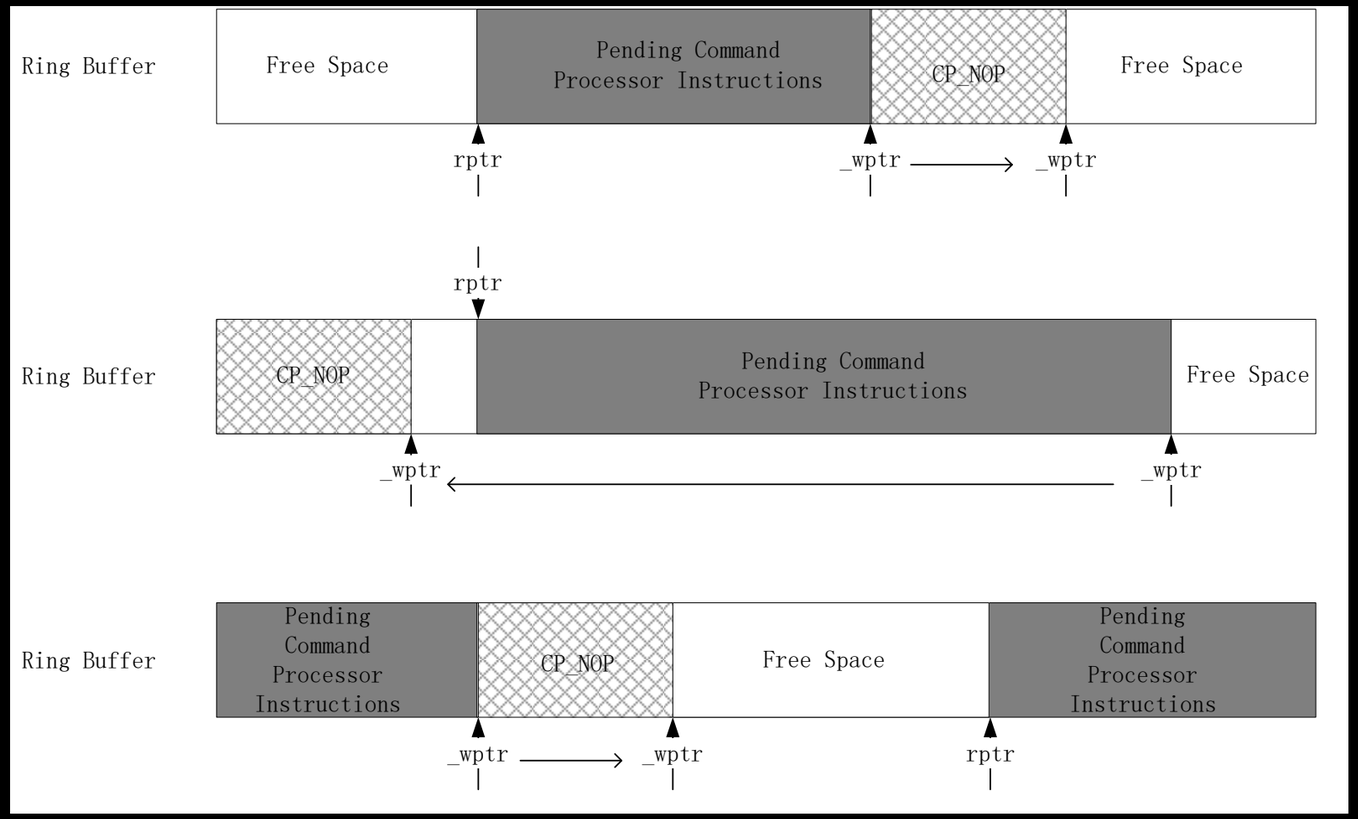
```
1 unsigned int *adreno_ringbuffer_allocspace(struct adreno_ringbuffer *rb,
2         unsigned int dwords)
3 {
4     struct adreno_device *adreno_dev = ADRENO_RB_DEVICE(rb);
5     unsigned int rptr = adreno_get_rptr(rb);   ---------------------------> read rptr from scratch memory
6     unsigned int ret;
7
8     if (rptr <= rb->_wptr) {
9         unsigned int *cmds;
10
11          if (rb->_wptr + dwords <= (KGSL_RB_DWORDS - 2)) {
12              ret = rb->_wptr;
13              rb->_wptr = (rb->_wptr + dwords) % KGSL_RB_DWORDS;
14              return RB_HOSTPTR(rb, ret);
15          } ......
16  }
```

Read And Write Pointer of a Ring Buffer

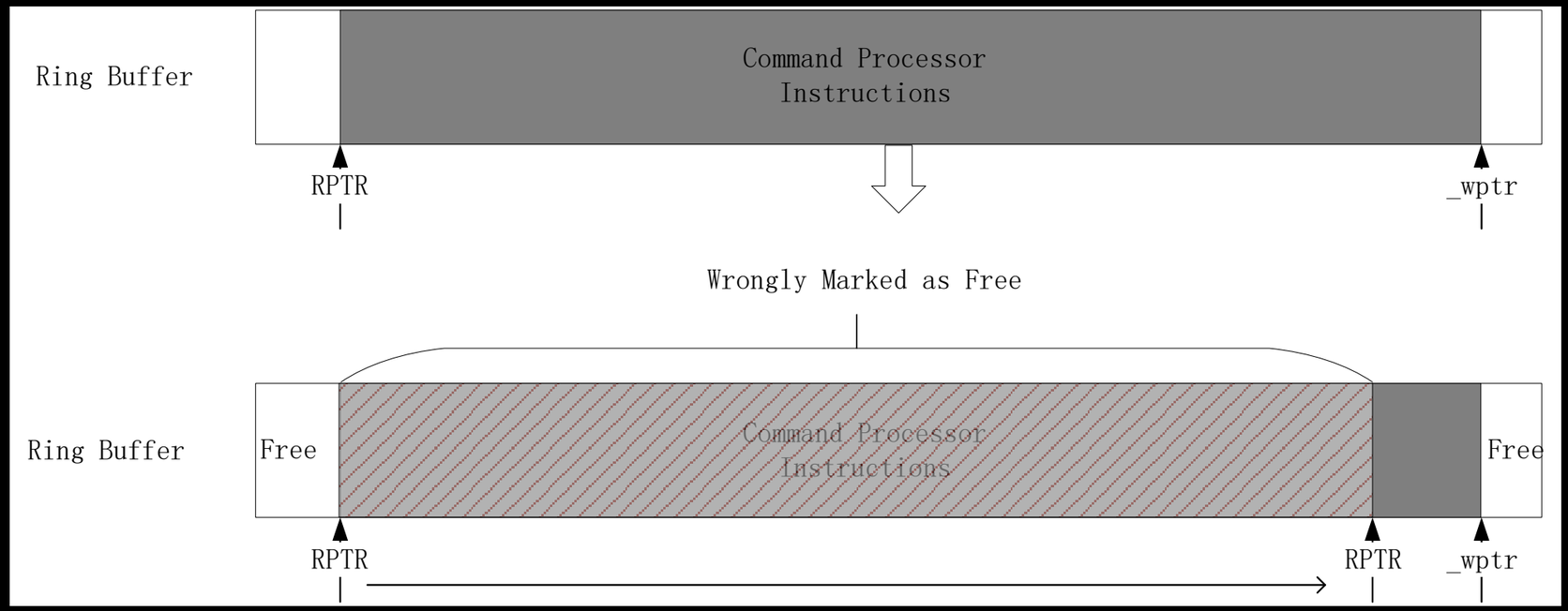


rptr points to the next instruction which will be executed by the GPU
_wptr points to the start of free space of a ring buffer

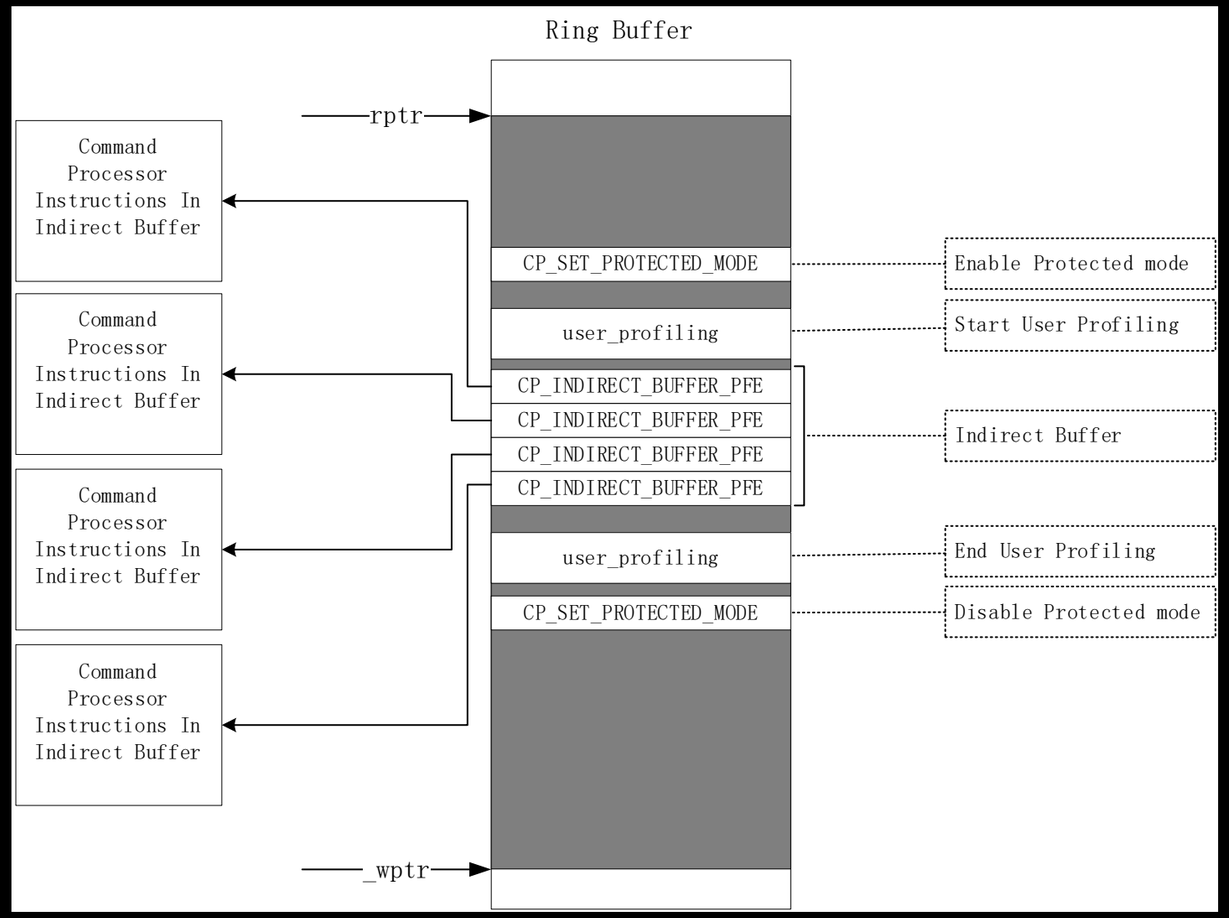
Allocate Space From Ring Buffer



Overwrite Exist Instructions



CP Instruction Sequence of Executing IOCTL_KGSL_GPU_COMMAND

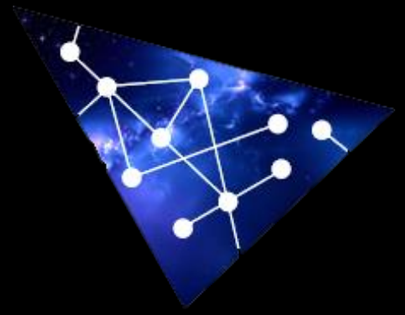


The Process of Exploiting CVE-2019-10567

Demo


ggong@360AlphaTeam:


8:18
Welcome to your new Pixel
Tap for tips



360 ALPHA

Thanks